Building Transparent Models

by

Guang-He Lee

B.S., National Taiwan University (2015)
M.S., National Taiwan University (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2021

Certified by . . . .
Tommi S. Jaakkola
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Building Transparent Models

by

Guang-He Lee

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Transparency has become a key desideratum of machine learning. Properties such as interpretability or robustness are indispensable when model predictions are fed into mission critical applications or those dealing with sensitive/controversial topics (e.g., social, legal, financial, medical, or security tasks). While the desired notion of transparency can vary widely across different scenarios, modern predictors (like deep neural networks) often lack any semblance of this concept, primarily due to their inherent complexity. In this thesis, we focus on a set of formal properties of transparency and design a series of algorithms to build models with these specified properties. In particular, these properties include:

- the model class (of oblique decision trees), effectively represented and trained via a new family of neural models,
- local model classes (e.g., locally linear models), induced from and estimated jointly with a black-box predictor, possibly over structured objects, and
- local certificates of robustness, derived for ensembles of any black-box predictors in continuous or discrete spaces.

The contributions of this thesis are mainly methodological and theoretical. We also emphasize scalability in large-scale settings. Compared to a human-centric approach to interpretability, our methods are particularly suited for scenarios that require factual verification or cases that are challenging to subjectively judge explanations by humans (e.g., for superhuman models).

Thesis Supervisor: Tommi S. Jaakkola
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I thank my advisor, Tommi Jaakkola, for his wonderful mentoring throughout my PhD. He accepted me as a student despite my lack of background and experience in theoretical machine learning, patiently tolerated and answered all my weird questions (right from the first time when I had the pleasure to meet him during my visiting days...), encouraged my sometimes whimsical ideas, challenged my research, writing, and presentations with critical, constructive, professional, and sometimes creative feedback, and educated me to become a competent researcher and thinker. I always learned a lot from and amazed by his unique insights and stunning wisdom.

I thank my previous advisor, Dina Katabi, for her extraordinary mentoring in my first semester and our continued interactions afterwards. I am extremely grateful to her for sharing her kindly advice, infectious enthusiasm, insights, and creative ideas with me. Her hard-working nature left a deep mark on me. Tommi and Dina are undoubtedly the two most influential role models in my PhD.

I thank my thesis committee members Stefanie Jegelka and Luca Daniel for all the stimulating questions and their valuable comments. I am indebted to them for their help in moving forward the entire thesis preparation process so quickly.

I thank my mentor at Google, Been Kim, for investigating brand new research directions with me, for enlightening me with her rich experience in interpretability and machine learning, and for sharing resources and opportunities with me.

I thank my mentor at Citadel, Yu Xin, for introducing me to the world of finance and hedge funds, for guiding me with his solid research experience in machine learning and financial industry, and for all the career opportunities and advice.

I thank all my collaborators throughout my PhD: David Alvarez Melis, Shiyu Chang, Hao He, Rumen Hristov, Chen-Yu Hsu, Wengong Jin, Tzyy-Shyang Lin, Jiaye Teng, Yonglong Tian, Hao Wang, Yang Yuan, Abbas Zeitoun, and Mingmin Zhao. I am particularly grateful to David for his guidance and for our early collaborations which greatly helped me during my transition to Tommi's group. I am also very grateful to Yang for his friendship as well as for our many discussions.

I thank Tommi's group members (Gary Bécigneul, Xiang Fu, Andreea Gane, Octavian Ganea, Vikas Garg, Timur Garipov, John Ingraham, Bracha Laufer, Paresh Malalur, Jonas Mueller, Amit Schechter, Tianxiao Shen, Shangyuan Tong, Shubhendu Trivedi, Tian Xie, and Yilun Xu), Dina's group members (Omid Abari, Zachary Kabelac, Tianhong Li, Ezzeldin O. Hamed, Deepak Vasisht, Shichao Yue, and Guo Zhang), and Regina's group members (Yujia Bao, Benson Chen, Jiang Guo, Jiaming Luo, Yujie Qian, Tal Schuster, Darsh Shah, and Rachel Wu) for all the delightful conversations. I am also thankful to the lab assistants Teresa Cataldo and Mary McDavitt.

I thank all my friends (too many to list!) for all their help during my PhD, for their company and emotional support, especially throughout the COVID-19 pandemic. I am especially grateful to John Lai, Rachel, and Shubhendu for helping me edit parts of this thesis. I also thank my families for the unconditional support.

Finally, I thank Siemens Corporation as well as the MIT-IBM collaborations for supporting different portions of the thesis work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 From Interpretability to Transparency

Modern machine learning tasks are increasingly complex, requiring flexible models with large numbers of parameters, such as deep networks. Such modeling gains often come at the cost of interpretability—these deep models, effectively black-boxes, provide no insight on their inner workings beyond the final prediction. This is a serious issue when the predictions are relied on in subsequent mission-critical applications such as social, legal, security, financial, or medical problems, where the ability to justify *why* and/or *how* the predictions are made is nearly as important as the raw predictive power. Indeed, explaining model predictions and/or factually verifying properties of the model (e.g., robustness) is becoming a key part of any data-driven decision making. For example, in order to reduce the risk of potentially catastrophic outcomes, an automatic medical diagnosis system should operate in an *explainable* and *reliable* fashion. Interpretability challenges become more pronounced in interactive settings, and include intricate issues such as *fairness* or *privacy*. Explainable AI has received much attention in society; for instance, European Parliament passed a law that establishes a "right to explanation" on algorithmic decisions [112]. Thus, the development of interpretable machines is a critical and pressing demand from society.

The challenge is that interpretability is hard to define in the first place. It involves humans' perception and understanding of explanations. Even in the psychology literature, fundamental questions of interpretability such as what constitutes an explanation to us are barely answered [92]. Moreover, interpretability is notoriously subjective: whether an explanation is interpretable or not can depend on the background and/or preference of the user with respect to the application scenario. For example, a novel Go strategy can be totally unintelligible to novice players, or it can be underappreciated for stubborn players who only learn from well-established results or their own experience, but it can be valuable for professional players who are excited about new explorations. Similar scenarios appear when it comes to explanations of complex models.

One way to circumvent the ill-specified definition of interpretability is to include humans in the loop [148]. For instance, we may ask a user to judge interpretability (e.g., whether an explanation of a model is interpretable). For instance, we may ask a user to select amongst a set of alternatives the model that is the most interpretable to her [78]. The problem is similar to, e.g., metric elicitation in fairness [61]. However, the subjectivity remains and the calibrated interpretability may not transfer to other users easily. Furthermore, assessing interpretability based on human feedback can be misleading. Indeed, an explanation (e.g., visualization) can be considered interpretable without bearing any relevance to the actual model.

Another major direction in literature is to first define certain explanations that are deemed intelligible, and then develop a method that distills the model into the specified form of explanations. Popular examples include exploiting input gradients to identify salient features [139, 132] and utilizing influence functions to find influential training examples [75]. The apparent advantage in this direction is its unrestricted nature: one can apply the method to almost every (potentially black-box) model. However, since the approximations taken during the distillation can easily distort the behavior of the model, it is not guaranteed that the explanation would preserve model behavior. Indeed, it has been observed that these kinds of post-hoc interpretability methods often lead to unstable explanations [4, 48].

Yet another direction is to embed the explanation mechanism directly into the model such that the model is self-explaining. This includes classic methods such as linear models, decision trees, and k-nearest neighbors. However, these methods are known to be limited in capacity, so there are a few recent efforts aiming to equip modern complex predictors with similar capability, typically via a customized architectural design [88, 141, 76] or as a regularization problem [5, 157]. For example, one can re-organize a neural network into a tree structure [76, 141] and interpret the resulting model as a decision tree.

In this thesis, we emphasize theoretical guarantees: the model must exhibit a stated interpretability property. For example, a property may ensure the model can be approximated (and thus explained) by a decision tree with a small, bounded error. This direction could be classified under a more general notion of transparency, which refers to properties that are revealed or imposed on the model. Transparency also encompasses other desirable properties. For instance, we need *robustness* of a security system (e.g. face identification) to protect the users from potential threats. Likewise, to prevent a judicial system from making biased actions towards an underrepresented group, *fairness* must be ensured. Finally, to prevent a medical diagnosis system from leaking sensitive information, *privacy* should be taken into account in the system. There are many other similar properties, such as *calibration* and *causality*. With transparency, stakeholders will have some guarantees about the models, and are thus able to make safe and appropriate interactions with them.

While many properties may be deemed beneficial, one may favor different properties depending on the uses or needs in the application. In interpretability, explaining prediction errors in terms of training examples [75, 161] is helpful when training data are noisy, while investigating model internals [71] is more important if the data are already carefully curated. Similarly, one might care about robustness to adversarial manipulation of input features [16, 140], which would fool the classifier, in security applications. In other cases, being robust to (training) data poisoning [10, 111] is more valuable, particularly if the test environment is properly controlled but the training data are provided by untrusted users. Likewise, common users accept systems where sensitive information is simply hidden, but diligent users would require that sensitive information cannot be inferred [65]. It is possible to exhibit multiple properties at the same time, but some of the properties can be potentially conflicting. For example, is has been shown that some existing fairness criteria (equalized odds [56], statistical parity, and predictive parity [25]) cannot be satisfied simultaneously, except for some highly constrained cases [74].

As interpretability and transparency are related, we discuss a few distinctions for clarity. Interpretability requires explainable presentation of the model, while transparency refers to stated properties that we wish to expose about the models. A model can be transparent about properties (e.g., robustness) without being interpretable. Of course, it is desirable to have a model that is both human interpretable and transparent, so we can understand both its interpretability and general properties. The co-existence typically relies on a simple architecture (e.g., linear models or decision trees) that makes the model easy to understand and immediate in terms of properties. Generalizing the setting towards more complex models is a key goal.

In this thesis, we propose several strategies towards this challenge.

- Existing interpretable and transparent models like decision trees and linear models work well for simple tasks but not complex problems due to their limited capacity. To enable their usage in these complex problems, we propose a divide-and-conquer strategy: we can first decompose the data into smaller groups and then use a local transparent model for each group. For example, while a shallow decision tree is not powerful enough for predicting chemical properties for all the molecules, it may be sufficient if it only handles a group of molecules with similar structural characteristics. However, a straightforward implementation suffers from a serious cold-start problem: it has zero generalization ability for groups with no training data. As an alternative, we propose to induce these local models from a powerful predictor, with a joint training procedure ensuring coherence among the locals models, the powerful predictor, and the data.
- Alternatively, we may also try to make these simple models more powerful while maintaining some facets of transparency or interpretability. For example, Alvarez-Melis and Jaakkola proposed to generalize linear models $\boldsymbol{\theta}^\top \boldsymbol{x}$ to deep

linear models $\boldsymbol{\theta}(\boldsymbol{x})^\top \boldsymbol{x}$ [5], where the linear coefficients $\boldsymbol{\theta}(\boldsymbol{x})$ are data-dependent and generated by a deep network. The authors also utilized a regularization term to ensure the local stability of data-dependent linear weights, leading to a model that is approximately locally linear, similar to the aforementioned locally transparent models. In this approach, the model always exercises a linear model when making the prediction for every single input $\boldsymbol{x}$, thus readily interpretable; the model is still powerful since it is allowed to use different linear weights $\boldsymbol{\theta}(\boldsymbol{x})$ for distinct input values $\boldsymbol{x}$. In this thesis, as a different direction, we focus on generalizing decision trees with more expressive yet interpretable decisions.

- We also investigate the broader transparency problem to reveal and/or ensure a chosen property in a given model. There are many plausible properties, including robustness, fairness, privacy, causality, and calibration. We focus on robustness to adversarial examples [16, 140], which are crafted examples that manipulate model predictions with slight, potentially unnoticeable perturbations from actual examples. Their existence makes a machine learning system particularly vulnerable since it is challenging to judge whether an error should be attributed to an inherent generalization error or an adversarial attack. Here the property can be naturally verified via certificates, guarantees that ensure no adversarial example exists. Although the exact (i.e., the best possible) certification of adversarial robustness is generally NP-complete [70], we can trade off efficiency with tightness. Our goal is to make certification efficient for a variety of models and scenarios (continuous and discrete data) while maintaining tightness with respect to the chosen family of models.

Technically, we accomplish these strategies in a perhaps quite counterintuitive way; we leverage complex models such as deep networks as a backbone tool for the above strategies, despite the fact that these models are recognized as un-interpretable or non-transparent. For example, we leverage a black-box neural network to induce locally transparent models. Besides, our method also inherits several benefits of these models, such as simple yet effective training procedures, compatibility with structured data, flexible architectural choices, the capability of learning representations, and, most notably, state-of-the-art performance. A more in-depth overview of our technical solution is provided in the next section.

1.2 Overview of this Thesis

This thesis addresses the emerging problem of understanding machine learning models. This problem is strongly motivated by the need of ensuring the models work properly in mission critical applications. The form of understanding also has multiple facets. It can be as simple as highlighting prediction rationales or providing explanations (i.e., interpretability), or more subtle, as revealing high-level properties that describe the way in which the model operates (i.e., transparency). Simple models like linear models or decision trees are both interpretable and transparent but limited in expressiveness. The majority of this thesis is dedicated to generalizing these simple models in a way that still preserves the interpretability and transparency. We also touch upon the more general transparency problem, and focus on the certification of adversarial robustness.

1.2.1 A Game-Theoretic Approach to Local Transparency

Chapter 2 provides a novel approach to training local transparent models that generalize to local regions outside the training data. Our approach naturally operates over structured data and functionally tailors the local models towards powerful predictors (cf. the raw data). The estimation problem is set up as a co-operative game between an unrestricted *predictor* such as a neural network, and a *witness* chosen from the desired transparent family. The goal of the witness is to mimic the predictor, locally, with the chosen family of functions, while the predictor is trained to fit the data while maintaining local approximability with respect to the witness. We emphasize that the witness remains globally powerful as it is only restricted to exercise transparent behaviors locally. We analyze the effect of the proposed game, provide example formulations in the context of graph and sequence networks versus decision trees and linear models, and empirically illustrate the idea in chemical property prediction, temporal modeling, and molecule representation learning.

This chapter is based on the publication [85]:

G.-H. Lee, W. Jin, D. Alvarez-Melis, and T. S. Jaakkola. Functional Transparency for Structured Data: a Game-Theoretic Approach. In *International Conference on Machine Learning (ICML)*, 2019.

1.2.2 Towards Robust, Locally Linear Models

Although the method in Chapter 2 is compatible with general transparent model classes, the method is not efficient for representing locally linear models. Indeed, $\Omega(D^3)$ time is generally required to find a local linear model in D dimensional space. Instead, we observe that neural networks with piecewise linear activation functions naturally realize locally linear models, and a gradient evaluation immediately provides the associated local linear model. However, one key challenge is that such derivatives (and thus the resulting linear models) are themselves inherently unstable.

In Chapter 3, we propose a new learning problem to encourage these neural networks to have stable derivatives over larger regions. At the heart of our learning algorithm are an inference step that identifies a region around a point where linear approximation is provably stable, and an optimization step to expand such regions. We propose a novel relaxation to scale the learning method to large-scale models, and a new perturbation algorithm to speed-up the computation. We illustrate our method with residual and recurrent networks on image and sequence datasets.

This chapter is based on the publication [83]:

G.-H. Lee, D. Alvarez-Melis, and T. S. Jaakkola. Towards Robust, Locally Linear Deep Networks. In *International Conference on Learning Representations (ICLR)*, 2019.

1.2.3 Oblique Decision Trees from Neural Networks

A natural generalization of decision trees via linear models is known as oblique decision trees, which extend the original coordinate cuts in the decision nodes to linear classifications. However, the slight generalization leads to a much challenging optimization problem, where even the greedy induction algorithm becomes intractable.

Chapter 4 presents a novel neural architecture, which we call locally constant networks, that is representationally equivalent to the family of oblique decision trees. Moreover, we highlight several advantageous properties of locally constant networks, including how they realize decision trees with parameter sharing across branching or leaves. Indeed, only $\tilde{M}$ neurons suffice to implicitly model an oblique decision tree with $2^{\tilde{M}}$ leaf nodes. The neural representation also enables us to adopt many tools developed for deep networks (e.g., DropConnect [149]) while implicitly training decision trees. We demonstrate that our method outperforms alternative techniques for training oblique decision trees in the context of molecular property classification and regression tasks.

This chapter is based on the publication [84]:

G.-H. Lee and T. S. Jaakkola. Oblique Decision Trees from Derivatives of ReLU Networks. In *International Conference on Learning Representations (ICLR)*, 2020.

1.2.4 Tight Certificates of Adversarial Robustness

It has been discovered that strong theoretical guarantees of adversarial robustness can be given for ensembles of (possibly black-box) classifiers generated by input randomization. Specifically, an ℓ_2 bounded adversary cannot alter the ensemble prediction generated by an additive isotropic Gaussian noise [29], where the radius for the adversary depends on both the variance of the distribution as well as the ensemble margin at the point of interest.

In Chapter 5, we build on and considerably expand this work across broad families of distributions (e.g. uniform). In particular, we offer adversarial robustness guarantees and associated algorithms for the discrete case where the adversary is ℓ_0

bounded. Moreover, we exemplify how the guarantees can be tightened with specific assumptions about the function class of the classifier such as a decision tree. We empirically illustrate these results with deep networks and decision trees across image and molecule datasets, respectively.

This chapter is based on the publication [86]:
G.-H. Lee, Y. Yuan, S. Chang, and T. S. Jaakkola. Tight Certificates of Adversarial Robustness for Randomly Smoothed Classifiers. In *Neural Information Processing Systems (NeurIPS)*, 2019.

1.3 Other Previously Published Work

This thesis includes a coherent line of the author's work during his PhD. The author has also investigated other research areas pertaining to intelligent wireless sensing [142, 63], generative models [57], and self-supervised learning [64], which are not included in this thesis, but the details can be found in the bibliography.

Chapter 2

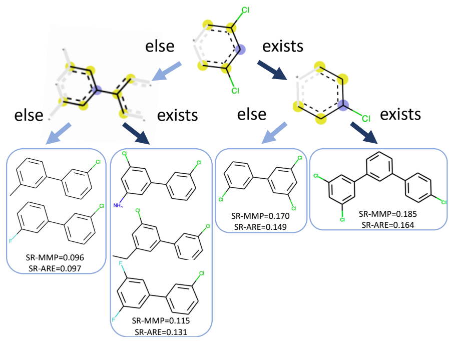
Local Transparency for Structured Data: a Game-Theoretic Approach

2.1 Introduction

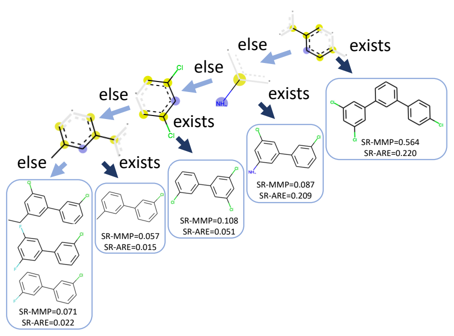
It is possible to realize a well-understood (i.e., transparent) yet flexible model. For instance, the 1-Lipschitz discriminator used to realize the Wasserstein-1 distance [6] can be as complex as a neural network. Adherence to a complex, global functional class is not the only way to achieve transparency. For example, linearity is a desirable characteristic for transparency but is sensible to enforce only locally. We offer therefore a new notion of transparency—local transparency—where the goal is to restrict models to adopt a desirable local behavior while guiding them to be more flexible globally.

Previous approaches to interpretability have mainly focused on models that operate on fixed-size data, such as scalar-features [79] or images [125, 96]. The emphasis has been on feature relevance or selection [121]. Recent methods do address some of the challenges in sequential data [88, 8, 3], primarily in natural language processing tasks where the input sequence is discrete. Interpretability for continuous temporal data [2, 157] or graph structures remains largely unexplored.

We develop a novel approach to transparency that is naturally suited for structured data. At the core of our approach is a game-theoretic definition of transparency.



(a) The explanation from our model (trained for transparency).



(b) The explanation from a normally trained model.

Figure 2.1.1: During testing, we fit decision trees to our predictor (i.e., finding the witness) and an unregularized model on molecule property prediction at the same local neighborhood such that the functional approximations are comparable in AUC (because the scale is not crucial). The split criterion on each node is based on the existence of a *complete chemical substructure* in Morgan fingerprints [122]. The color of each Morgan fingerprint simply reflects the radius of the fingerprint.

This is set up as a two-player co-operative game between a *witness* and a *predictor*. The witness is chosen from a simple transparent family whereas the predictor is unrestricted. Transparency arises from the fact that the witness always exercises simple behavior in each local region, while the co-operative game ensures *consistency* between the witness and the expressive predictor so as to make the witness powerful globally. Rather than directly fitting the data, exploiting the predictor as a reference allows the witness to generalize to local regions containing no training data. Alternatively, if transparency is only required approximately, our method also produces the predictor with such guarantees. The approach differs from global regularization of models towards interpretability [157], models that are constructed a priori to be interpretable in terms of the global function class [18], or from post-hoc explanations of black-box methods via local perturbations [121, 3].

To illustrate, we contrast our approach with methods that seek to obtain interpretable explanations after the fact (e.g., Ribeiro *et al.* [121]). Derived explanation after training can be misleading in some cases if the explanation does not match the functional behavior of the model. For example, Figure 2.1.1 shows our local decision tree witness (a, left) versus the local decision tree approximation to an unregularized model (b, right). The tree for the unregularized model only filters one sample in each split, lacking generality to explain the (local) behavior. This phenomenon is related to unstable explanations that arise with already trained models [4, 48].

The game-theoretic approach is very flexible in terms of models and scenarios. We therefore illustrate the approach across a few novel scenarios: exemplifying graph predictors via decision trees, revealing temporal variation of an autoregressive generative model, and instantiating the encoder in unsupervised graph representation learning using simple decision rules. Our main contributions are:

- A novel game-theoretic approach to transparency, applicable to a wide range of models, architectures, and local transparency classes, without requiring differentiability.
- Analysis on the effective size of the local regions and establishing equilibria pertaining to different game formulations.
- Illustration of transparent models across several tasks, from chemical property prediction, physical component modeling, to molecule representation learning.

2.2 Related Work

Our goal is to realize a model that is expressive yet exhibiting a desired local behavior. The approach confers an operational guarantee rather than directly interpretability. In contrast, examples of archetypal interpretable models include linear classifiers, decision trees [115], and decision sets [79]; recent approaches also guide complex models towards highlighting pieces of input used for prediction [88], learning representations that can be decomposed among training examples [161], generalizing linear models while maintaining interpretability [5], or finding interpretable partial substitutes for complex models [150]. A model conforming to a known functional behavior, at least locally, as in our approach, is not necessarily itself human-interpretable. The guarantee we offer is that the complex model indeed follows such a behavior.

Previous work on approximating a functional class (via neural networks) can be roughly divided into two types: parametrization-based and regularization-based methods. Works in the first category seek self-evident adherence to a functional class, which include maintaining Lipschitz continuity via weight clipping [6], orthogonal transformation via scaled Cayley transform of skew-symmetric matrices [60], and "stable" recurrent networks via spectral norm projection on the transition matrix [98].

A softer approach is to introduce a regularization problem that encourages neural networks to match properties of the functional class. Such regularization problem might come in the form of a gradient penalty as used in several variants of generative adversarial networks [55, 11, 102] under the framework of integral probability metrics [103], layer-wise regularization of transformation matrices [27] towards parseval tightness [77], and recent adversarial approaches to learning representations for certain independence statements [47, 164]. Typically, a tailored regularization problem is introduced for each functional class.

Our work provides a unique combination of the two directions. On one hand, the witness strictly adheres to the functional class locally. On the other hand, we cast the overall problem as a regularization problem. However, we focus on transparency and our approach—a general co-operative game—is quite different. Our methodology is applicable to any choice of (local) functional class without any architectural restrictions on the deep model whose behavior is used to guide the witness. The optimization of functional deviation in the game must remain tractable, of course.

2.3 Methodology

In this work, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N \subset \mathcal{X} \times \mathcal{Y}$, we learn a (supplementary) predictive function $f \in \mathcal{F} : \mathcal{X} \to \mathcal{Y}$ together with a transparent—and usually simpler—function $g \in \mathcal{G} : \mathcal{X} \to \mathcal{Y}$ defined over a functional class $\mathcal{G}$. We refer to functions f and g as the *predictor* and the *witness*, respectively, throughout the chapter. Note that we need not make any assumptions on the functional class $\mathcal{F}$, instead allowing a flexible class of predictors. In contrast, the family of witnesses $\mathcal{G}$ is strictly constrained to be a *transparent* functional set, such as the set of linear functions or decision trees. We assume to have a deviation function $d : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ such that $d(y, y') = 0 \iff y = y'$, which measures discrepancy between two elements in $\mathcal{Y}$ and can be used to optimize f and g. To simplify the notation, we define $\mathcal{D}_x \triangleq \{x_i : (x_i, y_i) \in \mathcal{D}\}$. We introduce our game-theoretic framework in this section, analyze it in §2.4, and instantiate the framework with concrete models in §2.5.

2.3.1 Game-Theoretic Consistency

There are many ways to distill a witness function $g \in \mathcal{G}$ from the predictor f by means of discrepancy measures. However, since the witness function can be weak such as a linear function, we cannot expect that a reasonable predictor would agree to it globally. Instead, we make a slight generalization to enforce this criterion only locally, over different sets of neighborhoods. To this end, we define *local consistency* by measuring how close f is to the family $\mathcal{G}$ over a local neighborhood $\mathcal{B}(x_i) \subset \mathcal{X}$ around an observed point x_i. One straightforward instantiation of such a neighborhood $\mathcal{B}(x_i)$ in temporal domain will be simply a local window of points $\{x_{i-\epsilon}, \ldots, x_{i+\epsilon}\}$. Our resulting local discrepancy measure is

$$\min_{g \in \mathcal{G}} \frac{1}{|\mathcal{B}(x_i)|} \sum_{x_j \in \mathcal{B}(x_i)} d(f(x_j), g(x_j)). \tag{2.1}$$

The summation can be replaced by an integral when a continuous neighborhood is used. The minimizing witness function, $\hat{g}_{x_i}$, is indexed by the point x_i around which it is estimated; depending on the function f, the minimizing witness can change from one neighborhood to another. If we view the minimization problem game-theoretically, $\hat{g}_{x_i}$ is the *best response strategy* of the local witness around x_i.

The local discrepancy measure can be incorporated into an overall estimation criterion in many ways so as to guide the predictor towards the desired functional form, thus accurately inducing witnesses. The guidance can be offered as a *uniform* constraint with a permissible δ-margin, as an additive *symmetric* penalty, or defined *asymmetrically* as a game-theoretic penalty where the information sets for the predictor and the witness are no longer identical. We consider each of these in turn.

2.3.2 Uniform Criterion

A straightforward formulation is to confine f to remain within a margin δ of the best fitting witness for every local neighborhood. Assume that a primal loss $\mathcal{L}(\cdot,\cdot)$ is given for a learning task. The criterion imposes the δ-margin constraint uniformly as

$$\sum_{(x_i,y_i)\in\mathcal{D}} \mathcal{L}(f(x_i),y_i), \quad s.t. \min_{g\in\mathcal{G}} \frac{1}{|\mathcal{B}(x_i)|} \sum_{x_j\in\mathcal{B}(x_i)} d(f(x_j),g(x_j)) \le \delta, \forall x_i \in \mathcal{D}_x. \qquad (2.2)$$

We assume that the optimal g with respect to each constraint may be efficiently found due to the simplicity of $\mathcal{G}$ and the regularity of $d(\cdot,\cdot)$. We also assume that the partial derivatives with respect to f, for fixed witnesses, can be computed straightforwardly under sufficiently regular $\mathcal{L}(\cdot,\cdot)$ in a Lagrangian form. In this case, we can solve for f, local witnesses, and the Lagrange multipliers using the mirror-prox algorithm [107].

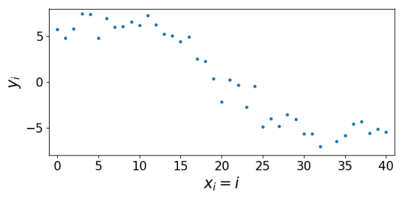
The hard constraints in the uniform criterion will lead to strict consistency between the witness and the predictor. However, the effect may be undesirable in some cases where the observed data (thus the predictor) do not agree with the witness in all places; the resulting loss of performance may be too severe. Instead, we can enforce the agreement with local witnesses to be small in aggregate across neighborhoods.

2.3.3 Symmetric Game

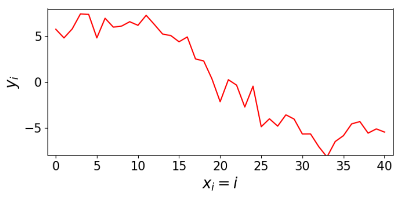
We define an additive, unconstrained, symmetric criterion to smoothly trade off between performance and consistency. The resulting objective is

$$\sum_{(x_i,y_i)\in\mathcal{D}} \left[\mathcal{L}(f(x_i),y_i) + \min_{g\in\mathcal{G}} \frac{\lambda}{|\mathcal{B}(x_i)|} \sum_{x_j\in\mathcal{B}(x_i)} d(f(x_j),g(x_j)) \right] \qquad (2.3)$$

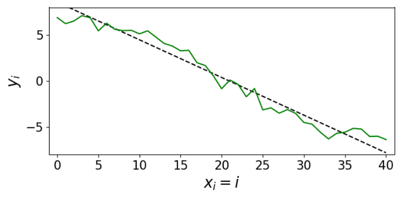
To illustrate the above idea, we generate a synthetic dataset to show a *neighborhood* in Figure 2.3.1a with a piecewise linear predictor $f \in \mathcal{F}_{\text{piecewise linear}}$ in Figure 2.3.1b. Clearly, f does not agree with a linear witness within this neighborhood. However, when we solve for f together with a linear witness $g_{x_i} \in \mathcal{G}_{\text{linear}}$ as in Figure 2.3.1c, the resulting function has a small residual deviation from $\mathcal{G}_{\text{linear}}$, more



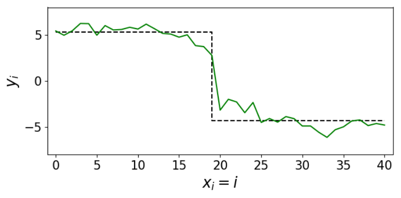
(a) Neighborhood $\mathcal{B}(x_{20})$



(b) $f \in \mathcal{F}_{\text{piecewise linear}}$



(c) $g_{x_{i=1}} \in \mathcal{G}_{\text{linear}}$



(d) $g_{x_{i=1}} \in \mathcal{G}_{\text{decision stump}}$

Figure 2.3.1: Examples of fitting a neighborhood $\mathcal{B}(x_{20})$ (2.3.1a) with a piecewise linear predictor (2.3.1b). Using different witness families (Figs. 2.3.1c&2.3.1d, dashed lines) leads to predictors (solid green) with different behaviors, despite yielding the same error (mean squared error = 1.026).

adhering to the linear functional class while still closely tracking the observed data. Figure 2.3.1d highlights the flexibility of our framework where a very different functional behavior can be induced by changing the functional class of the witness.

2.3.4 Asymmetric Game

Solving the symmetric criterion can be computationally inefficient since the predictor is guided by its deviation from each of the local witnesses on all points within each of the local neighborhoods. Moreover, the predictor value at any point x_i is subject to potentially conflicting regularization terms across the neighborhoods, which is undesirable. The inner summation in Eq. 2.3 may involve different sizes of neighborhoods $\mathcal{B}(x_i)$ (e.g., end-point boundary cases) and this makes it more challenging to parallelize the computation.

We would like to impose even functional regularization at every $f(x_i)$ based on how much the value deviates from the witness associated with the local region $\mathcal{B}(x_i)$. This approach leads to an *asymmetric* co-operative formulation, where the information sets for the predictor f and local witnesses g_{x_i} differ. Specifically, the local best-response witness $\hat{g}_{x_i}$ is chosen to minimize the local discrepancy as in Eq. (2.1), and thus depends on f values within the whole region; in contrast, the predictor f only receives feedback in terms of the resulting deviation at x_i, only seeing $\hat{g}_{x_i}(x_i)$. From the point of view of the predictor f, the best response strategy is obtained by minimizing

$$\sum_{(x_i,y_i)\in\mathcal{D}} \Big[\mathcal{L}(f(x_i), y_i) + \lambda\, d(f(x_i), \hat{g}_{x_i}(x_i))\Big] \tag{2.4}$$

To train the proposed method, we perform alternating updates for $f(\cdot)$ and $\hat{g}_{x_i}(\cdot)$ on their respective criteria. Note that in this case the objective cannot be written as a single minimization problem (different information sets) but can be still interpreted as a game. The deviation between the predictor and the witness, $d(\cdot,\cdot)$, can also be defined asymmetrically as we do in §2.5.3.

2.4 Analysis

We consider here the effectiveness of regularization in relation to the neighborhood size and establish fixed point equations for the predictor under the three estimation criteria. For simplicity, we assume $\mathcal{X} = \mathbb{R}^D$ and $\mathcal{Y} = \mathbb{R}$, but the results are generalizable to our examples in §2.5. All the proofs are in Appendix 2.A.

2.4.1 Neighborhood Size

The formulation involves a key trade-off between the size of the region where the function should be simple and the overall accuracy achieved by the predictor and thus the witness. When the neighborhood is too small, local witnesses become perfect, inducing no regularization on f and yielding no effective explanations. Thus the size of the region is a key parameter. A neighborhood size is sufficient if the witness class $\mathcal{G}$ cannot readily overfit f values within the neighborhood. Formally,

Definition 2.1. *We say that a neighborhood size m is* effective *for $\mathcal{G}$ if for any $f \notin \mathcal{G}$*

we can find $\mathcal{B} \subset \mathcal{X} : |\mathcal{B}| = m$ *s.t.*

$$\min_{g \in \mathcal{G}} \frac{1}{m} \sum_{x \in \mathcal{B}} d(f(x), g(x)) > 0. \tag{2.5}$$

A trivial example is when $\mathcal{G}$ is the constant class, a neighborhood size m is effective if $m > 1$. Note that the neighborhood $\mathcal{B}$ in the above definition can be any finite collection of points $\mathcal{B}(\cdot)$. For example, the points in the neighborhood induced by a temporal window $\{x_{i-\epsilon}, \dots, x_{i+\epsilon}\}$ need not remain in a small ℓ_p-norm ball.

For linear models and decision trees, we have

- $D + 1$ is the tight lower bound on the effective neighborhood size for the linear class.
- $2^k + 1$ is a lower bound on the effective neighborhood size for decision trees with depth bounded by k.

When the sample size within a neighborhood falls below the bounds, regularization can still be useful if the witness class is not uniformly flexible or if the algorithm for finding the witness is limited (e.g., greedy induction for decision trees).

2.4.2 Equilibrium Solutions

The symmetric game constitutes a standard minimization problem, but the existence or uniqueness of equilibria under the asymmetric game is not obvious. Our main results in this section make the following assumptions.

(A1) the predictor f is unconstrained.

(A2) both the loss and deviation are squared errors.

(A3) $|\mathcal{B}(x_i)| = m, \forall x_i \in \mathcal{D}_x$.

(A4) $x_j \in \mathcal{B}(x_i) \implies x_i \in \mathcal{B}(x_j), \forall x_i, x_j \in \mathcal{D}_x$.

(A5) $\cup_{x_i \in \mathcal{D}_x} \mathcal{B}(x_i) = \mathcal{D}_x$.

We note that **(A3)** and **(A4)** are not technically necessary but simplify the presentation. We denote the predictor in the uniform criterion (Eq. (2.2)), the symmetric game (Eq. (2.3)), and the asymmetric game (Eq. (2.4)) as f_U, f_S, and f_A, respectively. We use $X_i \in \mathbb{R}^{m\times D}$ to denote the neighborhood $\mathcal{B}(x_i) = \{x'_1, \ldots, x'_m\}$ ($X_i = [x'_1, \ldots, x'_m]^\top$), and $f(X_i) \in \mathbb{R}^m$ to denote the vector $[f(x'_1), \ldots, f(x'_m)]^\top$. $X_j^\dagger$ denotes the pseudo-inverse of X_j. Then we have

Theorem 2.2. *If* **(A1-5)** *hold and the witness is in the linear family, the optimal f_S satisfies*

$$f_S^*(x_i) = \frac{1}{1+\lambda}\left[y_i + \frac{\lambda}{m}\Big(\sum_{x_j \in \mathcal{B}(x_i)} X_j^\dagger f_S^*(X_j)\Big)^\top x_i\right],$$

and the optimal f_A, at every equilibrium, is the fixed point

$$f_A^*(x_i) = \frac{1}{1+\lambda}\left[y_i + \lambda(X_i^\dagger f_A^*(X_i))^\top x_i\right], \forall x_i \in \mathcal{D}_x.$$

The equilibrium in the linear class is not unique when the witness is not fully determined in a neighborhood due to degeneracy. To avoid these cases, we can use Ridge regression to obtain a stable equilibrium (discussed also in Appendix).

A special case of Theorem 2.2 is when $x_i = [1], \forall x_i \in \mathcal{D}_x$, which effectively yields the equilibrium result for the constant class; we found it particularly useful to understand the similarity between the two games in this scenario. Concretely, each $X_j^\dagger f(X_j) x_i$ becomes $\frac{1}{m}\sum_{x_k \in \mathcal{B}(x_j)} f(x_k)$, and the solutions for both the symmetric and asymmetric games induce the optimal predictors as recursive convolutional averaging of neighboring points with the same decay rate $\lambda/(1+\lambda)$, while the convolutional kernel evolves twice as fast in the symmetric game than in the asymmetric game.

Next, we show that the uniform criterion yields a very different equilibrium.

Theorem 2.3. *If* **(A1-5)** *hold and the witness is in the linear family, the optimal f_U satisfies*

$$f_U^*(x_i) = \begin{cases} \alpha(x_i, f_U^*), & \text{if } \alpha(x_i, f_U^*) > y_i, \\ \beta(x_i, f_U^*), & \text{if } \beta(x_i, f_U^*) < y_i, \\ y_i, & \text{otherwise,} \end{cases}$$

for $x_i \in \mathcal{D}_x$, where

$$\alpha(x_i, f_U^*) = \max_{x_j \in \mathcal{B}(x_i)} \left[(X_j^\dagger f_U^*(X_j))^\top x_i - \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j) \setminus \{x_i\}} (f_U^*(x_k) - (X_j^\dagger f_U^*(X_j))^\top x_k)^2} \right];$$

$$\beta(x_i, f_U^*) = \min_{x_j \in \mathcal{B}(x_i)} \left[(X_j^\dagger f_U^*(X_j))^\top x_i + \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j) \setminus \{x_i\}} (f_U^*(x_k) - (X_j^\dagger f_U^*(X_j))^\top x_k)^2} \right].$$

A noticeable difference from the games is that, under the uniform criterion, the optimal predictor $f_U^*(x_i)$ may faithfully output the actual label y_i if the functional constraint is satisfied, while the functional constraints are translated into a "convolutional" operator in the games.

2.4.3 Efficient Computation

We also analyze ways of accelerating the computation required for solving the symmetric game. An equivalent criterion is given by

Lemma 2.4. *If $d(\cdot, \cdot)$ is squared error, $\mathcal{L}(\cdot, \cdot)$ is differentiable, f is sub-differentiable, and* **A(4-5)** *hold, then*

$$\sum_{(x_i, y_i) \in \mathcal{D}} \mathcal{L}(f(x_i), y_i) + \frac{\lambda}{\bar{N}_i} \left[\bar{N}_i f(x_i) - \sum_{x_t \in \mathcal{B}(x_i)} \frac{\hat{g}_{x_t}(x_i)}{|\mathcal{B}(x_t)|} \right]^2, \tag{2.6}$$

where $\bar{N}_i := \sum_{x_t \in \mathcal{B}(x_i)} \frac{1}{|\mathcal{B}(x_t)|}$, induces the same equilibrium as the symmetric game.

The result is useful when training f on GPU and solving $\hat{g}_{x_i}$ on CPU. Compared to handling different neighborhood sizes for Eq. (2.3) on the GPU, computing a summarized feedback on CPU as in Eq. (2.6) is more efficient (and easier to implement).

2.4.4 Discussion

We investigated here discrete neighborhoods and they are suitable also for structured data as in the experiments. The method itself can be generalized to continuous neighborhoods with an additional difficulty: the exact computation and minimization of functional deviation between the predictor and the witness in such neighborhoods is in general intractable. We may apply results from learning theory (e.g., [128]) to bound the (generalization) gap between the deviation computed by a finite number of samples from the continuous neighborhood and the actual deviation under a uniform probability measure.

2.5 Examples

2.5.1 Conditional Sequence Generation

The basic idea of co-operative modeling extends naturally to conditional sequence generation over longer periods. Broadly, the mechanism allows us to inspect the temporal progression of sequences on a longer term basis.

Given an observation sequence $x_1, \ldots, x_t \in \mathbb{R}^c$, the goal is to estimate probability $p(x_{t+1:T}|x_{1:t})$ over future events $x_{t+1}, \ldots, x_T \in \mathbb{R}^c$, typically done via maximum likelihood. For brevity, we use $x_{1:i}$ to denote $x_1, \ldots, x_i$. We autoregressively model the conditional distribution of x_{i+1} given $x_{1:i}$ as a multivariate Gaussian distribution with mean $\mu(x_{1:i})$ and covariance $\Sigma(x_{1:i})$, both parametrized via recurrent neural networks. Each local witness model $g_{x_{1:i}}(\cdot)$ is estimated based on the neighborhood $\mathcal{B}(x_{1:i}) \triangleq \{x_{1:i-\epsilon}, \ldots, x_{1:i+\epsilon}\}$ with respect to the mean function $\mu(\cdot)$. A natural choice would be a K^{th}-order Markov autoregressive (AR) model with an ℓ_2 deviation loss as:

$$\min_{\theta} \sum_{x_{1:t} \in \mathcal{B}(x_{1:i})} \| \sum_{k=0}^{K-1} \theta_{k+1} \cdot x_{t-k} + \theta_0 - \mu(x_{1:t}) \|_2^2,$$

where $\theta_k \in \mathbb{R}^{c \times c}, \forall k > 0$ and $\theta_0 \in \mathbb{R}^c$. The AR model admits an analytical solution similar to linear regression.

2.5.2 Chemical Property Prediction

The models discussed in §2.3 can be instantiated on highly-structured data, such as molecules, too. A molecule can be represented as a graph $\mathcal{M} = (\mathcal{V}, \mathcal{E})$ whose nodes encode the atom types and edges encode the chemical bonds. Such representation enables the usage of recent graph convolutional networks (GCNs) [32, 89] as the predictor f. As it is hard to realize a simple witness on the raw graph representation, we exploit an alternative data representation for the witness model; we leverage depth-bounded decision trees that take as input Morgan fingerprints [122] $x(\mathcal{M})$, which are binary feature vectors indicating the presence of various chemical substructures (e.g., the nodes in Figure 2.1.1).

Through automatic matching molecular pair analysis [53], we construct the neighborhood $\mathcal{B}(\mathcal{M})$ with molecules $\{\mathcal{M}'\}$ whose Tanimoto similarity with M is greater than 0.6. Here we use a multi-label binary classification task as an example, and adopt a cross-entropy loss for each label axis for simplicity. At each neighborhood $\mathcal{B}(\mathcal{M})$, we construct a witness decision tree g that minimizes the total variation (TV) from the predictor as

$$\min_{g \in \mathcal{G}_{\text{tree}}} \frac{1}{|\mathcal{B}(\mathcal{M})|} \sum_{\mathcal{M}' \in \mathcal{B}(\mathcal{M})} \sum_{i=1}^{\dim(\mathcal{Y})} |f(\mathcal{M}')_i - g(x(\mathcal{M}'))_i|. \tag{2.7}$$

Note that Eq. (2.7) is an upper bound and an efficient alternative to fitting a tree for each label axis independently.

2.5.3 Molecule Representation Learning

Our approach can be further applied to learn transparent latent graph representations in variational autoencoders (VAEs) [73, 69]. Concretely, given a molecular graph $\mathcal{M} = (\mathcal{V}, \mathcal{E})$, the VAE encoder q outputs the approximated posterior $z_{\mathcal{M}} \sim \mathcal{N}(\mu_{\mathcal{M}}, \Sigma_{\mathcal{M}})$ over the latent space, where $z_{\mathcal{M}}$ is a stochastic, continuous representation of the molecule $\mathcal{M}$. Following common practice, $\Sigma_{\mathcal{M}}$ is restricted to be diagonal. The VAE decoder then reconstructs the molecule $\mathcal{M}$ from its probabilistic encoding $z_{\mathcal{M}}$. Our goal here is to guide the behavior of the neural encoder q such that the derivation of (probabilistic) $z_{\mathcal{M}}$ can be locally explained and/or replaced by a decision tree.

We adopt the same setting for the witness function and the neighborhoods as in §2.5.2, except that the local decision tree g now outputs a joint normal distribution with parameters $[\widehat{\mu}_{\mathcal{M}}, \widehat{\Sigma}_{\mathcal{M}}]$. To train the encoder, we extend the original VAE objective $\mathcal{L}^{\text{VAE}}$ with a local deviation loss $\mathcal{L}^{\mathcal{G}_{\text{tree}}}$ defined on the KL divergence between the VAE posterior $q(\mathcal{M}) = \mathcal{N}(\mu_{\mathcal{M}}, \Sigma_{\mathcal{M}})$ and the witness posterior $g(x(\mathcal{M})) = \mathcal{N}(\widehat{\mu}_{\mathcal{M}}, \widehat{\Sigma}_{\mathcal{M}})$, aggregated across neighborhoods, as

$$\mathcal{L}^{\mathcal{G}_{\text{tree}}} := \frac{1}{|\mathcal{D}|} \sum_{\mathcal{M} \in \mathcal{D}} \frac{1}{|\mathcal{B}(\mathcal{M})|} \min_{g \in \mathcal{G}_{\text{tree}}} \sum_{\mathcal{M}' \in \mathcal{B}(\mathcal{M})} \text{KL}(g(x(\mathcal{M}'))||q(\mathcal{M}'))$$

The VAE is trained to minimize $\mathcal{L}^{\text{VAE}} + \lambda \cdot \mathcal{L}^{\mathcal{G}_{\text{tree}}}$. For ease of optimization, we asymmetrically estimate each decision tree g with mean squared error between the vectors $[\mu_{\mathcal{M}}, \Sigma_{\mathcal{M}}]$ and $[\widehat{\mu}_{\mathcal{M}}, \widehat{\Sigma}_{\mathcal{M}}]$.

2.6 Experiments

We conduct experiments on chemical and time-series datasets. Due to the lack of existing works for explaining structured data, we adopt an ablation setting—comparing our approach (Game) versus an unregularized model (Deep)—and focus on measuring the resulting explanation (i.e., the local witnesses). We use subscripts to denote specific versions of the Game models. Note that we only fit the local witnesses to the Deep model during testing time for evaluation.

2.6.1 Chemical Property Prediction

We conduct experiments on molecular toxicity prediction on the Tox21 dataset from MoleculeNet [158], which contains 12 binary labels and $7,831$ molecules. The labels are very unbalanced; the fraction of the positive label is between 16.15% and 3.51% among the 12 labels. We use GCN as the predictor and decision trees as the local witnesses as described in §2.5.2. The neighborhood sizes m of about 60% of the molecules are larger than 2, whose median and maximum are 59 and 300, respectively. Since each neighborhood has a different size m, we set the maximum tree depth as $\max\{\lceil \log_2(m) \rceil - 1, 1\}$ for each neighborhood to prevent the corresponding size m from being not effective for $m > 2$ (see Definition 2.1). More details are in Appendix 2.B.

Table 2.6.1: Performance on the Tox-21 dataset. $\mathrm{AUC}_{\mathcal{D}}(\hat{g}_{\mathcal{M}}, f)$ and $\mathrm{AUC}_{\mathcal{B}}(\hat{g}_{\mathcal{M}}, f)$ generalize the AUC score to use f values as labels, computed on the testing data and their neighborhoods, respectively.

Aspect	Measure	GAME$_{\text{unif}}$	GAME$_{\text{sym}}$	DEEP
Performance	$\mathrm{AUC}(\hat{g}_{\mathcal{M}}, y)$	0.742	**0.824**	0.818
(the higher the better)	$\mathrm{AUC}(f, y)$	0.744	**0.826**	0.815
Consistency	$\mathrm{AUC}_{\mathcal{B}}(\hat{g}_{\mathcal{M}}, f)$	**0.764**	0.759	0.735
(the higher the better)	$\mathrm{AUC}_{\mathcal{D}}(\hat{g}_{\mathcal{M}}, f)$	0.959	**0.967**	0.922

Evaluation Measures: A more detailed version of how these measures are computed can be found in Appendix 2.B.

(1) Performance: We compare the predictions of the witness with the labels in AUC, denoted as $\mathrm{AUC}(\hat{g}_{\mathcal{M}}, y)$. We also evaluate the predictor similarly (denoted as $\mathrm{AUC}(f, y)$) so as to analyze the regularization effect of the games.

(2) Consistency: As labels are unavailable for testing data in practice, we may instead measure the similarity between the predictor and the local witnesses to understand the consistency of the distillation. To this end[1], we generalize the AUC criterion for continuous labels for N references y and predictions y' as

$$\frac{\sum_{i=1}^{N} \sum_{j=1}^{N} \mathbb{I}(y_i > y_j)\mathbb{I}(y'_i > y'_j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \mathbb{I}(y_i > y_j)}.$$

The proposed score has the same pairwise interpretation as AUC, recovers AUC when y is binary, and is normalized to $[0, 1]$. Locally, we measure the criterion for the local witnesses with respect to the predictor in each testing neighborhood as the local consistency, where the average result is denoted as $\mathrm{AUC}_{\mathcal{B}}(\hat{g}_{\mathcal{M}}, f)$. Globally, the criterion is also validated among the testing data, denoted as $\mathrm{AUC}_{\mathcal{D}}(\hat{g}_{\mathcal{M}}, f)$.

The results with the uniform criterion and the symmetric game are shown in Table 2.6.1. A baseline vanilla decision tree, with depth tuned between 2 and 30, yields 0.617 in $\mathrm{AUC}(f, y)$. Compared to GAME$_{\text{sym}}$, the local consistency in GAME$_{\text{unif}}$ is marginally improved due to the strict constraint at the cost of severe performance

[1]Since the predictor probability can be scaled arbitrarily to minimize the TV from decision trees without affecting performance, using TV to measure consistency as used in training is not ideal.

Table 2.6.2: $\mathrm{AUC}_{\mathcal{D}}(\hat{g}_{\mathcal{M}}, f)$ score on different Δ in the Tox-21 dataset (lower Δ implies shallower trees).

Model	$\Delta = 0$	$\Delta = -1$	$\Delta = -2$	$\Delta = -3$
GAME	0.967	0.967	0.964	0.958
DEEP	0.922	0.916	0.915	0.914

loss. We investigate the behaviors in training neighborhoods and find that GAME$_{\mathrm{sym}}$ exhibits a tiny fraction of high deviation losses, allowing the model to behave more flexibly than the strictly constrained GAME$_{\mathrm{unif}}$ (see Figure 2.A.1 in Appendix 2.B). In terms of performance, our GAME$_{\mathrm{sym}}$ model is superior to the DEEP model in terms of both the local witnesses and the predictor. When comparing the witness to the predictor, locally and globally, the GAME models significantly improve the consistency from the DEEP model. The local consistency should be interpreted relatively since the tree depth inherently prevents local overfitting.

We visualize the resulting witness trees in Figure 2.1.1 under the same consistency constraint: for a local neighborhood, we grow the witness tree for the DEEP model until the local consistency in $\mathrm{AUC}_{\mathcal{B}}$ is comparable to the GAME$_{\mathrm{sym}}$ model. For explaining the same molecule, the tree for the DEEP model is deeper and extremely unbalanced. Since a Morgan fingerprint encodes the existence of a substructure of molecule graphs, an unbalanced tree focusing on the left branch (non-existence of a substructure) does not capture much generality. Hence, the explanation of the DEEP model does not provide as much insight as our GAME$_{\mathrm{sym}}$ model.

Here we do an analysis on the tree depth constraint for the witness model, as a shallower tree is easier to interpret, but more challenging to establish consistency due to the restricted complexity. To this end, we revise the depth constraint to $\max\{\lceil \log_2(m) \rceil - 1 + \Delta, 1\}$ during training and testing, and vary $\Delta \in \{-3, \dots, 0\}$. All the resulting GAME models outperform the DEEP models in $\mathrm{AUC}(f, y)$, and we report the consistency score in terms of $\mathrm{AUC}_{\mathcal{D}}(\hat{g}_{\mathcal{M}}, f)$ in Table 2.6.2. Even when $\Delta = -3$, the witness trees in our GAME model still represent the predictor more faithfully than those in the DEEP model with $\Delta = 0$, demonstrating the significance of our game-theoretic approach which leads to such coherency.

Table 2.6.3: Performance of the symmetric and asymmetric setting of the GAME model with $\epsilon = 9$.

$(\times 10^{-2})$	λ	0	0.1	1	10	100	AR
GAME$_{\text{asym}}$	Error	8.136	8.057	8.309	9.284	9.794	9.832
	Deviation	4.197	4.178	3.431	1.127	0.186	0.000
	TV	7.341	7.197	5.706	1.177	0.144	0.000
GAME$_{\text{sym}}$	Error	8.136	8.089	8.315	9.314	9.807	9.832
	Deviation	4.197	4.169	3.426	1.116	0.182	0.000
	TV	7.341	7.292	5.621	1.068	0.132	0.000

2.6.2 Physical Component Modeling

We next validate our approach on a physical component modeling task with the bearing dataset from NASA [87], which records 4-channel acceleration data on 4 co-located bearings. We divide each sequence into disjoint subsequences, resulting in $200,736$ subsequences in total. Since the dataset exhibits high frequency periods of 5 points and low frequency periods of 20 points, we use the first 80 points in an sequence to forecast the next 20. We parametrize $\mu(\cdot)$ and $\Lambda(\cdot)$ jointly by stacking 1 layer of CNN, LSTM, and 2 fully connected layers. We set the neighborhood radius ϵ to 9 such that the local witnesses are fit with completely different data for the beginning and the end of the sequence. The Markov order K is set to 2 to ensure the effectiveness of the neighborhood sizes. More details are in Appendix 2.C.

In testing time, the local witnesses are estimated based on the autoregressive generative trajectories, so it is natural to treat each local witness as purely an explanation rather than a predictive function, as in the typical post-hoc explanation setting.

Evaluation involves three different types of errors: 1) 'error' is the root mean squared error (RMSE) between greedy autoregressive generation and the ground truth, 2) 'deviation' is RMSE between the predictor $\mu(x_{1:i})$ and the witness $\hat{g}_{x_{1:i}}(x_{1:i})$, and 3) 'TV' is the average total variation of witness $\hat{g}_{x_{1:i}}$ parameters $[\theta, \theta_0]$ between every two consecutive time points. Since the deviation and error are both computed on the same space in RMSE, the two measures are readily comparable.

We present the results in Table 2.6.3 to study the impact of the game coefficient

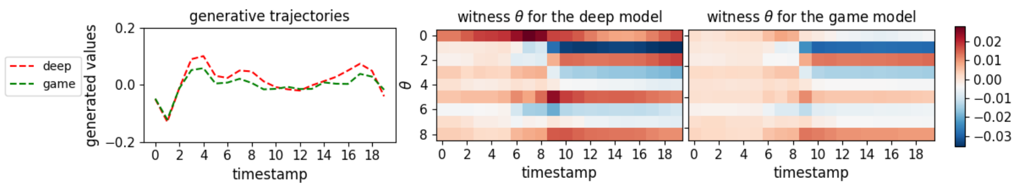


Figure 2.6.1: Visualization of the local linear witnesses (middle and right plots) on the first channel (left plot) along the autoregressive generative trajectory (x-axis) on the bearing dataset. The y-axis of the parameters from 0 to 8 denotes the bias $(\theta_0)_1$ and weights $(\theta_1)_{1,1:4}$, $(\theta_2)_{1,1:4}$.

λ and the symmetry of the games. The trends in the measures are quite monotonic on λ: with an increasing λ, the predictor gradually operates toward the AR family with lower deviation and TV but higher error. When $\lambda = 0.1$, the GAME models are more accurate than the DEEP model ($\lambda = 0$) due to the regularization effect. Given the same hyper-parameters, marginally lower deviation in the symmetric game than in the asymmetric game confirms our analysis about the similarity between the two. In practice, the asymmetric game is more efficient and substantially easier to implement than the symmetric game. Indeed, the training time is 20.6 sequences/second for the asymmetric game, and 14.6 sequences/second for the symmetric game. If we use the formula in Lemma 2.4, the symmetric game can be accelerated to 20.4 sequences/second, but the formula does not generalize to other deviation losses.

We visualize the local witnesses with their parameters $[\theta_0, \theta]$ along the autoregressive generative trajectories in Figure 2.6.1. The stable funcitonal patterns of the GAME model as reflected by θ, before and after the 9th point, highlight not only close local alignments between the predictor and the AR family (being constant vectors across columns) but also flexible variation of functional properties on the predictor across regions. In contrast, the DEEP model yields unstable linear coefficients, and relies more on biases θ_0 than the GAME model, although the linear weights are more useful for grounding the coordinate relevance for interpretability. Finally, we remark that despite the uninterpretable nature of temporal signals, the functional pattern reflected by the linear weights as shown here yields a simple medium to understand its behavior. Some additional analyses and visualization are included in Appendix 2.C.

Table 2.6.4: The performance in ELBO for the raw neural encoders and locally adapted decision trees. The deviation is defined in §2.5.3.

Model	$\text{ELBO}_{\text{neural encoder}}$	$\text{ELBO}_{\text{decision tree}}$	deviation ($\mathcal{L}^{\mathcal{G}_{\text{tree}}}$)
Deep	-21.6	-25.4	4.64
Game	**-21.5**	**-25.1**	**3.98**

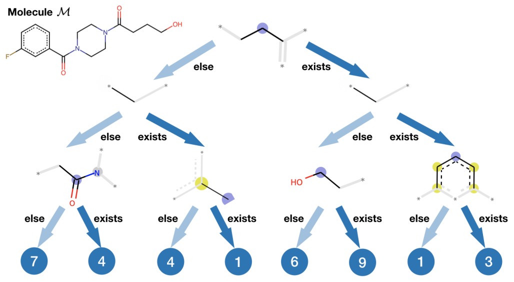


Figure 2.6.2: The local decision tree explains the latent representation for a molecule (upper left) by identifying locally discriminative chemical substructures. The leaf nodes are annotated with their sizes (number of molecules belonging to that cluster).

2.6.3 Molecule Representation Learning

Finally, we validate our approach on learning representations for molecules with VAEs, where we use the junction tree VAE [69] as an example. Here the encoders of VAEs, with and without the guidance of local decision trees (see §2.5.3), are again denoted as Deep and Game, respectively. The models are trained on the ZINC dataset [137] containing 1.5M molecules, and evaluated on a test set with 20K molecules. We measure the performance in terms of the evidence lower bound (ELBO) over the test set. Here we consider two scenarios: the ELBO using the raw latent representations from the original neural encoder, and using the transparent latent representations generated by the local decision tree witnesses. The average deviation loss in KL divergence $\mathcal{L}^{\mathcal{G}_{\text{tree}}}$, defined in §2.5.3, over the testing neighborhoods is also evaluated.

The results are shown in Table 2.6.4. Our Game model performs consistently better than the baseline Deep model under all the metrics. Figure 2.6.2 shows an example of how our decision tree witness embeds the local neighborhood of a molecule. Most of the substructures selected by the decision tree occur in the side chains outside of Bemis-Murcko scaffold [12]. This shows the variation in the latent representation mostly reflects the local changes in the molecules, which is expected since changes in the scaffold typically lead to global changes such as chemical property changes.

2.7 Conclusion

We propose a novel game-theoretic approach to learning transparent models on structured data. The game articulates how locally transparent witnesses can be guided towards a flexible predictor with consistency. This work opens up many avenues for future work, from theoretical analysis of the games to a multi-player setting.

2.A Proofs

2.A.1 Proof and Discussion of Theorem 2.2

Proof. We first re-write the symmetric criterion explicitly as a game:

$$\min_f \sum_i (f(x_i) - y_i)^2 + \frac{\lambda}{m} \sum_{x_j \in \mathcal{B}(x_i)} (f(x_j) - \hat{g}_{x_i}(x_j))^2,$$

where $\hat{g}_{x_i}$ is the best response strategy from the local witness.

Since f is unconstrained and the objective in convex in it, we can treat each $f(x_i)$ as a distinct variable, and use the derivative to find its optimum:

$$f_S^*(x_i) = \frac{1}{1+\lambda}\Bigg[y_i + \frac{\lambda}{m} \sum_{x_j \in \mathcal{B}^{-1}(x_i)} \hat{g}_{x_j}(x_i)\Bigg] = \frac{1}{1+\lambda}\Bigg[y_i + \frac{\lambda}{m} \sum_{x_j \in \mathcal{B}(x_i)} \hat{g}_{x_j}(x_i)\Bigg], \quad (2.8)$$

where $\mathcal{B}^{-1}(x_i) = \{x_j \in \mathcal{D}_x : x_i \in \mathcal{B}(x_j)\}$. Note that we only have to collect witnesses $\hat{g}_{x_j}$ that are relevant to $f(x_i)$ for the first equality, and the second equality is due to **(A4)**. On the other hand, the objective for f in the asymmetric game is:

$$\min_f \sum_i (f(x_i) - y_i)^2 + \lambda (f(x_i) - \hat{g}_{x_i}(x_i))^2.$$

The corresponding optimum is:

$$f_A^*(x_i) = \frac{1}{1+\lambda}\bigg[y_i + \lambda \hat{g}_{x_i}(x_i)\bigg]. \tag{2.9}$$

For both games, the objective for g_{x_i} can be described as:

$$\min_{g_{x_i}} \frac{\lambda}{m} \sum_{x_j \in \mathcal{B}(x_i)} (f(x_j) - g_{x_i}(x_j))^2 = \min_{\theta_i} \frac{\lambda}{m} \|f(X_i) - X_i\theta_i\|_2^2, \tag{2.10}$$

Then Eq. (2.11) is an optimal witness $g_{x_i}^*$ at x_i.

$$g_{x_i}^*(x_j) = \theta_i^\top x_j = (X_i^\dagger f(X_i))^\top x_j, \forall x_j \in \mathcal{X}, \tag{2.11}$$

and we note that every optimal witness $g_{x_i}^*$ has the same values on $\mathcal{B}(x_i)$

Since the optimal $g_{x_i}^*$ is functionally dependent to f. we put Eq. (2.11) back to Eq. (2.8) to obtain the optimal condition for f_S^* (at equilibrium) as

$$f_S^*(x_i) = \frac{1}{1+\lambda}\bigg[y_i + \frac{\lambda}{m}(\sum_{x_j \in \mathcal{B}(x_i)} X_j^\dagger f_S^*(X_j))^\top x_i\bigg].$$

Again, putting Eq. (2.11) back to Eq. (2.9), we obtain the optimal condition for f_A^* at equilibrium as

$$f_A^*(x_i) = \frac{1}{1+\lambda}\bigg[y_i + \lambda(X_i^\dagger f_A^*(X_i))^\top x_i\bigg].$$

$\square$

Note that the equilibrium for the linear class is not unique when the solution of Eq. (2.10) is not unique: there may be infinitely many optimal solution to the witness in a neighborhood due to degeneracy. In this case, Theorem 2.2 adopts the minimum norm solution as used in the pseudo-inverse in Eq. (2.11). In this case, one may use Ridge regression instead to establish a strongly convex objective for the witness to ensure a unique solution.

2.A.2 Proof of Theorem 2.3

Proof. The objective for the uniform criterion is:

$$\min_f \sum_{i=1}^{N} (f(x_i) - y_i)^2, s.t. \ \min_{g \in \mathcal{G}} \frac{1}{m} \sum_{x_j \in \mathcal{B}(x_i)} (f(x_j) - g(x_j))^2 \leq \delta, \forall x_i \in \mathcal{D}_x.$$

Our strategy is to temporarily treat each g as a fixed function, and then replace it with its best response strategy.

Since f is unconstrained, we can treat each $f(x_i)$ as a distinct variable for optimization. For each $f(x_i)$, we first filter its relevant criteria:

$$\begin{aligned} &\min_{f(x_i)} (f(x_i) - y_i)^2 \\ &s.t. \ (f(x_i) - g_{x_j}(x_i))^2 \leq \delta m - \sum_{x_k \in \mathcal{B}(x_j) \setminus \{x_i\}} (f(x_k) - g_{x_j}(x_k))^2, \forall x_j \in \mathcal{B}(x_i). \end{aligned}$$

For any feasible f, we can rewrite the constraint of $f(x_i)$ with respect to each x_j as:

$$\begin{aligned} &g_{x_j}(x_i) - \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j) \setminus \{x_i\}} (f(x_k) - g_{x_j}(x_k))^2} \\ &\leq f(x_i) \leq g_{x_j}(x_i) + \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j) \setminus \{x_i\}} (f(x_k) - g_{x_j}(x_k))^2}. \end{aligned}$$

Collectively, we can fold all the upper bounds of $f(x_i)$ as

$$f(x_i) \leq \min_{x_j \in \mathcal{B}(x_i)} \left[g_{x_j}(x_i) + \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j) \setminus \{x_i\}} (f(x_k) - g_{x_j}(x_k))^2} \right].$$

All the lower bounds can be folded similarly.

Since the objective for $f(x_i)$ is simply a squared error with an interval constraint, evidently if y_i satisfies the lower bounds and upper bounds, then $f_U^*(x_i) = y_i$. If

$$y_i > \min_{x_j \in \mathcal{B}(x_i)} \left[g_{x_j}(x_i) + \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j) \setminus \{x_i\}} (f(x_k) - g_{x_j}(x_k))^2} \right],$$

then we have

$$f^*_U(x_i) = \min_{x_j \in \mathcal{B}(x_i)} \left[g_{x_j}(x_i) + \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j)\setminus\{x_i\}} (f(x_k) - g_{x_j}(x_k))^2} \right].$$

For the remaining case, we have

$$f^*_U(x_i) = \max_{x_j \in \mathcal{B}(x_i)} \left[g_{x_j}(x_i) - \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j)\setminus\{x_i\}} (f(x_k) - g_{x_j}(x_k))^2} \right].$$

For each g_{x_i} is in the linear class, Eq. (2.12) is an optimal solution.

$$g^*_{x_j}(x_i) = (X_j^\dagger f(X_j))^\top x_i, \forall x_i \in \mathcal{X}, \tag{2.12}$$

and we note that every optimal witness $g^*_{x_j}$ has the same values on $\mathcal{B}(x_j)$.

Since the optimal $g^*_{x_i}$ is functionally dependent to f, to obtain the optimal f^*_U, we combine our previous result with $g^*_{x_i}$ such that the optimality conditions for f and g_{x_i} are both satisfied. Finally, we have

$$f^*_U(x_i) = \begin{cases} \alpha(x_i, f^*_U), & \text{if } \alpha(x_i, f^*_U) > y_i, \\ \beta(x_i, f^*_U), & \text{if } \beta(x_i, f^*_U) < y_i, \\ y_i, & \text{otherwise,} \end{cases}$$

for $x_i \in \mathcal{D}_x$, where

$$\alpha(x_i, f^*_U) = \max_{x_j \in \mathcal{B}(x_i)} \left[(X_j^\dagger f^*_U(X_j))^\top x_i - \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j)\setminus\{x_i\}} (f^*_U(x_k) - (X_j^\dagger f^*_U(X_j))^\top x_k)^2} \right];$$

$$\beta(x_i, f^*_U) = \min_{x_j \in \mathcal{B}(x_i)} \left[(X_j^\dagger f^*_U(X_j))^\top x_i + \sqrt{\delta m - \sum_{x_k \in \mathcal{B}(x_j)\setminus\{x_i\}} (f^*_U(x_k) - (X_j^\dagger f^*_U(X_j))^\top x_k)^2} \right].$$

□

2.A.3 Proof of Lemma 2.4

Proof. Since the criteria for the witness g_{x_i} are the same in the symmetric game and the proposed asymmetric criterion here, we only have to check for the optimality condition for the predictor f. Here we use $\nabla_\theta f(x)$ to denote the subgradient of f at x with respect to the underlying parameter θ, the optimality condition for Eq. (2.6) is

$$\begin{aligned}\mathbf{0} &\in \sum_{(x_i,y_i)\in\mathcal{D}} \left[\frac{\partial}{\partial f(x_i)}\mathcal{L}(f(x_i),y_i) + 2\lambda(\sum_{x_t\in\mathcal{B}(x_i)} \frac{f(x_i)}{|\mathcal{B}(x_t)|} - \sum_{x_t\in\mathcal{B}(x_i)} \frac{\hat{g}_{x_t}(x_i)}{|\mathcal{B}(x_t)|})\right]\nabla_\theta f(x_i)\\ &= \sum_{(x_i,y_i)\in\mathcal{D}} \left[\frac{\partial}{\partial f(x_i)}\mathcal{L}(f(x_i),y_i)\nabla_\theta f(x_i) + \sum_{x_t\in\mathcal{B}(x_i)} \frac{2\lambda}{|\mathcal{B}(x_t)|}(f(x_i)-\hat{g}_{x_t}(x_i))\nabla_\theta f(x_i)\right]\end{aligned}$$

For the symmetric game, the optimality condition is

$$\mathbf{0} \in \sum_{(x_i,y_i)\in\mathcal{D}} \left[\frac{\partial}{\partial f(x_i)}\mathcal{L}(f(x_i),y_i)\nabla_\theta f(x_i) + \sum_{x_t\in\mathcal{B}(x_i)} \frac{2\lambda}{|\mathcal{B}(x_i)|}(f(x_t)-\hat{g}_{x_i}(x_t))\nabla_\theta f(x_t)\right]$$

It is evident that the two conditions coincide if Eq. (2.13) is equal to Eq. (2.14).

$$\begin{aligned}&\sum_{(x_i,y_i)\in\mathcal{D}}\sum_{x_t\in\mathcal{B}(x_i)} \frac{1}{|\mathcal{B}(x_i)|}(f(x_t)-\hat{g}_{x_i}(x_t))\nabla_\theta f(x_t) && (2.13)\\ &= \sum_{x_t\in\cup_{x_i\in\mathcal{D}_x}\mathcal{B}(x_i)}\sum_{x_i\in\mathcal{B}^{-1}(x_t)} \frac{1}{|\mathcal{B}(x_i)|}(f(x_t)-\hat{g}_{x_i}(x_t))\nabla_\theta f(x_t)\\ &= \sum_{x_t\in\mathcal{D}_x}\sum_{x_i\in\mathcal{B}(x_t)} \frac{1}{|\mathcal{B}(x_i)|}(f(x_t)-\hat{g}_{x_i}(x_t))\nabla_\theta f(x_t)\\ &= \sum_{(x_i,y_i)\in\mathcal{D}}\sum_{x_t\in\mathcal{B}(x_i)} \frac{1}{|\mathcal{B}(x_t)|}(f(x_i)-\hat{g}_{x_t}(x_i))\nabla_\theta f(x_i), && (2.14)\end{aligned}$$

where the first equality is simply re-ordering of the two summations, and the second equality is due to $x_t \in \mathcal{B}(x_i) \iff x_i \in \mathcal{B}(x_t)$ and $\cup_{x_i\in\mathcal{D}_x}\mathcal{B}(x_i) = \mathcal{D}_x$. □

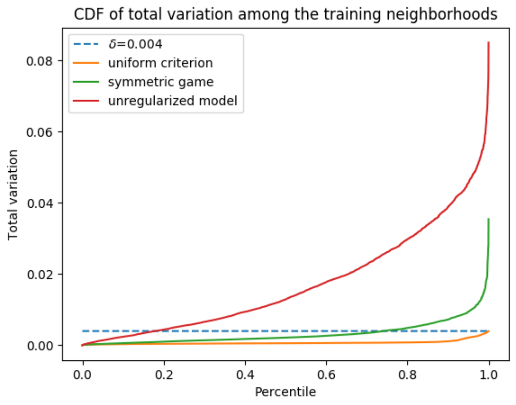


Figure 2.A.1: The cumulative distribution function of the total variation loss between the predictor f and the local witness g in each training neighborhood.

2.B Supplementary Materials for Molecule Property Prediction

Implementation. To conduct training, we use GCNs as the predictor with 6 layers of graph convolution with 1800 hidden dimension. We use a 80%/10%/10% split for training / validation / testing.

Evaluation Measures. We use the `roc_auc_score` in `scikit-learn` [114] to compute the AUC score. Note that for each criterion, we evaluate the model with respect to each label, and then report the average score across the 12 labels. Here N denotes the number of testing data.

- AUC(f, y): we compare $f(\mathcal{M}_i)$ with the labels y_i among the testing data $\{(\mathcal{M}_i, y_i)\}_{i=1}^N$ in AUC.
- AUC$(\hat{g}_{\mathcal{M}}, y)$: we compare $\hat{g}_{\mathcal{M}_i}(x(\mathcal{M}_i))$ with the labels y_i among the testing data $\{(\mathcal{M}_i, y_i)\}_{i=1}^N$ in AUC.
- AUC$_{\mathcal{B}}(\hat{g}_{\mathcal{M}}, f)$: for each testing data $(\mathcal{M}, y)$, we evaluate the following score among the neighborhood $\mathcal{B}(\mathcal{M}) = \{\mathcal{M}_1, \ldots, \mathcal{M}_{N_{\mathcal{M}}}\}$, where $N_{\mathcal{M}} := |\mathcal{B}(\mathcal{M})|$,

around $\mathcal{M}$:

$$\frac{\sum_{i=1}^{N_\mathcal{M}}\sum_{j=1}^{N_\mathcal{M}}\mathbb{I}(f(\mathcal{M}_i) > f(\mathcal{M}_j))\mathbb{I}(\hat{g}_\mathcal{M}(\mathcal{M}_i) > \hat{g}_\mathcal{M}(\mathcal{M}_j))}{\sum_{i=1}^{N_\mathcal{M}}\sum_{j=1}^{N_\mathcal{M}}\mathbb{I}(f(\mathcal{M}_i) > f(\mathcal{M}_j))}.$$

The average score across all the testing neighborhood is then reported.

- $\text{AUC}_\mathcal{D}(\hat{g}_\mathcal{M}, f)$: we evaluate the following score among the testing data $\{(\mathcal{M}_i, y_i)\}_{i=1}^N$:

$$\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbb{I}(f(\mathcal{M}_i) > f(\mathcal{M}_j))\mathbb{I}(\hat{g}_{\mathcal{M}_i}(\mathcal{M}_i) > \hat{g}_{\mathcal{M}_j}(\mathcal{M}_j))}{\sum_{i=1}^{N}\sum_{j=1}^{N}\mathbb{I}(f(\mathcal{M}_i) > f(\mathcal{M}_j))}.$$

Visualization. To investigate the behavior of the models, we plot their total variation loss from the local witness among the training neighborhoods in Figure 2.A.1. The uniform criterion imposes a strict functional constraint, while the symmetric game allows a more flexible model, exhibiting a tiny fraction of high deviation among the training neighborhoods.

2.C Supplementary Materials for Physical Component Modeling

Implementation. We randomly sample 85%, 5%, and 10% of the data for training, validation, and testing. We set the learning rate as 10^{-5} with the Adam optimizer [72]. The batch size is set to 128. All the hidden dimensions are set to 128. We use the `MultivariateNormalTriL` function in Tensorflow [1] to parametrize the multivariate Gaussian distribution. Specifically, we let the network output a $N + \frac{(N+1)(N)}{2}$ dimensional vector. The first N dimensions are treated as the mean. The second part is transformed to a lower triangular matrix, where the diagonal is further processed with a softplus nonlinearity. Such representation satisfies the Cholesky decomposition for covariance matrix.

For fitting the linear witness, we use Ridge regression in `scikit-learn` [114] with

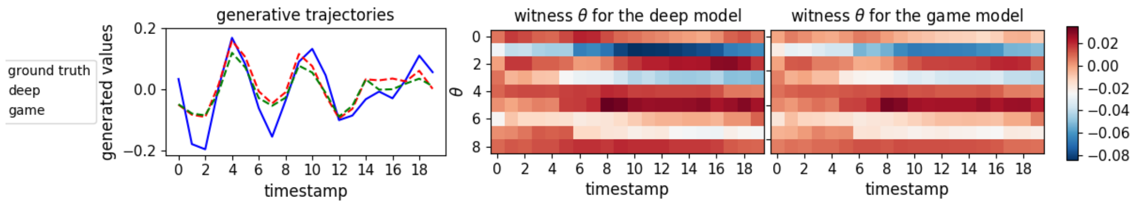


Figure 2.C.1: Visualization of the local linear witnesses (middle and right plots) on the first channel (left plot) along the *teacher-forced* trajectory (x-axis) on the bearing dataset. The y-axis of the parameters from 0 to 8 denotes the bias $(\theta_0)_1$ and weights $(\theta_1)_{1,1:4}, (\theta_2)_{1,1:4}$.

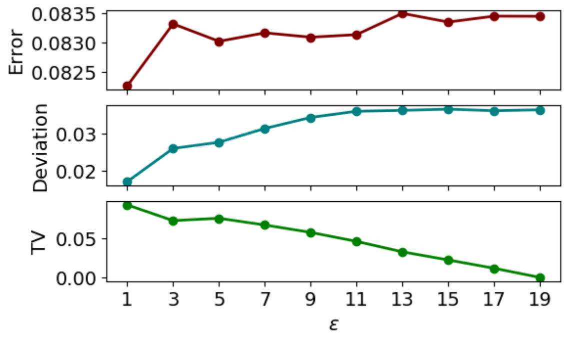


Figure 2.C.2: Parameter analysis of ϵ on the Game model with $\lambda = 1$.

the default hyperparameter. The usage of Ridge regression instead of vanilla linear regression is justified by our analysis of the equilibrium for linear witnesses.

Visualization. The visualization for the teacher-forced generative trajectory is in Figure 2.C.1.

Neighborhood size analysis. Here we investigate the effect of neighborhood radius ϵ. The results are shown in Figure 2.C.2. The impact of the neighborhood size is quite monotonic to deviation and TV, but in a reverse way. As ϵ increases, the weight of the witness on fitting the current point x_i among the neighborhood $\mathcal{B}(x_i)$ decreases, so the deviation of the witness $\hat{g}_{x_i}(x_i)$ from $f(x_i)$ increases. In contrast, as more points are overlapped between the neighborhoods of consecutive points, the resulting witnesses are more similar and thus yield smaller TV. In terms of prediction error, as the neighborhood radius ϵ determines the region to impose coherency, a larger region leads to greater restriction on the predictive model. All the arguments are well supported by the empirical results. We suggest users to trade off faithfulness (deviation) and smooth transition of functional properties (TV) based on the application at hand. Note that smooth transition of functional properties does not imply smoothness of f.

Finally, we remark that our sample complexity analysis for the linear class suggests that the neighborhood size is guaranteed to be effective for $2\epsilon + 1 > d = 2c + 1 = 9$. However, since the result is an sufficient condition, the regularization may still happens for $\epsilon < 5$ if the neighborhood matrix $X_i = [x_{i-\epsilon}, \ldots, x_{i+\epsilon}]^\top$ is not full rank.

Chapter 3

Towards Robust, Locally Linear Models

3.1 Introduction

As briefly discussed in §1.2.2, despite the flexibility of the game-theoretic approach proposed in Chapter 2, this approach is not very efficient when it comes to linear witnesses in high dimensions. Indeed, the neighborhood size should be at least $D+1$ in D dimensions in order to be effective, and thus, in general, each local estimation will be $\Omega(D^3)$ in time. In this chapter, we tackle the same problem as building models that are locally transparent, but focus on the case of locally linear models.

Here we make the observation that piecewise linear mappings are naturally locally linear. In particular, the gradient evaluation with respect to the input coordinates directly yields the linear model corresponding to the linear region where the input resides. Such mappings are quite prevalent. Every feed-forward network with a piecewise linear activation such as ReLU [106] is piecewise linear by definition [101]. We thus utilize these networks to represent locally linear models.

The key challenge lies in the fact that derivatives of functions parameterized by deep learning models are not stable in general [48]. State-of-the-art deep learning models [59, 66] are typically over-parametrized [162], leading to unstable functions as a by-product. The instability is reflected in both the function values [50] as well as the derivatives [48, 4]. As an immediate consequence of the unstable derivatives, the resulting local linear models also lack robustness [48, 4].

We note that gradient stability is a notion different from adversarial examples. A stable gradient can be large or small, so long as it remains (approximately) invariant within a local region. Adversarial examples, on the other hand, are small perturbations of the input that change the predicted output [50]. A large local gradient, whether stable or not in our sense, is likely to contribute to finding an adversarial example. Robust estimation techniques used to protect against adversarial examples (e.g., [95]) focus on stable function values rather than stable gradients but can nevertheless indirectly impact (potentially help) gradient stability. A direct extension of robust estimation to ensure gradient stability would involve finding maximally distorted derivatives and require access to approximate Hessians of deep networks.

Instead, we leverage the special structure of this class of networks (functional characteristics) to make the problem tractable. In particular, we infer lower bounds on the ℓ_p *margin*—the maximum radius of ℓ_p-norm balls around a point where derivatives are provably stable, and formulate a regularization problem to maximize it. The resulting objective is, however, rigid and non-smooth, and we further relax the learning problem in a manner resembling (locally) support vector machines (SVMs) [146, 30].

Both the inference and learning problems in our setting require evaluating the gradient of each neuron with respect to the inputs, posing a significant computational challenge. Given D-dimensional data, we propose a novel perturbation algorithm that collects all the *exact* gradients by means of forward propagating $D+1$ carefully crafted samples *in parallel* without any back-propagation. When the GPU memory cannot fit $D+1$ samples in one batch, we develop an unbiased approximation to the objective with a random subset of such samples.

Empirically, we examine our inference and learning algorithms with fully connected neural networks (FCNNs), residual networks (ResNets) [59], and recurrent neural networks (RNNs) on image and time-series datasets with quantitative and qualitative experiments. The main contributions of this work are as follows:

- Inference algorithms that identify input regions of neural networks, with piecewise linear activation functions, that are provably stable.

- A novel learning criterion that effectively expand regions of provably stable derivatives.
- Novel perturbation algorithms that scale computation to high dimensional data.
- Empirical evaluation with several types of networks.

3.2 Related Work

We focus in this chapter on neural networks with piecewise linear activation functions, such as ReLU [106], leaky ReLU [94], parametric ReLU [58], and other variants [51, 7]. Since the nonlinear behavior of deep models is governed by the activation function, a neural network defined with affine transformations and piecewise linear activation functions is inherently piecewise linear [101]. For example, FCNNs, convolutional neural networks (CNNs) [80], RNNs, ResNets [59], and densely connected networks (DenseNets) [66] are all plausible candidates under our consideration. We will call this class of networks *piecewise linear networks* throughout the thesis.

The proposed approach is based on a mixed integer linear representation of piecewise linear networks, *activation pattern* [116], which encodes the active linear piece (integer) of the activation function for each neuron; once an activation pattern is fixed, the network degenerates to a linear model (linear). Thus the feasible set corresponding to an activation pattern in the input space is a natural region where derivatives are provably stable (same linear function). Note the possible degenerate case where neighboring regions (with different activation patterns) nevertheless have the same end-to-end linear coefficients [126]. We call the feasible set induced by an activation pattern [126] a *linear region*, and a maximal connected subset of the input space subject to the same derivatives of the network [101] a *complete linear region.* Activation pattern has been studied in various contexts, such as visualizing neurons [43], reachability of a specific output value [93], its connection to vector quantization [9], counting the number of linear regions of piecewise linear networks [116, 100, 126], and adversarial attacks [24, 43, 151] or defense [155]. Note the distinction between locally linear regions of the functional mapping and decision regions defined by classes [155, 159, 99, 31].

Here we elaborate differences between our work and the two most relevant categories above. In contrast to quantifying the number of linear regions as a measure of complexity, we focus on the local linear regions, and try to expand them via learning. The notion of stability we consider differs from adversarial examples. The methods themselves are also different. Finding the exact adversarial example is in general NP-complete [70, 131], and mixed integer linear programs that compute the exact adversarial example do not scale [24, 43]. Layer-wise relaxations of ReLU activations [151, 155] are more scalable but yield bounds instead of exact solutions. Empirically, even relying on relaxations, the defense (learning) methods [155, 156] are still intractable on ImageNet scale images [33]. In contrast, our inference algorithm certifies the exact ℓ_p margin around a point subject to its activation pattern by forwarding $\Theta(D)$ samples in parallel. In a high-dimensional setting, where it is computationally challenging to compute the learning objective, we develop an unbiased estimation by a simple sub-sampling procedure, which scales to ResNet [59] on $299 \times 299 \times 3$ dimensional images in practice.

The proposed learning algorithm is based on the inference problem with ℓ_p margins. The derivation is reminiscent of the SVM objective [146, 30], but differs in its purpose; while SVM training seeks to maximize the ℓ_p margin between data points and a linear classifier, our approach instead maximizes the ℓ_p margin of linear regions around each data point. Since there is no label information to guide the learning algorithm for each linear region, the objective is unsupervised and more akin to transductive/semi-supervised SVMs (TSVMs) [147, 14]. In the literature, the idea of margin is also extended to nonlinear classifiers in terms of decision boundaries [41].

3.3 Methodology

The approaches are developed under the notation of fully connected DenseNets with ReLU activations, which subsumes other feed-forward architectures and naturally generalizes to other piecewise linear activation functions. We first introduce notation, and then present our inference and learning algorithms. All the proofs are provided in Appendix 3.A.

3.3.1 Notation

We consider a neural network θ with L hidden layers and M_i neurons in the i^{th} layer, and the corresponding function $f_\theta : \mathbb{R}^D \to \mathbb{R}^Y$ it represents. We use $\boldsymbol{z}^i \in \mathbb{R}^{M_i}$ and $\boldsymbol{a}^i \in \mathbb{R}^{M_i}$ to denote the vector of (raw) neurons and activated neurons in the i^{th} layer, respectively. We will use $\boldsymbol{x}$ and $\boldsymbol{a}^0$ interchangeably to represent an input instance from $\mathbb{R}^D = \mathbb{R}^{M_0}$. For generality, we consider the DenseNet architecture [66], where each hidden layer takes as input all the previous layers; it subsumes many existing feed-forward architectures such as FCNNs, CNNs [80], and ResNets [59]. The neurons $\boldsymbol{a}^i$ and $\boldsymbol{z}^i$ are computed with the transformation weight matrix $\boldsymbol{W}^i \in \mathbb{R}^{M_i \times \sum_{j=0}^{i-1} M_j}$ and the bias vector $\boldsymbol{b}^i \in \mathbb{R}^{M_i}$ as

$$\boldsymbol{a}^0 \triangleq \boldsymbol{x}, \;\; \boldsymbol{z}^i \triangleq \boldsymbol{W}^i \cdot \texttt{CONCATENATE}[\boldsymbol{a}^0, \boldsymbol{a}^1, ..., \boldsymbol{a}^{i-1}] + \boldsymbol{b}^i, \;\; \boldsymbol{a}^i \triangleq \sigma(\boldsymbol{z}^i), \forall i \in [L], \quad (3.1)$$

where $[L]$ denotes the set $\{1, \dots, L\}$ and $\sigma(\cdot)$ is a point-wise activation function. We use a subscript to further denote a specific neuron (e.g., $\boldsymbol{z}^i_j$ denotes the j^{th} neuron in the i^{th} layer). Note that both $\boldsymbol{a}$ and $\boldsymbol{z}$ are functions of the specific instance denoted by $\boldsymbol{x}$, where we drop the functional dependency to simplify notation. We use the set $\mathcal{I}$ to denote the set of all the neuron indices in this network $\{(i,j) | j \in [M_i], i \in [L]\}$. In this chapter, we will use ReLU [106] as a canonical example for the activation function

$$\boldsymbol{a}^i_j = \sigma(\boldsymbol{z}^i)_j \triangleq \max(0, \boldsymbol{z}^i_j), \forall (i,j) \in \mathcal{I}, \quad (3.2)$$

but the results naturally generalize to other piecewise linear activation functions [94, 58, 7]. The output of the network $f_\theta(\boldsymbol{x})$ is an affine transformation from all the hidden units $\boldsymbol{a}^0, \boldsymbol{a}^1, ..., \boldsymbol{a}^L$ to $\mathbb{R}^Y$. The output can be further processed by a nonlinearity such as softmax for classification problems. However, we focus on the piecewise linear property of neural networks represented by $f_\theta(\boldsymbol{x})$, and leverage a generic loss function $\mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y})$ to fold such nonlinear mechanism.

The activation pattern [116] used in this chapter is defined as:

Definition 3.1. *(Activation Pattern) An activation pattern is a set of indicators for neurons* $\bar{\mathcal{O}} = \{\bar{\boldsymbol{o}}^i \in \{-1, 1\}^{M_i} | i \in [L]\}$ *that specifies the functional constraints* $\boldsymbol{z}^i_j \bar{\boldsymbol{o}}^i_j \geq 0, \forall (i,j) \in \mathcal{I}$.

Each $\bar{\boldsymbol{o}}^i_j$ is called an activation indicator. Note that a point on the boundary of a linear region is feasible for multiple activation patterns (e.g., consider a network with a single neuron). The definition fits the property of the activation pattern discussed in §2.2. In particular, the feasible set of an activation pattern in the input space is a linear region of f_θ. We define $\nabla_{\boldsymbol{x}} \boldsymbol{z}^i_j$ to be the sub-gradient found by back-propagation using $\partial \boldsymbol{a}^{i'}_{j'} / \partial \boldsymbol{z}^{i'}_{j'} \triangleq \max(\bar{\boldsymbol{o}}^{i'}_{j'}, 0), \forall (i', j') \in \mathcal{I}$, whenever $\bar{\boldsymbol{o}}^{i'}_{j'}$ is defined in the context.

We use $\mathcal{D}$ to denote the set of training data $\{(\boldsymbol{x}, \boldsymbol{y})\}$, $\mathcal{D}_{\boldsymbol{x}}$ to denote the same set without labels $\boldsymbol{y}$, and $\mathcal{B}_{\epsilon,p}(\boldsymbol{x}) \triangleq \{\bar{\boldsymbol{x}} \in \mathbb{R}^D : \|\bar{\boldsymbol{x}} - \boldsymbol{x}\|_p \leq \epsilon\}$ to denote the ℓ_p-ball around $\boldsymbol{x}$ with radius ϵ.

3.3.2 Inference for Regions with Stable Derivatives

Although the activation pattern implicitly describes a linear region, it does not yield explicit constraints on the input space, making it hard to develop algorithms directly. Hence, we first derive an explicit characterization of the feasible set on the input space $\mathbb{R}^D$ with Lemma 3.2.[1]

Lemma 3.2. *Given an activation pattern* $\bar{\mathcal{O}}$ *with any feasible point* $\boldsymbol{x}$*, each activation*

[1]Similar characterization also appeared in [9].

indicator $\bar{\boldsymbol{o}}^i_j \in \bar{\mathcal{O}}$ *induces a feasible set*

$$\mathcal{S}^i_j(\boldsymbol{x}) = \{\bar{\boldsymbol{x}} \in \mathbb{R}^D : \bar{\boldsymbol{o}}^i_j[(\nabla_{\boldsymbol{x}} \boldsymbol{z}^i_j)^\top \bar{\boldsymbol{x}} + (\boldsymbol{z}^i_j - (\nabla_{\boldsymbol{x}} \boldsymbol{z}^i_j)^\top \boldsymbol{x})] \geq 0\},$$

and the feasible set of the activation pattern is equivalent to $\mathcal{S}(\boldsymbol{x}) = \cap_{i=1}^{M} \cap_{j=1}^{N_i} \mathcal{S}^i_j(\boldsymbol{x})$*.*

Remark 3.3. $\mathcal{S}(\boldsymbol{x})$ *can be characterized as a feasible set with a finite set of* linear *constraints with respect to the* input space $\mathbb{R}^D$*, so* $\mathcal{S}(\boldsymbol{x})$ *is a* convex polyhedron*.*

ℓ_p **margin.** Combining the linear property of f_θ subject to to an activation pattern with the input space constraints from Lemma 3.2 yields the definition of $\hat{\epsilon}_{\boldsymbol{x},p}$, the ℓ_p margin of a linear region of f_θ around $\boldsymbol{x}$ subject to its activation pattern:

$$\hat{\epsilon}_{\boldsymbol{x},p} \triangleq \max_{\epsilon \geq 0 : \mathcal{B}_{\epsilon,p}(\boldsymbol{x}) \subseteq \mathcal{S}(\boldsymbol{x})} \epsilon, \tag{3.3}$$

where $\mathcal{S}(\boldsymbol{x})$ can be based on *any* feasible activation pattern $\bar{\mathcal{O}}$ on $\boldsymbol{x}$, since $\hat{\epsilon}_{\boldsymbol{x},p}$ is always 0 when $\boldsymbol{x}$ has multiple feasible activation patterns $\bar{\mathcal{O}}$. Therefore, $\partial \boldsymbol{a}^i_j / \partial \boldsymbol{z}^i_j$ at $\boldsymbol{z}^i_j = 0$ from now on can take 0 or 1 arbitrarily as long as consistency among sub-gradients $\{\nabla_{\boldsymbol{x}} \boldsymbol{z}^i_j | (i,j) \in \mathcal{I}\}$ is ensured with respect to some feasible activation pattern $\bar{\mathcal{O}}$. Note that $\hat{\epsilon}_{\boldsymbol{x},p}$ is a lower bound of the ℓ_p margin subject to a derivative specification (i.e., a complete linear region).

Directional verification, the cases $p = 1$ **and** $p = \infty$**.** We first exploit the convexity of $\mathcal{S}(\boldsymbol{x})$ to check the feasibility of a directional perturbation.

Proposition 3.4. *Given a point* $\boldsymbol{x}$*, a feasible set* $\mathcal{S}(\boldsymbol{x})$ *and a unit vector* $\Delta\boldsymbol{x}$*, if* $\exists \bar{\epsilon} \geq 0$ *such that* $\boldsymbol{x} + \bar{\epsilon}\Delta\boldsymbol{x} \in \mathcal{S}(\boldsymbol{x})$*, then* f_θ *is linear in* $\{\boldsymbol{x} + \epsilon\Delta\boldsymbol{x} : 0 \leq \epsilon \leq \bar{\epsilon}\}$*.*

The feasibility of $\boldsymbol{x} + \bar{\epsilon}\Delta\boldsymbol{x} \in \mathcal{S}(\boldsymbol{x})$ can be computed by simply checking whether $\boldsymbol{x} + \bar{\epsilon}\Delta\boldsymbol{x}$ satisfies the activation pattern $\bar{\mathcal{O}}$ in $\mathcal{S}(\boldsymbol{x})$. Proposition 3.4 can be applied to the feasibility problem on ℓ_1-balls.

Proposition 3.5. *Given a point* $\boldsymbol{x}$*, a feasible set* $\mathcal{S}(\boldsymbol{x})$*, and an* ℓ_1*-ball* $\mathcal{B}_{\epsilon,1}(\boldsymbol{x})$ *with extreme points* $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^{2D}$*, if* $\boldsymbol{x}^i \in \mathcal{S}(\boldsymbol{x}), \forall i \in [2D]$*, then* f_θ *is linear in* $\mathcal{B}_{\epsilon,1}(\boldsymbol{x})$*.*

Proposition 3.5 can be extended to an ℓ_∞-ball. However, in high dimension D, the number of extreme points of an ℓ_∞-ball is exponential to D, making it intractable. Instead, the number of extreme points of an ℓ_1-ball is only linear to D ($+\epsilon$ and $-\epsilon$ for each dimension). With the above methods to verify feasibility, we can do binary searches to find the certificates of the margins for directional perturbations $\hat{\epsilon}_{\boldsymbol{x},\Delta\boldsymbol{x}} \triangleq \max_{\{\epsilon\geq 0:\boldsymbol{x}+\epsilon\Delta\boldsymbol{x}\in\mathcal{S}(\boldsymbol{x})\}}\epsilon$ and ℓ_1-balls $\hat{\epsilon}_{\boldsymbol{x},1}$. The details are in Appendix 3.B.

The case $p = 2$. The feasibility of ℓ_1-balls is tractable due to convexity of $\mathcal{S}(\boldsymbol{x})$ and its certification is efficient by a binary search; by further exploiting the polyhedron structure of $\mathcal{S}(\boldsymbol{x})$, $\hat{\epsilon}_{\boldsymbol{x},2}$ can be certified analytically.

Proposition 3.6. *Given a point $\boldsymbol{x}$, $\hat{\epsilon}_{\boldsymbol{x},2}$ is the minimum ℓ_2 distance between $\boldsymbol{x}$ and the union of hyperplanes $\cup_{i=1}^{M}\cup_{j=1}^{N_i}\{\bar{\boldsymbol{x}}\in\mathbb{R}^D : (\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i)^\top\bar{\boldsymbol{x}} + (\boldsymbol{z}_j^i - (\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i)^\top\boldsymbol{x}) = 0\}$.*

To compute the ℓ_2 distance between $\boldsymbol{x}$ and the hyperplane induced by a neuron $\boldsymbol{z}_j^i$, we evaluate $|(\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i)^\top\boldsymbol{x} + (\boldsymbol{z}_j^i - (\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i)^\top\boldsymbol{x})|/\|\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i\|_2 = |\boldsymbol{z}_j^i|/\|\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i\|_2$. Then $\hat{\epsilon}_{\boldsymbol{x},2}$ can be computed as $\hat{\epsilon}_{\boldsymbol{x},2} = \min_{(i,j)\in\mathcal{I}}|\boldsymbol{z}_j^i|/\|\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i\|_2$, where all the $\boldsymbol{z}_j^i$ can be computed by a single forward pass.[2] We will show in §3.4.1 that all the $\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i$ can also be computed efficiently by forward passes in parallel. We refer readers to Figure 3.3.1c to see a visualization of the certificates of ℓ_2 margins.

Number of complete linear regions. The sizes of linear regions are also related to their overall number, especially if we consider a bounded input space. Counting the number of linear regions in f_θ is, however, intractable due to the combinatorial nature of the activation patterns [126]. We argue that counting the number of linear regions on the whole space does not capture the structure of data manifold, and we propose to certify the number of complete linear regions (#CLR) of f_θ among the data points $\mathcal{D}_{\boldsymbol{x}}$, which turns out to be efficient to compute given a mild condition. Here we use $|\mathcal{A}|$ to denote the cardinality of a set $\mathcal{A}$, and we have

Lemma 3.7. *If every data point $\boldsymbol{x}\in\mathcal{D}_{\boldsymbol{x}}$ has only one feasible activation pattern denoted as $\mathcal{O}(\boldsymbol{x})$, the number of complete linear regions of f_θ among $\mathcal{D}_{\boldsymbol{x}}$ is upper-*

[2]Later, Croce *et al.* [31] found that the ℓ_p margin $\hat{\epsilon}_{\boldsymbol{x},p}$ can be similarly computed as $\min_{(i,j)\in\mathcal{I}}|\boldsymbol{z}_j^i|/\|\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^i\|_q$, where $\|\cdot\|_q$ is the dual norm of the ℓ_p-norm.

bounded by the number of different activation patterns $|\{\mathcal{O}(\boldsymbol{x})|\boldsymbol{x} \in \mathcal{D}_{\boldsymbol{x}}\}|$*, and lower-bounded by the number of different Jacobians* $|\{J_{\boldsymbol{x}} f_\theta(\boldsymbol{x})|\boldsymbol{x} \in \mathcal{D}_{\boldsymbol{x}}\}|$*.*

3.3.3 Learning: Maximizing the Margins of Stable Derivatives

In this section, we aim to maximize the ℓ_2 margin $\hat{\epsilon}_{\boldsymbol{x},2}$, which is (sub-)differentiable. We first formulate a regularization problem in the objective to maximize the margin:

$$\min_\theta \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \left[\mathcal{L}(f_\theta(\boldsymbol{x}),\boldsymbol{y}) - \lambda \min_{(i,j)\in\mathcal{I}} \frac{|\boldsymbol{z}_j^i|}{\|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2}\right] \tag{3.4}$$

However, the objective itself is rather rigid due to the inner-minimization and the reciprocal of $\|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2$. Qualitatively, such rigid loss surface hinders optimization and may attend infinity. To alleviate the problem, we do a hinge-based relaxation to the distance function similar to SVM.

Relaxation. An ideal relaxation of Eq. (3.4) is to disentangle $|\boldsymbol{z}_j^i|$ and $\|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2$ for a smoother problem. Our first attempt is to formulate an equivalent problem with special constraints which we can leverage later.

Lemma 3.8. *If there exists a (global) optimal solution of Eq. (3.4) that satisfies* $\min_{(i,j)\in\mathcal{I}} |\boldsymbol{z}_j^i| > 0, \forall \boldsymbol{x} \in \mathcal{D}_{\boldsymbol{x}}$*, then every optimal solution of Eq. (3.5) is also optimal for Eq. (3.4).*

$$\min_\theta \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathcal{L}(f_\theta(\boldsymbol{x}),\boldsymbol{y}) - \lambda \min_{(i,j)\in\mathcal{I}} \frac{|\boldsymbol{z}_j^i|}{\|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2}, \;\; s.t. \min_{(i,j)\in\mathcal{I}} |\boldsymbol{z}_j^i| \geq 1, \forall (\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}. \tag{3.5}$$

If the condition in Lemma 3.8 does not hold, Eq. (3.5) is still a valid upper bound of Eq. (3.4) due to a smaller feasible set. An upper bound of Eq. (3.5) can be obtained consequently due to the constraints:

$$\min_\theta \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathcal{L}(f_\theta(\boldsymbol{x}),\boldsymbol{y}) - \lambda \min_{(i,j)\in\mathcal{I}} \frac{1}{\|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2}, \;\; s.t. \min_{(i,j)\in\mathcal{I}} |\boldsymbol{z}_j^i| \geq 1, \forall (\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}. \tag{3.6}$$

We then derive a relaxation that solves a smoother problem by relaxing the squared root and reciprocal on the ℓ_2 norm as well as the hard constraint with a hinge loss to a soft regularization problem:

$$\min_{\theta} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathcal{L}(f_\theta(\boldsymbol{x}),\boldsymbol{y}) + \lambda \max_{(i,j)\in\mathcal{I}} \left[\|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2^2 + C\max(0, 1-|\boldsymbol{z}_j^i|)\right], \tag{3.7}$$

where C is a hyper-parameter. The relaxed regularization problem can be regarded as a maximum aggregation of TSVM losses among all the neurons, where a TSVM loss with only unannotated data $\mathcal{D}_{\boldsymbol{x}}$ can be written as:

$$\min_{\boldsymbol{w},b} \sum_{\boldsymbol{x}\in\mathcal{D}_{\boldsymbol{x}}} \|\boldsymbol{w}\|_2^2 + C\max(0, 1-|\boldsymbol{w}^\top\boldsymbol{x}+b|), \tag{3.8}$$

which pursues a similar goal to maximize the ℓ_2 margin in a linear model scenario, where the margin is computed between a linear hyperplane (the classifier) and the training points.

To visualize the effect of the proposed methods, we make a toy 2D binary classification dataset, and train a 4-layer FCNN with 1) (vanilla) binary cross-entropy loss $\mathcal{L}(\cdot,\cdot)$, 2) distance regularization as in Eq. (3.4), and 3) relaxed regularization as in Eq. (3.7). Implementation details are in Appendix 3.E. The resulting piecewise linear regions and prediction heatmaps along with gradient $\nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x})$ annotations are shown in Figure 3.3.1. The distance regularization enlarges the linear regions around each training point, and the relaxed regularization further generalizes the property to the whole space; the relaxed regularization possesses a smoother prediction boundary, and has a special central region where the gradients are $\mathbf{0}$ to allow gradients to change directions smoothly.

Improving sparse learning signals. Since a linear region is shaped by a set of neurons that are "close" to a given a point, a noticeable problem of Eq. (3.7) is that it only focuses on the "closest" neuron, making it hard to scale the effect to large networks. Hence, we make a generalization to the relaxed loss in Eq. (3.7) with a set of neurons that incur high losses to the given point. We denote $\hat{\mathcal{I}}(\boldsymbol{x},\gamma)$ as the set of

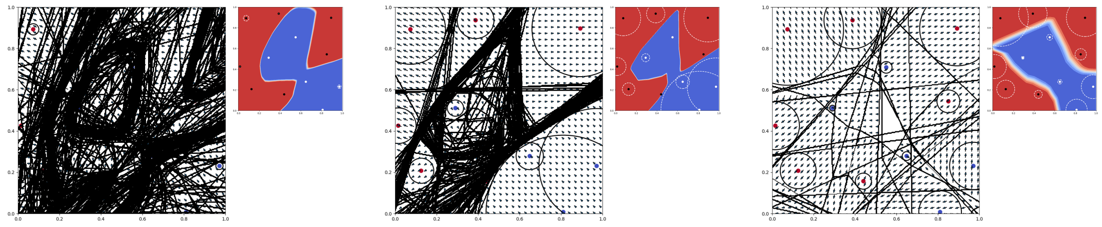

(a) Vanilla loss (b) Distance regularization (c) Relaxed regularization

Figure 3.3.1: Toy examples of a synthetic 2D classification task. For each model (regularization type), we show a prediction heatmap (smaller pane) and the corresponding locally linear regions. The boundary of each linear region is plotted with line segments, and each circle shows the ℓ_2 margin $\hat{\epsilon}_{\boldsymbol{x},2}$ around the training point. The gradient is annotated as arrows with length proportional to its ℓ_2 norm.

neurons with top γ percent relaxed loss (TSVM loss) on $\boldsymbol{x}$. The generalized loss is our final objective for learning RObust Local Linearity (ROLL) and is written as:

$$\min_{\theta} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathcal{L}(f_\theta(\boldsymbol{x}),\boldsymbol{y}) + \frac{\lambda}{|\hat{\mathcal{I}}(\boldsymbol{x},\gamma)|} \sum_{(i,j)\in\hat{\mathcal{I}}(\boldsymbol{x},\gamma)} \left[\|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2^2 + C\max(0, 1-|\boldsymbol{z}_j^i|)\right]. \quad (3.9)$$

A special case of Eq. (3.9) is when $\gamma = 100$ (i.e. $\hat{\mathcal{I}}(\boldsymbol{x},100) = \mathcal{I}$), where the nonlinear sorting step effectively disappears. Such simple additive structure without a nonlinear sorting step can stabilize the training process, is simple to parallelize computation, and allows for an approximate learning algorithm as will be developed in §3.4.2. Besides, taking $\gamma = 100$ can induce a strong synergy effect, as all the gradient norms $\|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2^2$ in Eq. (3.9) between any two layers are highly correlated.

3.4 Computation, Approximate Learning, and Compatibility

3.4.1 Parallel Computation of Gradients

The ℓ_2 margin $\hat{\epsilon}_{\boldsymbol{x},2}$ and the ROLL loss in Eq. (3.9) demands heavy computation on gradient norms. While calling back-propagation $|\mathcal{I}|$ times is intractable, we develop a parallel algorithm without calling a single back-propagation by exploiting the functional structure of f_θ.

Given an activation pattern, we know that each hidden neuron $\boldsymbol{z}_j^i$ is also a linear function of $\boldsymbol{x} \in \mathcal{S}(\boldsymbol{x})$. We can construct another *linear* network g_θ that is identical to f_θ in $\mathcal{S}(\boldsymbol{x})$ based on the same set of parameters but fixed linear activation functions mimicking f_θ in $\mathcal{S}(\boldsymbol{x})$. Due to the linearity of g_θ, the derivatives of all the neurons to an input axis can be computed by forwarding two samples: subtracting the neurons with an one-hot input from the same neurons with a zero input. The procedure can be amortized and parallelized to all the dimensions by feeding $D+1$ samples to g_θ in parallel. We remark that the algorithm generalizes to all the piecewise linear networks, and refer readers to Appendix 3.C for algorithmic details.[3]

To analyze the complexity of the proposed approach, we assume that parallel computation does not incur any overhead and a batch matrix multiplication takes a unit operation. To compute the gradients of all the neurons for a batch of inputs, our perturbation algorithm takes $2L$ operations, while back-propagation takes $\sum_{i=1}^{L} 2iM_i$ operations. The detailed analysis is also in Appendix 3.C.

3.4.2 Approximate Learning

Despite the parallelizable computation of $\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i$, it is still challenging to compute the loss for large networks in a high dimension setting, where even calling $D+1$ forward passes in parallel as described in §3.4.1 is infeasible due to memory constraints. Hence, we propose an unbiased estimator of the ROLL loss in Eq. (3.9) when $\hat{\mathcal{I}}(\boldsymbol{x}, \gamma) = \mathcal{I}$. Note that $\sum_{(i,j)\in\mathcal{I}} C \max(0, 1 - |\boldsymbol{z}_j^i|)$ is already computable in one single forward pass. For the sum of gradient norms, we use the following equivalent decoupling:

$$\frac{1}{|\mathcal{I}|} \sum_{(i,j)\in\mathcal{I}} \|\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i\|_2^2 = \frac{1}{|\mathcal{I}|} \sum_{k=1}^{D} \sum_{(i,j)\in\mathcal{I}} \left(\frac{\partial \boldsymbol{z}_j^i}{\partial \boldsymbol{x}_k}\right)^2 = \frac{D}{|\mathcal{I}|} \mathbb{E}_{k\sim \mathrm{Unif}([D])} \Bigg[\sum_{(i,j)\in\mathcal{I}} \left(\frac{\partial \boldsymbol{z}_j^i}{\partial \boldsymbol{x}_k}\right)^2 \Bigg], \quad (3.10)$$

where the summation inside the expectation in the last equation can be efficiently computed using the procedure in §3.4.1 and is in general storable within GPU memory. In practice, we can uniformly sample D' ($1 \leq D' \ll D$) input axes to have an unbiased

[3]When the network is fully connected (or can efficiently be represented as such), one can use a dynamic programming algorithm to compute the gradients [110].

approximation to Eq. (3.10), where computing all the partial derivatives with respect to D' axes only requires $D'+1$ times as much memory (one hot vectors and a zero vector) as the typical forward pass for $\boldsymbol{x}$.

3.4.3 Compatibility

The proposed algorithms can be used on all the deep learning models with affine transformations and piecewise linear activation functions by enumerating every neuron that will be imposed an ReLU-like activation function as $\boldsymbol{z}^i_j$. They do not immediately generalize to the nonlinearity of maxout/max-pooling [51] that also yields a piecewise linear function. We provide an initial step towards doing so in the Appendix 3.D, but we suggest to use an average-pooling or convolution with large strides instead, since they do not induce extra linear constraints as max-pooling and do not in general yield significant difference in performance [134].

3.5 Experiments

In this section, we compare our approach ('Roll') with a baseline model with the same training procedure except the regularization ('vanilla') in several scenarios. All the reported quantities are computed on a testing set. Experiments are run on a GPU with 12G memory.

3.5.1 MNIST

Evaluation measures: 1) accuracy (ACC), 2) the number of complete linear regions (#CLR), and 3) ℓ_p margins of linear regions $\hat{\epsilon}_{\boldsymbol{x},p}$. We compute the margin $\hat{\epsilon}_{\boldsymbol{x},p}$ for each testing point $\boldsymbol{x}$ with $p \in \{1,2\}$, and we evaluate $\hat{\epsilon}_{\boldsymbol{x},p}$ on 4 different percentiles $P_{25}, P_{50}, P_{75}, P_{100}$ among the testing data.

We use a $55,000/5,000/10,000$ split of the MNIST dataset for training/validation/testing. Experiments are conducted on a 4-layer FCNN with ReLU activations. The implementation details are in Appendix 3.F. We report the two models with the

Table 3.5.1: FCNN on the MNIST dataset. Here #CLR is the number of complete linear regions among the 10K testing points, and $\hat{\epsilon}_{\boldsymbol{x},p}$ shows the ℓ_p margin for each $r \in \{25, 50, 75, 100\}$ percentile P_r.

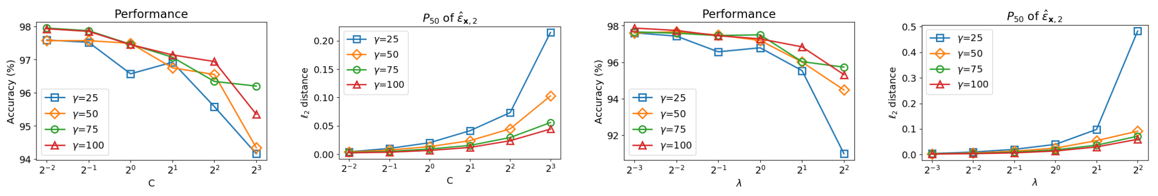
Loss	C	ACC	#CLR	$\hat{\epsilon}_{\boldsymbol{x},1}(\times 10^{-4})$				$\hat{\epsilon}_{\boldsymbol{x},2}(\times 10^{-4})$			
				P_{25}	P_{50}	P_{75}	P_{100}	P_{25}	P_{50}	P_{75}	P_{100}
Vanilla		98%	10000	22	53	106	866	3	6	13	91
ROLL	0.25	98%	9986	219	530	1056	6347	37	92	182	1070
ROLL	1.00	97%	8523	665	1593	3175	21825	125	297	604	4345

Table 3.5.2: Running time for a gradient descent step of FCNNs on the MNIST dataset. The full setting refers to Eq. (3.9) ($\gamma = 100$), and 3-samples refers to approximating Eq. (3.10) with 3 samples.

	Vanilla	ROLL (full; back-prop)	ROLL (full; perturb)	ROLL (3-samples; perturb)
second ($\times 10^{-5}$)	129	31185	2667	298

largest median $\hat{\epsilon}_{\boldsymbol{x},2}$ among validation data given the same and 1% less validation accuracy compared to the baseline model.

The results are shown in Table 3.5.1. The tuned models have $\gamma = 100, \lambda = 2$, and different C as shown in the table. The condition in Lemma 3.7 for certifying #CLR is satisfied with tight upper bound and lower bound, so a single number is reported. Given the same performance, the ROLL loss achieves about 10 times larger margins for most of the percentiles than the vanilla loss. By trading off 1% accuracy, about 30 times larger margins can be achieved. The Spearman's rank correlation between $\hat{\epsilon}_{\boldsymbol{x},1}$ and $\hat{\epsilon}_{\boldsymbol{x},2}$ among testing data is at least 0.98 for all the cases. The lower #CLR in our approach than the baseline model reflects the existence of certain larger linear regions that span across different testing points. All the points inside the same linear region in the ROLL model with ACC$=$ 98% have the same label, while there are visually similar digits (e.g., 1 and 7) in the same linear region in the other ROLL model. We do a parameter analysis in Figure 3.5.1 with the ACC and P_{50} of $\hat{\epsilon}_{\boldsymbol{x},2}$ under different C, λ and γ when the other hyper-parameters are fixed. As expected, with increased C and λ, the accuracy decreases with an increased ℓ_2 margin. Due to the smoothness of the curves, higher γ values reflect less sensitivity to hyper-parameters C and λ.



(a) $\lambda = 0.5$ (b) $\lambda = 0.5$ (c) $C = 1$ (d) $C = 1$

Figure 3.5.1: Parameter analysis on the MNIST dataset. P_{50} of $\hat{\epsilon}_{\boldsymbol{x},2}$ is the median of $\hat{\epsilon}_{\boldsymbol{x},2}$ in the testing data.

To validate the efficiency of the proposed method, we measure the running time for performing a complete mini-batch gradient descent step (starting from the forward pass) on average. We compare 1) the vanilla loss, 2) the full Roll loss ($\gamma = 100$) in Eq. (3.9) computed by back-propagation, 3) the same as 2) but computed by our perturbation algorithm, and 4) the approximate Roll loss in Eq. (3.10) computed by perturbations. The approximation is computed with $3 = D/256$ samples. The results are shown in Table 3.5.2. The accuracy and ℓ_2 margins of the approximate Roll loss are comparable to the full loss. Overall, our approach is only twice slower than the vanilla loss. The approximate loss is about 9 times faster than the full loss. Compared to back-propagation, our perturbation algorithm achieves about 12 times empirical speed-up. In summary, the computational overhead of our method is minimal compared to the vanilla loss, which is achieved by the perturbation algorithm and the approximate loss.

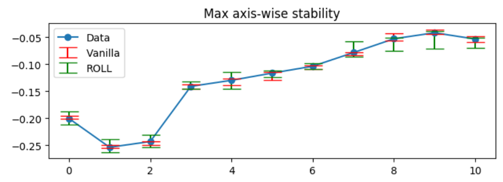
3.5.2 Speaker Identification

We train RNNs for speaker identification on the Japanese Vowel dataset from the UCI machine learning repository [34] with the official training/testing split.[4] The dataset has variable sequence length between 7 and 29 with 12 channels and 9 classes. We implement the network with the state-of-the-art scaled Cayley orthogonal RNN (scoRNN) [60], which parameterizes the transition matrix in RNN using orthogonal matrices to prevent gradient vanishing/exploding, with LeakyReLU activation. The

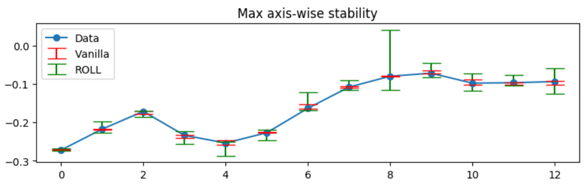
[4]The parameter is tuned on the testing set and thus the performance should be interpreted as validation.

Table 3.5.3: RNNs on the Japanese Vowel dataset. $\hat{\epsilon}_{\boldsymbol{x},p}$ shows the ℓ_p margin for each $r \in \{25, 50, 75, 100\}$ percentile P_r in the testing data (the larger the better).

Loss	λ	C	ACC	$\hat{\epsilon}_{\boldsymbol{x},1}(\times 10^{-6})$				$\hat{\epsilon}_{\boldsymbol{x},2}(\times 10^{-6})$			
				P_{25}	P_{50}	P_{75}	P_{100}	P_{25}	P_{50}	P_{75}	P_{100}
Vanilla			98%	66	177	337	1322	23	61	113	438
ROLL	2^{-5}	2^4	98%	264	562	1107	5227	95	207	407	1809
ROLL	2^{-1}	2^2	97%	1284	2898	6086	71235	544	1249	2644	22968



(a) The 9th channel of the sequence that yields P_{50} of $\hat{\epsilon}_{\boldsymbol{x},2}$ on the ROLL model.



(b) The 9th channel of the sequence that yields P_{75} of $\hat{\epsilon}_{\boldsymbol{x},2}$ on the ROLL model.

Figure 3.5.2: Stability bounds on derivatives on the Japanese Vowel dataset.

implementation details are in Appendix 3.G. The reported models are based on the same criterion as §3.5.1.

The results are reported in Table 3.5.3. With the same/1% inferior ACC, our approach leads to a model with about 4/20 times larger margins among the percentiles on testing data, compared to the vanilla loss. The Spearman's rank correlation between $\hat{\epsilon}_{\boldsymbol{x},1}$ and $\hat{\epsilon}_{\boldsymbol{x},2}$ among all the cases are 0.98. We also conduct sensitivity analysis on the derivatives by finding $\hat{\epsilon}_{\boldsymbol{x},\Delta\boldsymbol{x}}$ along each coordinate $\Delta\boldsymbol{x} \in \cup_i \cup_{j=1}^{12} \{-\boldsymbol{e}^{i,j}, \boldsymbol{e}^{i,j}\}$ ($\boldsymbol{e}^{i,j}_{k,l} = 0, \forall k, l$ except $\boldsymbol{e}^{i,j}_{i,j} = 1$), which identifies the stability bounds $[\hat{\epsilon}_{\boldsymbol{x},-\boldsymbol{e}^{i,j}}, \hat{\epsilon}_{\boldsymbol{x},\boldsymbol{e}^{i,j}}]$ at each timestamp i and channel j that guarantees stable derivatives. The visualization using the vanilla and our ROLL model with 98% ACC is in Figure 3.5.2. Qualitatively, the stability bound of the ROLL regularization is consistently larger than the vanilla model.

3.5.3 Caltech-256

We conduct experiments on Caltech-256 [54], which has 256 classes, each with at least 80 images. We downsize the images to $299 \times 299 \times 3$ and train a 18-layer ResNet [59] with initializing from parameters pre-trained on ImageNet [33]. The approximate ROLL loss in Eq. (3.10) is used with 120 random samples on each channel. We randomly select 5 and 15 samples in each class as the validation and testing set, respectively, and put the remaining data into the training set. The implementation details are in Appendix 3.H.

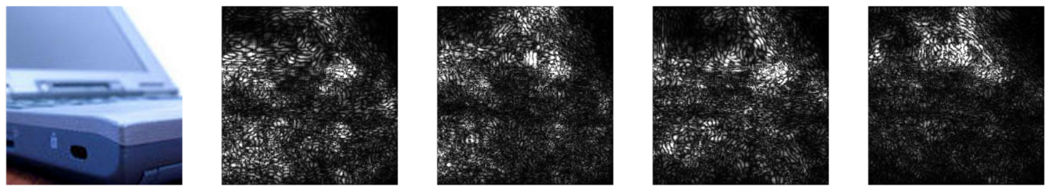
Evaluation Measures: Due to high input dimensionality ($D \approx 270K$), computing the certificates $\hat{\epsilon}_{\boldsymbol{x},1}, \hat{\epsilon}_{\boldsymbol{x},2}$ is computationally challenging without a cluster of GPUs. Hence, we turn to a sample-based approach to evaluate the stability of the gradients $f_\theta(\boldsymbol{x})_{\boldsymbol{y}}$ for the ground-truth label in a local region with a goal to reveal the stability across different linear regions. Note that evaluating the gradient of the prediction instead is problematic to compare different models in this case.

Given labeled data $(\boldsymbol{x}, \boldsymbol{y})$, we evaluate the stability of gradient $\nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x})_{\boldsymbol{y}}$ in terms of expected ℓ_1 distortion (over a uniform distribution) and the maximum ℓ_1 distortion within the intersection $\bar{\mathcal{B}}_{\epsilon,\infty}(\boldsymbol{x}) = \mathcal{B}_{\epsilon,\infty}(\boldsymbol{x}) \cap \mathcal{X}$ of an ℓ_∞-ball and the domain of images $\mathcal{X} = [0,1]^{299\times299\times3}$. The ℓ_1 gradient distortion is defined as $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y}) := \|\nabla_{\boldsymbol{x}'} f_\theta(\boldsymbol{x}')_{\boldsymbol{y}} - \nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x})_{\boldsymbol{y}}\|_1$. For a fixed $\boldsymbol{x}$, we refer to the maximizer $\nabla_{\boldsymbol{x}'} f_\theta(\boldsymbol{x}')_{\boldsymbol{y}}$ as the *adversarial gradient*. Computation of the maximum ℓ_1 distortion requires optimization, but gradient-based optimization is not applicable since the gradient of the loss involves the Hessian $\nabla^2_{\boldsymbol{x}'} f_\theta(\boldsymbol{x}')_{\boldsymbol{y}}$ which is either 0 or ill-defined due to piecewise linearity. Hence, we use a genetic algorithm [152] for black-box optimization. Implementation details are provided in Appendix 3.I. We use 8000 samples to approximate the expected ℓ_1 distortion. Due to computational limits, we only evaluate 1024 random images in the testing set for both maximum and expected ℓ_1 gradient distortions. The ℓ_∞-ball radius ϵ is set to 8/256.

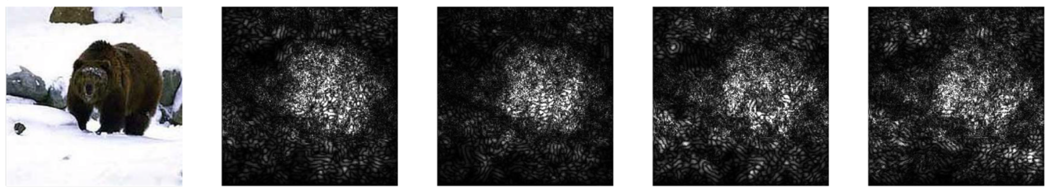
The results along with precision at 1 and 5 (P@1 and P@5) are presented in Table 3.5.4. The ROLL loss yields more stable gradients than the vanilla loss with marginally superior precisions. Out of 1024 examined examples $\boldsymbol{x}$, only 40 and 42 gradient-distorted images change prediction labels in the ROLL and vanilla model, respectively. We visualize some examples in Figure 3.5.3 with the original and adversarial gradients for each loss. Qualitatively, the ROLL loss yields stable shapes

Table 3.5.4: ResNet on Caltech-256. Here $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ denotes ℓ_1 gradient distortion $\|\nabla_{\boldsymbol{x}'} f_\theta(\boldsymbol{x}')_{\boldsymbol{y}} - \nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x})_{\boldsymbol{y}}\|_1$ (the smaller the better for each r percentile P_r among the testing data).

Loss	P@1	P@5	$\mathbb{E}_{\boldsymbol{x}' \sim \text{Unif}(\bar{\mathcal{B}}_{\epsilon,\infty}(\boldsymbol{x}))}[\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})]$				$\max_{\boldsymbol{x}' \in \bar{\mathcal{B}}_{\epsilon,\infty}(\boldsymbol{x})}[\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})]$			
			P_{25}	P_{50}	P_{75}	P_{100}	P_{25}	P_{50}	P_{75}	P_{100}
Vanilla	80.7%	93.4%	583.8	777.4	1041.9	3666.7	840.9	1118.2	1477.6	5473.5
ROLL	80.8%	94.1%	540.6	732.0	948.7	2652.2	779.9	1046.7	1368.2	3882.8



(a) Image (Laptop) (b) Orig. gradient (ROLL) (c) Adv. gradient (ROLL) (d) Orig. gradient (Vanilla) (e) Adv. gradient (Vanilla)



(f) Image (Bear) (g) Orig. gradient (ROLL) (h) Adv. gradient (ROLL) (i) Orig. gradient (Vanilla) (j) Adv. gradient (Vanilla)

Figure 3.5.3: Visualization of the examples in Caltech-256 that yield the P_{50} (above) and P_{75} (below) of the maximum ℓ_1 gradient distortions among the testing data on our ROLL model. The adversarial gradient is found by maximizing the distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ over the ℓ_∞-norm ball with radius 8/256.

and intensities of gradients, while the vanilla loss does not. More examples (with additional integrated gradient attributions) [139] are provided in Appendix 3.J.

3.6 Conclusion

This paper introduces a new learning problem to endow piecewise linear networks with robust local linearity. The central attempt is to construct locally transparent models, where the derivatives stably represent the local linear function and lends itself to be robust tools for further applications. We utilize piecewise linear networks and solve the problem based on a margin principle similar to SVM. Empirically, the proposed ROLL loss expands regions with provably stable derivatives, and further generalize the stable gradient property across linear regions.

3.A Proofs

3.A.1 Proof of Lemma 3.2

Proof. For $j \in [M_1]$, we have $\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^1 = \boldsymbol{W}_{j,:}^1, (\boldsymbol{z}_j^1 - (\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^1)^\top \boldsymbol{x}) = \boldsymbol{b}_j^1$. If $\bar{\boldsymbol{x}}$ is feasible to the fixed activation pattern $\bar{\boldsymbol{o}}_j^1$, it is equivalent to that $\bar{\boldsymbol{x}}$ satisfies the linear constraint

$$\bar{\boldsymbol{o}}_j^1[(\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i)^\top \bar{\boldsymbol{x}} + (\boldsymbol{z}_j^1 - (\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^1)^\top \boldsymbol{x})] = \bar{\boldsymbol{o}}_j^1[(\boldsymbol{W}_{j,:}^1)^\top \bar{\boldsymbol{x}} + \boldsymbol{b}_j^1] \geq 0 \tag{3.11}$$

in the first layer.

Assume $\bar{\boldsymbol{x}}$ has satisfied all the constraints before layer $i > 1$. We know if all the previous layers follows the fixed activation indicators, it is equivalent to rewrite each

$$\boldsymbol{a}_{j'}^{i'} = \max(0, \bar{\boldsymbol{o}}_{j'}^{i'}) \cdot \boldsymbol{z}_{j'}^{i'}, \forall j' \in [M_{i'}], i' \in [i-1]. \tag{3.12}$$

Then for $j \in [M_i]$, it is clear that $\boldsymbol{z}_j^i$ is a fixed linear function of $\boldsymbol{x}$ with linear weights equal to $\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i$ by construction. If $\bar{\boldsymbol{x}}$ is also feasible to the fixed activation indicator $\bar{\boldsymbol{o}}_j^i$, it is equivalent to that $\bar{\boldsymbol{x}}$ also satisfies the linear constraint

$$\bar{\boldsymbol{o}}_j^i[(\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i)^\top \bar{\boldsymbol{x}} + (\boldsymbol{z}_j^i - (\nabla_{\boldsymbol{x}} \boldsymbol{z}_j^i)^\top \boldsymbol{x})] \geq 0. \tag{3.13}$$

The proof follows by induction. □

3.A.2 Proof of Proposition 3.4

Proof. Since $\mathcal{S}(\boldsymbol{x})$ is a convex set and $\boldsymbol{x}, \boldsymbol{x} + \bar{\epsilon}\Delta\boldsymbol{x} \in \mathcal{S}(\boldsymbol{x})$, $\{\boldsymbol{x} + \epsilon\Delta\boldsymbol{x} : 0 \leq \epsilon \leq \bar{\epsilon}\} \subseteq \mathcal{S}(\boldsymbol{x})$. □

3.A.3 Proof of Proposition 3.5

Proof. $\mathcal{S}(\boldsymbol{x})$ is a convex set and $\boldsymbol{x}^i \in \mathcal{S}(\boldsymbol{x}), \forall i \in [2D]$. Hence, $\forall \boldsymbol{x}' \in \mathcal{B}_{\epsilon,1}(\mathbf{x})$, we know $\boldsymbol{x}'$ is a convex combination of $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^{2D}$, which implies $\boldsymbol{x}' \in \mathcal{S}(\boldsymbol{x})$. □

3.A.4 Proof of Proposition 3.6

Proof. Since $\mathcal{S}(\boldsymbol{x})$ is a convex polyhedron and $\boldsymbol{x} \in \mathcal{S}(\boldsymbol{x})$, $\mathcal{B}_{\epsilon,2}(\boldsymbol{x}) \subseteq \mathcal{S}(\boldsymbol{x})$ is equivalent to the statement: the hyperplanes induced from the linear constraints in $\mathcal{S}(\boldsymbol{x})$ are away from $\boldsymbol{x}$ for at least ϵ in ℓ_2 distance. Accordingly, the minimizing ℓ_2 distance between $\boldsymbol{x}$ and the hyperplanes is the maximizing distance that satisfies $\mathcal{B}_{\epsilon,2}(\boldsymbol{x}) \subseteq \mathcal{S}(\boldsymbol{x})$. □

3.A.5 Proof of Lemma 3.7

Proof. The number of different activation patterns is an upper bound since it counts the number of linear regions instead of the number of complete linear regions (a complete linear region can contain multiple linear regions). The number of different Jacobians is a lower bound since it only count the number of different linear coefficients $f_\theta(\boldsymbol{x})$ on $\mathcal{D}_{\boldsymbol{x}}$ without distinguishing whether they are in the same connected region. □

3.A.6 Proof of Lemma 3.8

Proof. The main idea is to construct a neural network feasible in Eq. (3.5) that has the same loss as the optimal model in Eq. (3.4). Since the optimum in Eq. (3.5) is lower-bounded by the optimum in Eq. (3.4) due to smaller feasible set, a model feasible in Eq. (3.5) and having the same loss as the optimum in Eq. (3.4) implies that it is also optimal in Eq. (3.5).

Given the optimal model f_θ in Eq. (3.4) satisfying the constraint $\min_{(i,j)\in\mathcal{I}} |\boldsymbol{z}_j^i| > 0, \forall(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$, we construct a model feasible in Eq. (3.5). For $i = 1, ..., L$, we compute the smallest neuron response $\delta_j^i = \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} |\boldsymbol{z}_j^i|, \forall j \in [M_i]$ in f_θ, and revise the parameters by the following rules:

$$\boldsymbol{W}^i_{j,:} \leftarrow \frac{1}{\delta^i_j}\boldsymbol{W}^i_{j,:},\ \boldsymbol{b}^i_j \leftarrow \frac{1}{\delta^i_j}\boldsymbol{b}^i_j, \forall j \in [M_i]. \tag{3.14}$$

We also scale all the weight values that take as input $\boldsymbol{z}^i_j$ by $\delta^i_j, \forall j \in [M_i]$. The rules only scale the neuron value of $\boldsymbol{z}^i_j$ without changing the value of all the higher layers, so the realized function of f_θ does not change.

That says, the constructed network achieves the optimal loss in Eq. (3.4) while being feasible in Eq. (3.5), so it is optimal in both Eq. (3.4) and Eq. (3.5). Since the other optimal solutions in Eq. (3.5) have the same loss as the constructed network, they are also optimal in Eq. (3.4). □

3.B Certificates of Directional Margin and ℓ_1 Margin

To certify the directional margin $\hat{\epsilon}_{\boldsymbol{x},\Delta\boldsymbol{x}} \triangleq \max_{\{\epsilon\geq 0:\boldsymbol{x}+\epsilon\Delta\boldsymbol{x}\in S(\boldsymbol{x})\}} \epsilon$ along a given $\Delta\boldsymbol{x}$, we can do a binary search between a lower bound ϵ^l and upper bound ϵ^u. We initialize $\epsilon^l_0 = 0$ and run a sub-routine to exponentially find an arbitrary upper bound ϵ^u_0 such that $\boldsymbol{x} + \epsilon^u_0\Delta\boldsymbol{x} \notin S(\boldsymbol{x})$. If ϵ^u_0 does not exist, we have $\hat{\epsilon}_{\boldsymbol{x},\Delta\boldsymbol{x}} = \infty$; otherwise, a binary search is executed by iterating t as

$$\begin{cases} \epsilon^l_{t+1} \triangleq 0.5(\epsilon^l_t + \epsilon^u_t), \epsilon^u_{t+1} \triangleq \epsilon^u_t, & \text{if } \boldsymbol{x} + 0.5(\epsilon^l_t + \epsilon^u_t)\Delta\boldsymbol{x} \in S(\boldsymbol{x}) \\ \epsilon^l_{t+1}(\boldsymbol{x}) \triangleq \epsilon^l_t, \epsilon^u_{t+1} \triangleq 0.5(\epsilon^l_t + \epsilon^u_t), & \text{otherwise} \end{cases} \tag{3.15}$$

Clearly, $\boldsymbol{x} + \epsilon^l_t\Delta\boldsymbol{x} \in S(\boldsymbol{x}), \boldsymbol{x} + \epsilon^u_t\Delta\boldsymbol{x} \notin S(\boldsymbol{x})$ always holds, and the gap between ϵ^l_t and ϵ^u_t decreases exponentially fast: $\epsilon^u_t - \epsilon^l_t = 0.5^t(\epsilon^u_0 - \epsilon^l_0)$, which upper-bounds $\hat{\epsilon}_{\boldsymbol{x},\Delta\boldsymbol{x}} - \epsilon^l_t \leq \epsilon^u_t - \epsilon^l_t$. In practice, we run the binary search with finite iteration T and return the lower bound ϵ^l_T with an identifiable bound $\epsilon^u_T - \epsilon^l_T$ from the optimal solution. In our experiments, we run the binary search algorithm until the bound is less than 10^{-7}.

If we denote $\boldsymbol{e}^1, \ldots, \boldsymbol{e}^D$ as the set of unit vector in each axis ($\boldsymbol{e}^i_i = 1, \boldsymbol{e}^i_j = 0, \forall j \neq i$), the margin $\hat{\epsilon}_{\boldsymbol{x},1}$ for the ℓ_1-ball $\mathcal{B}_{\epsilon,1}(\boldsymbol{x})$ can be certified by 1) computing the set of directional margins for each extreme point direction of an ℓ_1 ball $\{\hat{\epsilon}_{\boldsymbol{x},-\boldsymbol{e}^1}, \hat{\epsilon}_{\boldsymbol{x},\boldsymbol{e}^1}, \ldots, \hat{\epsilon}_{\boldsymbol{x},-\boldsymbol{e}^D}, \hat{\epsilon}_{\boldsymbol{x},\boldsymbol{e}^D}\}$, and 2) returning the minimum in the set as $\hat{\epsilon}_{\boldsymbol{x},1}$.

3.C Parallel Computation of the Gradients by Linearity

We denote the corresponding neurons $\boldsymbol{z}^i_j$ and $\boldsymbol{a}^i_j$ of f_θ in g_θ as $\hat{\boldsymbol{z}}^i_j(\hat{\boldsymbol{x}})$ and $\hat{\boldsymbol{a}}^i_j(\hat{\boldsymbol{x}})$ given $\hat{\boldsymbol{x}}$, highlighting its functional relationship with respect to a new input $\hat{\boldsymbol{x}}$. The network g_θ is constructed with exactly the same weights and biases as f_θ but with a carefully-crafted *linear* activation function $\hat{\boldsymbol{o}}^i_j = \max(0, \bar{\boldsymbol{o}}^i_j) \in \{0, 1\}$. Note that since $\bar{\boldsymbol{o}}$ is given, $\hat{\boldsymbol{o}}$ is fixed. Then each layer in g_θ is represented as:

$$\hat{\boldsymbol{a}}^i(\hat{\boldsymbol{x}}) = \hat{\boldsymbol{o}}^i \odot \hat{\boldsymbol{z}}^i(\hat{\boldsymbol{x}}), \;\; \hat{\boldsymbol{z}}^i(\hat{\boldsymbol{x}}) = \boldsymbol{W}^i \hat{\boldsymbol{z}}^{i-1}(\hat{\boldsymbol{x}}) + \boldsymbol{b}^i, \forall i \in [L]. \;\; \hat{\boldsymbol{z}}^0(\hat{\boldsymbol{x}}) = \hat{\boldsymbol{x}}. \tag{3.16}$$

We note that $\hat{\boldsymbol{a}}^i(\hat{\boldsymbol{x}}), \hat{\boldsymbol{o}}^i$, and $\hat{\boldsymbol{z}}^i(\hat{\boldsymbol{x}})$ are also functions of $\boldsymbol{x}$, which we omitted for simplicity. Since the new activation function $\hat{\boldsymbol{o}}$ is fixed given $\boldsymbol{x}$, effectively it applies the same linearity to $\hat{\boldsymbol{z}}^i_j$ as $\boldsymbol{z}^i_j$ in $\mathcal{S}(\boldsymbol{x})$ and each $\hat{\boldsymbol{z}}^i_j(\hat{\boldsymbol{x}})$ is *linear* to $\hat{\boldsymbol{x}}, \forall \hat{\boldsymbol{x}} \in \mathbb{R}^D$. As a direct result of linearity and the equivalence of g_θ and f_θ (and all the respective $\hat{\boldsymbol{z}}^i_j$ and $\boldsymbol{z}^i_j$) in $\mathcal{S}(\boldsymbol{x})$, we have to following equality:

$$\frac{\partial \hat{\boldsymbol{z}}^i_j(\hat{\boldsymbol{x}})}{\partial \hat{\boldsymbol{x}}_k} = \frac{\partial \boldsymbol{z}^i_j}{\partial \boldsymbol{x}_k}, \forall \hat{\boldsymbol{x}} \in \mathbb{R}^D. \tag{3.17}$$

We then do the following procedure to collect the partial derivatives with respect to an input axis k: 1) feed a zero vector $\mathbf{0}$ to g_θ to get $\hat{\boldsymbol{z}}^i_j(\mathbf{0})$ and 2) feed a one-hot vector on the k^{th} axis $\boldsymbol{e}^k$ to get $\hat{\boldsymbol{z}}^i_j(\boldsymbol{e}^k)$. Then the derivative of each neuron $\boldsymbol{z}^i_j$ with respect to $\boldsymbol{x}_k$ can be computed as

$$\hat{\boldsymbol{z}}^i_j(\boldsymbol{e}^k) - \hat{\boldsymbol{z}}^i_j(\mathbf{0}) = \left[\nabla_{\boldsymbol{e}^k} \hat{\boldsymbol{z}}^i_j(\boldsymbol{e}^k)^\top \boldsymbol{e}^k + \hat{\boldsymbol{z}}^i_j(\mathbf{0})\right] - \hat{\boldsymbol{z}}^i_j(\mathbf{0}) = \frac{\partial \hat{\boldsymbol{z}}^i_j(\boldsymbol{e}^k)}{\partial \boldsymbol{e}^k_k} \times 1 = \frac{\partial \boldsymbol{z}^i_j}{\partial \boldsymbol{x}_k},$$

where the first equality comes from the linearity of $\hat{\boldsymbol{z}}_j^i(\hat{\boldsymbol{x}})$ with respect to any $\hat{\boldsymbol{x}}$. With the procedure, the derivative of *all the neurons* to an input dimension can be computed with 2 forward passes, which can be further amortized and scaled by computing all the gradients of $\boldsymbol{z}_j^i$ with respect to all the D dimensions with $D + 1$ forward passes *in parallel*. We remark that the implementation is very simple and essentially the same across all the piecewise linear networks.

To analyze the complexity of the proposed approach, we assume that parallel computation does not incur any overhead and a batch matrix multiplication takes a unit operation. In this setting, a typical forward pass up to the last hidden layer takes L operations. To compute the gradients of all the neurons for a batch of inputs, our perturbation algorithm first takes a forward pass to obtain the activation patterns for the batch of inputs, and then takes another forward pass with perturbations to obtain the gradients. Since both forward passes are done up to the last hidden layers, it takes $2L$ operations in total. We remark that the efficiency can be further improved by combining the two forward passes since the weight matrices in f_θ and g_θ are the same for each layer, but we adopt the $2L$ operations version for simplicity.

In contrast, back-propagation cannot be parallelized among neurons, so computing the gradients of all the neurons must be done sequentially. For each neuron $\boldsymbol{z}_j^i$, it takes $2i$ operations for back-propagation to compute its gradient (i operations for each of the forward pass and the backward pass). Hence, it takes $\sum_{i=1}^{L} 2iM_i$ operations in total for back-propagations to compute the same thing.

3.D Derivations for Maxout/Max-Pooling Nonlinearity

Here we only make an introductory guide to the derivations for maxout/max-pooling nonlinearity. The goal is to highlight that it is feasible to derive inference and learning methods upon a (piecewise linear) network with max-pooling nonlinearity, but we do not suggest to use it since a max-pooling neuron would induce new linear constraints;

instead, we suggest to use convolution with large strides or average-pooling which do not incur any constraint.

For simplicity, we assume the target network has a single nonlinearity, which maps M neurons to 1 output by the maximum

$$\boldsymbol{a}_1^1 = \max(\boldsymbol{z}_1^1, \dots, \boldsymbol{z}_M^1), \boldsymbol{z}^1 = \boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1. \tag{3.18}$$

Then we can define the corresponding activation pattern $\bar{\boldsymbol{o}} = \bar{\boldsymbol{o}}_1^1 \in [M]$ as which input is selected:

$$\boldsymbol{z}_i^1 \geq \boldsymbol{z}_j^1, \forall j \neq i \in [M], \text{ if } \bar{\boldsymbol{o}}_1^1 = i. \tag{3.19}$$

It is clear to see once an activation pattern is fixed, the network again degenerates to a linear model, as the nonlinearity in the max-pooling effectively disappears. Such activation pattern induces a feasible set in the input space where derivatives are guaranteed to be stable, but such representation may have a similar degenerate case where two activation patterns yield the same linear coefficients.

The feasible set $\mathcal{S}(\boldsymbol{x})$ of a feasible activation pattern $\bar{\mathcal{O}} = \{\bar{\boldsymbol{o}}_1^1\}$ at $\boldsymbol{x}$ can be derived as:

$$\cap_{j\in[M]\setminus\{i\}}\{\bar{\boldsymbol{x}} \in \mathbb{R}^D : (\nabla_{\boldsymbol{x}}\boldsymbol{z}_i^1)^\top\bar{\boldsymbol{x}} + (\boldsymbol{z}_i^1 - (\nabla_{\boldsymbol{x}}\boldsymbol{z}_i^1)^\top\boldsymbol{x}) \geq (\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^1)^\top\bar{\boldsymbol{x}} + (\boldsymbol{z}_j^1 - (\nabla_{\boldsymbol{x}}\boldsymbol{z}_j^1)^\top\boldsymbol{x})\}. \tag{3.20}$$

To check its correctness, we know that Eq. (3.20) is equivalent to

$$\begin{aligned}\mathcal{S}(\boldsymbol{x}) &= \cap_{j\in[M]\setminus\{i\}}\{\bar{\boldsymbol{x}} \in \mathbb{R}^D : \boldsymbol{W}_i^1\bar{\boldsymbol{x}} + \boldsymbol{b}_i^1 \geq \boldsymbol{W}_j^1\bar{\boldsymbol{x}} + \boldsymbol{b}_j^1\} && (3.21)\\ &= \cap_{j\in[M]\setminus\{i\}}\{\bar{\boldsymbol{x}} \in \mathbb{R}^D : (\boldsymbol{W}_i^1 - \boldsymbol{W}_j^1)\bar{\boldsymbol{x}} + (\boldsymbol{b}_i^1 - \boldsymbol{b}_j^1) \geq 0\}, && (3.22)\end{aligned}$$

where the linear constraints are evident, and the feasible set is thus again a convex polyhedron. As a result, all the inference and learning algorithms can be applied with the linear constraints. Clearly, for each max-pooling neuron with M inputs, it will induce $M-1$ linear constraints.

3.E Implementation Details on the Toy Dataset

The FCNN consists of $L = 4$ fully-connected hidden layers, where each hidden layer has 100 neurons. The input dimension D is 2 and the output dimension Y is 1. The loss function $\mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y})$ is sigmoid cross entropy. We train the model for 5000 epochs with Adam [72] optimizer, and select the model among epochs based on the training loss. We fix $C = 5$, and increase $\lambda \in \{10^{-2}, \dots, 10^{2}\}$ for both the distance regularization and relaxed regularization problems until the resulting classifier is not perfect. The tuned λ in both cases are 1.

3.F Implementation Details on the MNIST Dataset

The data are normalized with $\mu = 0.1307$ and $\sigma = 0.3081$. We first compute the margin $\hat{\epsilon}_{\boldsymbol{x},p}$ in the normalized data, and report the scaled margin $\sigma\hat{\epsilon}_{\boldsymbol{x},p}$ in the table, which reflects the actual margin in the original data space since

$$\|\boldsymbol{x} - \boldsymbol{x}'\|_p = \sigma\|\boldsymbol{x}/\sigma - \boldsymbol{x}'/\sigma\|_p = \sigma\|(\boldsymbol{x} - \mu)/\sigma - (\boldsymbol{x}' - \mu)/\sigma\|_p, \tag{3.23}$$

so the reported margin should be perceived in the data space of $\mathcal{X} = [0, 1]^{28\times 28}$.

We compute the exact ROLL loss during training (i.e., approximate learning is not used). The FCNN consists of $L = 4$ fully-connected hidden layers, where each hidden layer has 300 neurons. The activation function is ReLU. The loss function $\mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y})$ is a cross-entropy loss with softmax performed on $f_\theta(\boldsymbol{x})$. The number of epochs is 20, and the model is chosen from the best validation loss from all the epochs. We use stochastic gradient descent with Nesterov momentum. The learning rate is 0.01, the momentum is 0.5, and the batch size is 64.

Tuning: We do a grid search on λ, C, γ, with $\lambda \in \{2^{-3}, \dots, 2^{2}\}$, $C \in \{2^{-2}, \dots, 2^{3}\}$, $\gamma \in \{\max, 25, 50, 75, 100\}$ (max refers to Eq. (3.7)), and report the models with the largest validation $\hat{\epsilon}_{2,50}$ given the same and 1% less validation accuracy compared to the baseline model (the vanilla loss).

3.G Implementation Details on the Japanese Vowel Dataset

The data are not normalized.

We compute the exact ROLL loss during training (i.e., approximate learning is not used). The representation is learned with a single layer scoRNN, where the state embedding from the last timestamp for each sequence is treated as the representation along with a fully-connected layer to produce a prediction as $f_\theta(\boldsymbol{x})$. We use LeakyReLU as the activation functions in scoRNN. The dimension of hidden neurons in scoRNN is set to 512. The loss function $\mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y})$ is a cross-entropy loss with softmax performed on $f_\theta(\boldsymbol{x})$. We use AMSGrad optimizer [119]. The learning rate is 0.001, and the batch size is 32 (sequences).

Tuning: We do a grid search on $\lambda \in \{2^{-6}, \ldots, 2^{3}\}$, $C \in \{2^{-5}, \ldots, 2^{7}\}$, and set $\gamma = 100$. The models with the largest testing $\hat{\epsilon}_{2,50}$ given the same and 1% less testing accuracy compared to the baseline model (the vanilla loss) are reported. (We do not have validation data in this dataset, so the performance should be interpreted as validation.)

3.H Implementation Details on the Caltech-256 Dataset

The data are normalized with

$$\mu = [0.485, 0.456, 0.406], \text{ and } \sigma = [0.225, 0.225, 0.225]$$

along each channel. We train models on the normalized images, and establish a bijective mapping between the normalized distance and the distance in the original space with the trick introduced in Appendix 3.F. The bijection is applied to our sample-based approach to compute

$$\mathbb{E}_{\boldsymbol{x}' \sim \mathrm{Unif}(\mathcal{B}_{8/256,\infty}(\boldsymbol{x}) \cap \mathcal{X})}[\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})]$$

and

$$\max_{\boldsymbol{x}' \in \mathcal{B}_{8/256,\infty}(\boldsymbol{x}) \cap \mathcal{X}} [\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})]$$

that we ensure the perturbed space is consistent with $\mathcal{X} = [0,1]^{299\times 299\times 3}$ and $\mathcal{B}_{8/256,\infty}(\boldsymbol{x})$ in the original space.

We download the pre-trained ResNet-18 [59] from PyTorch [113], and we revise the model architecture as follows: 1) we replace the max-pooling after the first convolutional layer with average-pooling to reduce the number of linear constraints (because max-pooling induces additional linear constraints on activation pattern, while average-pooling does not), and 2) we enlarge the receptive field of the last pooling layer such that the output will be 512 dimension, since ResNet-18 is originally used for smaller images in ImageNet data (most implementations use $224\times 224\times 3$ dimensional images for ImageNet while our data has even higher dimension $299 \times 299 \times 3$).

We train the model with stochastic gradient descent with Nesterov momentum for 20 epochs. The initial learning rate is 0.005, which is adjusted to 0.0005 after the first 10 epochs. The momentum is 0.5. The batch size is 32. The model achieving the best validation loss among the 20 epochs is selected.

Tuning: Since the training is computationally demanding, we first fix $C = 8$, use only 18 samples (6 per channel) for approximate learning, and tune $\lambda \in \{10^{-6}, 10^{-5}, \dots\}$ until the model yields significantly inferior validation accuracy than the vanilla model. Afterwards, we fix λ to the highest plausible value ($\lambda = 0.001$) and try to increase $C \in \{8, 80, \dots\}$, but we found that $C = 8$ is already the highest plausible value. Finally, we train a model with 360 random samples (120 per channel) for approximate learning to improve the quality of approximation.

3.I Implementation Details of the Genetic Algorithm

We implement a genetic algorithm (GA) [152] with 4800 populations $\boldsymbol{P}$ and 30 epochs. Initially, we first uniformly sample 4800 samples (called chromosome in GA literature) in the domain $\mathcal{B}_{\epsilon,\infty}(\boldsymbol{x}) \cap \mathcal{X}$ for $\boldsymbol{P}$. In each epoch,

1. $\forall \boldsymbol{c} \in \boldsymbol{P}$, we evaluate the ℓ_1 distance of its gradient from that of the target $\boldsymbol{x}$:

$$\|\nabla_{\boldsymbol{c}} f_\theta(\boldsymbol{c})_{\boldsymbol{y}} - \nabla_{\boldsymbol{x}} f_\theta(\boldsymbol{x})_{\boldsymbol{y}}\|_1 \tag{3.24}$$

2. (Selection) we sort the samples based on the ℓ_1 distance and keep the top 25% samples in the population (denoted as $\hat{\boldsymbol{P}}$).

3. (Crossover) we replace the remaining 75% samples with a random linear combination of a pair $(\boldsymbol{c}, \boldsymbol{c}')$ from $\hat{\boldsymbol{P}}$ as:

$$\bar{\boldsymbol{c}} = \alpha \boldsymbol{c} + (1-\alpha)\boldsymbol{c}', \boldsymbol{c}, \boldsymbol{c}' \in \hat{\boldsymbol{P}}, \alpha \in \text{Unif}([-0.25, 1.25]). \tag{3.25}$$

4. (Projection) For all the updated samples $\boldsymbol{c} \in \boldsymbol{P}$, we do an ℓ_∞-projection to the domain $\mathcal{B}_{\epsilon,\infty}(\boldsymbol{x}) \cap \mathcal{X}$ to ensure the feasibility.

Finally, the sample in $\boldsymbol{P}$ that achieves the maximum ℓ_1 distance is returned. We didn't implement mutation in our GA algorithm due to computational reasons. For the readers who are not familiar with GA, we comment that the crossover operator is analogous to a gradient step where the direction is determined by other samples and the step size is determined randomly.

3.J Visualization of Adversarial Gradients in the Caltech-256 dataset

We visualize the following images:

- Original image.

- Original gradient: the gradient on the original image.
- Adv. gradient: the maximum ℓ_1 distorted gradient in $\mathcal{B}_{\epsilon,\infty}(\boldsymbol{x}) \cap \mathcal{X}$.
- Image of adv. gradient: the image that yields the adversarial gradient.
- Original int. gradient: the integrated gradient attribution [139] on the original image.
- Adv. int. gradient: the integrated gradient attribution [139] on the 'image of adv. gradient'. Note that we didn't perform optimization to find the image that yields the maximum distorted integrated gradient.

We follow a common implementation in the literature [132, 139] to visualize gradients and integrated gradients by the following procedure:

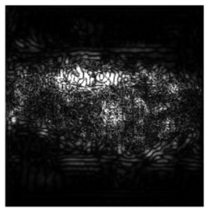
1. Aggregating derivatives in each channel by summation.
2. Taking absolute value of aggregated derivatives.
3. Normalizing the aggregated derivatives by the 99th percentile.
4. Clipping all the values above 1.

After this, the derivatives are in the range $[0, 1]^{299\times 299}$, which can be visualized as a gray-scaled image. The original integrated gradient paper visualizes the element-wise product between the gray-scaled integrated gradient and the original image, but we only visualize the integrated gradient to highlight its difference in different settings since the underlying images (the inputs) are visually indistinguishable.

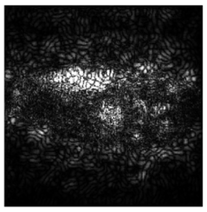
We visualize the examples in Caltech-256 dataset that yield the $P_{25}, P_{50}, P_{75}, P_{100}$ (P_r denotes the r^{th} percentile) of the maximum ℓ_1 gradient distortions among the testing data on our ROLL model in Figure 3.J.1-3.J.4, where the captions show the exact values of the maximum ℓ_1 gradient distortion for each image. Note that the exact values are slightly different from Table 3.5.4, because each percentile in Table 3.5.4 is computed by an interpolation between the closest ranks (as in `numpy.percentile`), and the figures in Figure 3.J.1-3.J.4 are chosen from the images that are the closest to the percentiles.

(a) Original image (Projector)

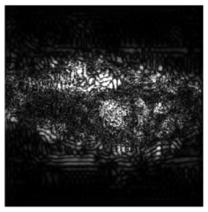
(b) Image of adv. gradient (Roll)



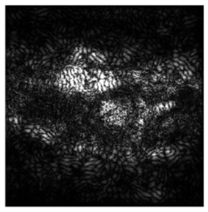
(c) Original gradient (Roll)



(d) Adv. gradient (Roll)

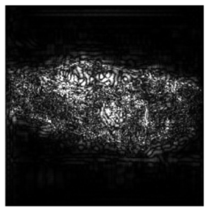


(e) Original gradient (Vanilla)

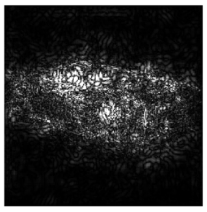


(f) Adv. gradient (Vanilla)

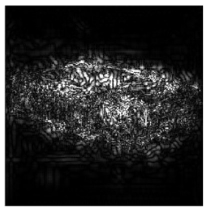
(g) Image of adv. gradient (Vanilla)



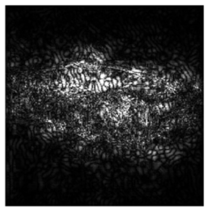
(h) Original int. gradient (Roll)



(i) Adv. int. gradient (Roll)



(j) Original int. gradient (Vanilla)

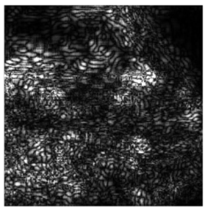


(k) Adv. int. gradient (Vanilla)

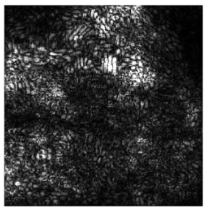
Figure 3.J.1: Visualization of the examples in Caltech-256 dataset that yield the P_{25} of the maximum ℓ_1 gradient distortions among the testing data using our Roll model. For the vanilla model, the maximum ℓ_1 gradient distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ is equal to 893.3 in Figure 3.J.1f. For the Roll model, the maximum ℓ_1 gradient distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ is equal to 779.9 in Figure 3.J.1d.

(a) Original image (Laptop)

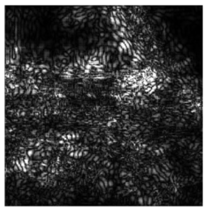
(b) Image of adv. gradient (Roll)



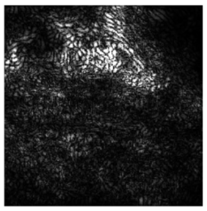
(c) Original gradient (Roll)



(d) Adv. gradient (Roll)

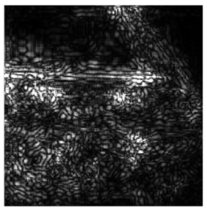


(e) Original gradient (Vanilla)

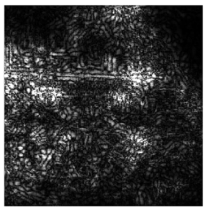


(f) Adv. gradient (Vanilla)

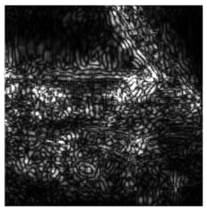
(g) Image of adv. gradient (Vanilla)



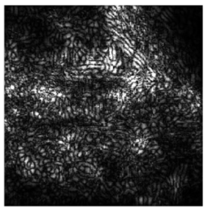
(h) Original int. gradient (Roll)



(i) Adv. int. gradient (Roll)



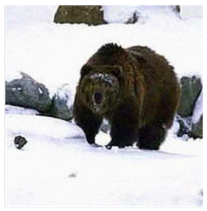
(j) Original int. gradient (Vanilla)



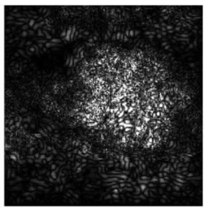
(k) Adv. int. gradient (Vanilla)

Figure 3.J.2: Visualization of the examples in Caltech-256 dataset that yield the P_{50} of the maximum ℓ_1 gradient distortions among the testing data using our Roll model. For the vanilla model, the maximum ℓ_1 gradient distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ is equal to 1199.4 in Figure 3.J.2f. For the Roll model, the maximum ℓ_1 gradient distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ is equal to 1045.4 in Figure 3.J.2d.

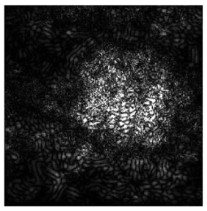
(a) Original image (Bear)



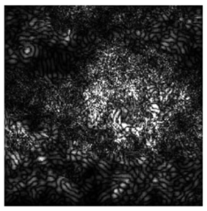
(b) Image of adv. gradient (Roll)



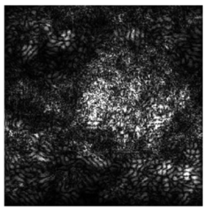
(c) Original gradient (Roll)



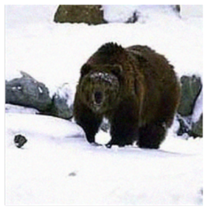
(d) Adv. gradient (Roll)



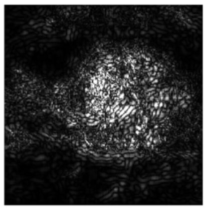
(e) Original gradient (Vanilla)



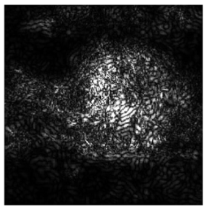
(f) Adv. gradient (Vanilla)



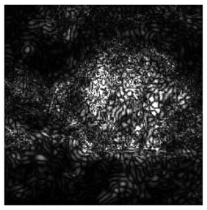
(g) Image of adv. gradient (Vanilla)



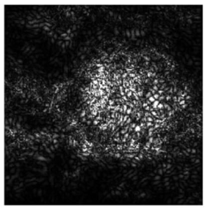
(h) Original int. gradient (Roll)



(i) Adv. int. gradient (Roll)



(j) Original int. gradient (Vanilla)

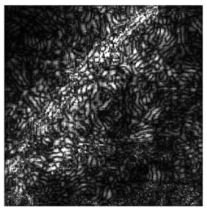


(k) Adv. int. gradient (Vanilla)

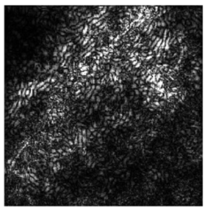
Figure 3.J.3: Visualization of the examples in Caltech-256 dataset that yield the P_{75} of the maximum ℓ_1 gradient distortions among the testing data using our Roll model. For the vanilla model, the maximum ℓ_1 gradient distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ is equal to 1547.1 for in Figure 3.J.3f. For the Roll model, the maximum ℓ_1 gradient distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ is equal to 1367.9 for in Figure 3.J.3d.

(a) Original image (Rainbow)

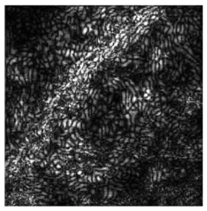
(b) Image of adv. gradient (ROLL)



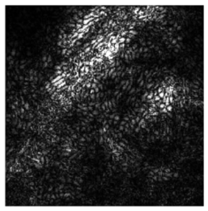
(c) Original gradient (ROLL)



(d) Adv. gradient (ROLL)

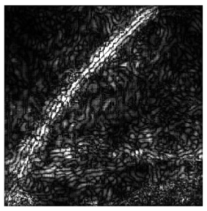


(e) Original gradient (Vanilla)

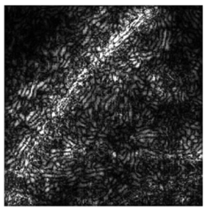


(f) Adv. gradient (Vanilla)

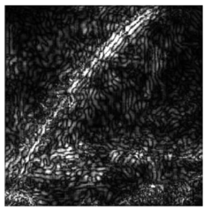
(g) Image of adv. gradient (Vanilla)



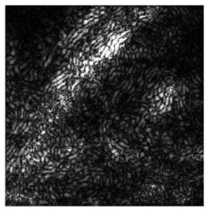
(h) Original int. gradient (ROLL)



(i) Adv. int. gradient (ROLL)



(j) Original int. gradient (Vanilla)



(k) Adv. int. gradient (Vanilla)

Figure 3.J.4: Visualization of the examples in Caltech-256 dataset that yield the P_{100} of the maximum ℓ_1 gradient distortions among the testing data using our ROLL model. For the vanilla model, the maximum ℓ_1 gradient distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ is equal to 5473.5 in Figure 3.J.4f. For the ROLL model, the maximum ℓ_1 gradient distortion $\Delta(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{y})$ is equal to 3882.8 in Figure 3.J.4d.

Chapter 4

Oblique Decision Trees from Derivatives of ReLU Networks

4.1 Introduction

Prior work has attempted to generalize classic decision trees by extending coordinate cuts to be weighted, linear classifications. The resulting family of models is known as oblique decision trees [105]. However, the generalization accompanies a challenging combinatorial, non-differentiable optimization problem over the linear parameters at each decision point. Simple sorting procedures used for successively finding branch-wise optimal coordinate-wise cuts are no longer available, making these models considerably harder to train. While finding the optimal oblique decision tree can be cast as a mixed integer linear program [15], scaling remains a challenge.

In this chapter, we provide an effective, implicit representation of piecewise constant mappings such as oblique decision trees, termed *locally constant networks*. Our approach exploits piecewise linear models such as ReLU networks as basic building blocks. Linearity of the mapping in each region in such models means that the gradient with respect to the input coordinates is locally constant. We therefore implicitly represent locally constant networks through gradients evaluated from ReLU networks. We prove the equivalence between the class of oblique decision trees and these proposed locally constant neural models. However, the sizes required for equivalent representations can be substantially different. For example, a locally constant network with $\tilde{M}$ neurons can implicitly realize an oblique decision tree whose explicit form requires $2^{\tilde{M}} - 1$ oblique decision nodes. The exponential complexity reduction in the corresponding neural representation illustrates the degree to which parameters are shared across the locally constant regions.

Our locally constant networks can be learned via gradient descent, and they can be explicitly converted to oblique decision trees for interpretability. For learning via gradient descent, however, it is necessary to employ some smooth annealing of piecewise linear activation functions so as to keep the gradients themselves continuous. Moreover, we need to evaluate the gradients of all the neurons with respect to the inputs. To address this bottleneck, we devise a dynamic programming algorithm which computes all the necessary gradient information in a single forward pass. A number of extensions are possible. For instance, we can construct *approximately* locally constant networks by switching activation functions, or apply helpful techniques used with normal deep learning models (e.g., DropConnect [149]) while implicitly training tree models.

We empirically test our model in the context of molecular property classification and regression tasks [158], where tree-based models remain state-of-the-art. We compare our approach against recent methods for training oblique decision trees and classic ensemble methods such as gradient boosting [46] and random forest [19]. Empirically, a locally constant network always outperforms alternative methods for training oblique decision trees by a large margin, and the ensemble of locally constant networks is competitive with classic ensemble methods.

4.2 Related Work

Locally constant networks use the gradients of deep networks with respect to inputs as the representations to build discriminative models. Such gradients have been used in literature for different purposes. They have been widely used for local sensitivity analysis of trained networks [129, 132]. When the deep networks model an energy function [81], the gradients can be used to draw samples from the distribution specified by the normalized energy function [35, 133]. The gradients can also be used to train generative models [49] or perform knowledge distillation [135].

The class of locally constant networks is equivalent to the class of oblique decision trees. There are some classic methods that also construct neural networks that reproduce decision trees [127, 20, 26], by utilizing step functions and logic gates (e.g., `AND`/`NEGATION`) as the activation function. The methods were developed when back-propagation was not yet practically useful, and the motivation is to exploit effective learning procedures of decision trees to train neural networks. Instead, our goal is to leverage the successful deep models to train oblique decision trees. Recently, Yang *et al.* [160] proposed a network architecture with $\arg\max$ activations to represent classic decision trees with coordinate cuts, but their parameterization scales exponentially with input dimension. In stark contrast, our parameterization only scales linearly with input dimension (see our complexity analyses in §4.3.5).

Learning oblique decision trees is challenging, even for a greedy algorithm; for a single oblique split, there can be $\sum_{k=0}^{D} \binom{N}{k}$ different ways to separate N data points in D-dimensional space [145] (cf. ND possibilities for coordinate-cuts). Existing learning algorithms for oblique decision trees include greedy induction, global optimization, and iterative refinements on an initial tree. We review some representative works, and refer the readers to the references therein.

Optimizing each oblique split in greedy induction can be realized by coordinate descent [104] or a coordinate-cut search in some linear projection space [97, 153]. However, the greedy constructions tend to get stuck in poor local optimum. There are some works which attempt to find the global optimum given a fixed tree structure by formulating a linear program [13] or a mixed integer linear program [15], but the methods are not scalable to ordinary tree sizes (e.g., depth more than 4). The iterative refinements are more scalable than global optimization, where CART [18] is the typical initialization. Carreira-Perpinán & Tavallali [23] developed an alternating optimization method via iteratively training a linear classifier on each decision node, which yield the state-of-the-art empirical performance, but the approach is only applicable to classification problems. Norouzi *et al.* [109] proposed to do gradient descent on a sub-differentiable upperbound of tree prediction errors, but the gradients with respect to oblique decision nodes are unavailable whenever the upperbound is tight. In contrast, our method conducts gradient descent on a differentiable relaxation, which is gradually annealed to a locally constant network.

4.3 Methodology

In this section, we construct the locally constant networks in §4.3.1, analyze the networks in §4.3.2-4.3.3, and develop practical formulations and algorithms in §4.3.4-4.3.5. Note that we will propose two (equivalent) architectures of locally constant networks in §4.3.1 and §4.3.4, which are useful for theoretical analyses and practical purposes, respectively.

In this chapter, we inherit the same notation as in §3.3.1 except for the activation pattern, which is re-defined as the collection of activation indicator *functions* for each neuron $\boldsymbol{o}^i_j : \mathbb{R}^D \to \{0,1\}, \forall (i,j) \in \mathcal{I}$ (or, equivalently, the derivatives of ReLU units; see below):

$$\boldsymbol{o}^i_j = \frac{\partial \boldsymbol{a}^i_j}{\partial \boldsymbol{z}^i_j} \triangleq \mathbb{I}[\boldsymbol{z}^i_j \geq 0], \forall (i,j) \in \mathcal{I}, \tag{4.1}$$

where $\mathbb{I}[\cdot]$ is the indicator function. Note that each $\boldsymbol{o}^i_j$ is a function of $\boldsymbol{x}$, where we omit the dependency for brevity. For mathematical correctness, we *define* $\partial \boldsymbol{a}^i_j / \partial \boldsymbol{z}^i_j = 1$ at $\boldsymbol{z}^i_j = 0$; this choice is arbitrary, and one can change it to $\partial \boldsymbol{a}^i_j / \partial \boldsymbol{z}^i_j = 0$ at $\boldsymbol{z}^i_j = 0$ without affecting most of the derivations.

4.3.1 Canonical Locally Constant Networks

Since the ReLU network $f_\theta(\boldsymbol{x})$ is piecewise linear, it immediately implies that its derivatives with respect to the input $\boldsymbol{x}$ is a piecewise constant function. Here we use $J_{\boldsymbol{x}} f_\theta(\boldsymbol{x}) \in \mathbb{R}^{Y \times D}$ to denote the Jacobian matrix (i.e., $[J_{\boldsymbol{x}} f_\theta(\boldsymbol{x})]_{i,j} = \partial f_\theta(\boldsymbol{x})_i / \partial \boldsymbol{x}_j$), and we assume the Jacobian is consistent with Eq. (4.1) at the boundary of the locally

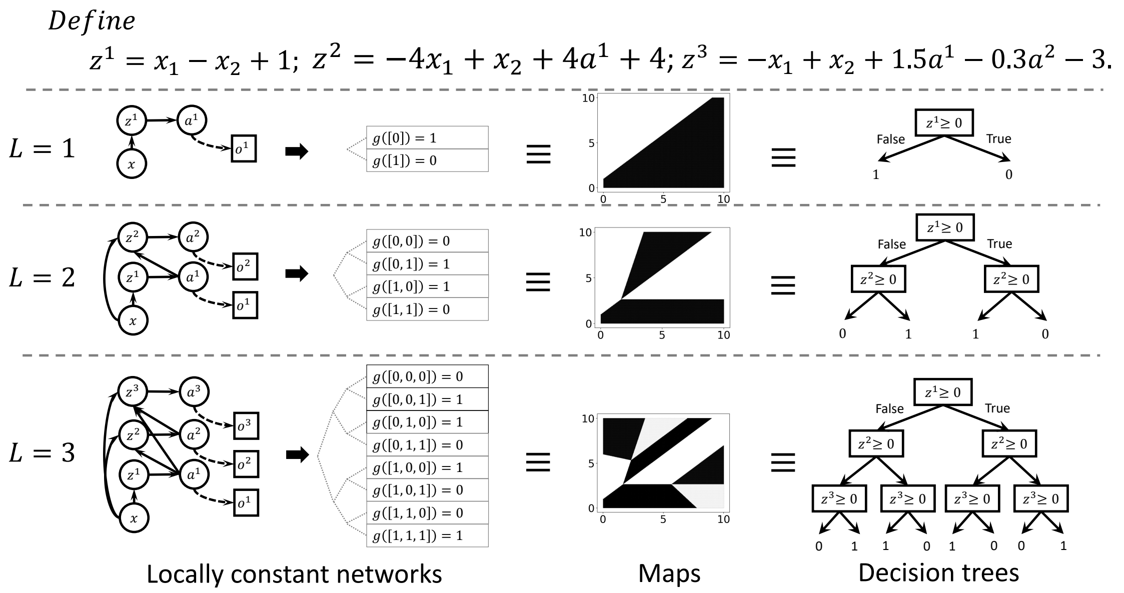


Figure 4.3.1: Toy examples for the equivalent representations of the same mappings for different L. Here the locally constant networks have 1 neuron per layer. We show the locally constant networks on the LHS, the raw mappings in the middle, and the equivalent oblique decision trees on the RHS.

linear regions. Since any function taking the piecewise constant Jacobian as input will remain itself piecewise constant, we can construct a variety of locally constant networks by composition.

However, in order to simplify the derivation, we first make a trivial observation that the activation pattern in each locally linear region is also locally invariant. More broadly, any invariant quantity in each locally linear region can be utilized so as to build locally constant networks. We thus define the locally constant networks as any composite functions that leverage the local invariance of piecewise linear networks. For the theoretical analyses, we consider the below architecture.

Canonical architecture. Let $\tilde{M} \triangleq \sum_{i=1}^{L} M_i$, $\tilde{\boldsymbol{o}} : \mathbb{R}^D \to \{0,1\}^{\tilde{M}}$ denote the concatenation of $(\boldsymbol{o}^1, \boldsymbol{o}^2, \ldots, \boldsymbol{o}^L)$, and $g : \{0,1\}^{\tilde{M}} \to \mathbb{R}^Y$ be a table. Then we define the canonical locally constant networks as the composite function $g(\tilde{\boldsymbol{o}})$.

Before elucidating on the representational equivalence to oblique decision trees, we first show some toy examples of the canonical locally constant networks and their equivalent mappings in Fig. 4.3.1, which illustrates their constructions when there is only 1 neuron per layer (i.e., $\boldsymbol{z}^i = \boldsymbol{z}^i_1$, and similarly for $\boldsymbol{o}^i$ and $\boldsymbol{a}^i$) and thus $\tilde{M} = L$.

When $L = 1$, $\boldsymbol{o}^1 = 1 \Leftrightarrow \boldsymbol{x}_1 - \boldsymbol{x}_2 + 1 \geq 0$, thus the locally constant network is equivalent to a linear model shown in the middle, which can also be represented as an oblique decision tree with depth = 1. When $L > 1$, the activations in the previous layers control different linear behaviors of a neuron with respect to the input, thus realizing a hierarchical structure as an oblique decision tree. For example, for $L = 2$, $\boldsymbol{o}^1 = 0 \Leftrightarrow \boldsymbol{z}^1 < 0 \Rightarrow \boldsymbol{z}^2 = -4\boldsymbol{x}_1 + \boldsymbol{x}_2 + 4$ and $\boldsymbol{o}^1 = 1 \Leftrightarrow \boldsymbol{z}^1 \geq 0 \Rightarrow \boldsymbol{z}^2 = -3\boldsymbol{x}_2 + 8$; hence, it can also be interpreted as the decision tree on the RHS, where the *concrete realization* of $\boldsymbol{z}^2$ depends on the previous decision variable $\boldsymbol{z}^1 \geq 0$. Afterwards, we can map either the activation patterns on the LHS or the decision patterns on the RHS to an output value, which leads to the mapping in the middle.

4.3.2 Representational Equivalence

In this section, we prove how oblique decision trees and locally constant networks can be represented by each other. We first make an observation that any unbalanced oblique decision tree can be rewritten to be balanced by adding dummy decision nodes (e.g, $\mathbf{0}^\top \boldsymbol{x} \geq -1$). Thus we define the class of oblique decision trees with depth T as:

Definition 4.1. *The class of oblique decision trees with depth $T \in \mathbb{Z}_{>0}$, denoted as $\mathcal{F}_T^{\text{Tree}}$, contains any functions that can be procedurally defined for $\boldsymbol{x} \in \mathbb{R}^D$:*

1. $\boldsymbol{r}_1 \triangleq \mathbb{I}[\boldsymbol{\omega}_\varnothing^\top \boldsymbol{x} + \beta_\varnothing \geq 0]$, *where $\boldsymbol{\omega}_\varnothing \in \mathbb{R}^D$ and $\beta_\varnothing \in \mathbb{R}$ denote the weight and bias of the root decision node.*
2. *For $i \in (2, 3, \ldots, T)$, $\boldsymbol{r}_i \triangleq \mathbb{I}[\boldsymbol{\omega}_{\boldsymbol{r}_{1:i-1}}^\top \boldsymbol{x} + \beta_{\boldsymbol{r}_{1:i-1}} \geq 0]$, where $\boldsymbol{\omega}_{\boldsymbol{r}_{1:i-1}} \in \mathbb{R}^D$ and $\beta_{\boldsymbol{r}_{1:i-1}} \in \mathbb{R}$ denote the weight and bias for the decision node after the decision pattern $\boldsymbol{r}_{1:i-1}$.*
3. *$v : \{0,1\}^T \to \mathbb{R}^Y$ outputs the leaf value $v(\boldsymbol{r}_{1:T})$ associated with the decision pattern $\boldsymbol{r}_{1:T}$.*

The class of locally constant networks with $\tilde{M} \in \mathbb{Z}_{>0}$ hidden neurons, defined by the *canonical architecture*, is denoted as $\mathcal{F}_{\tilde{M}}^{\text{Net}}$.

We will first prove that locally constant networks can represent an oblique decision tree with the same number of decision nodes (i.e., ReLU units). Since a typical oblique decision tree can produce an arbitrary weight in each decision node (cf. the structurally dependent weights in the oblique decision trees in Fig. 4.3.1), the idea is to utilize a network with only 1 hidden layer such that the neurons do not constrain one another. Concretely,

Theorem 4.2. $\mathcal{F}^{\text{Net}}_{2^T-1} \supseteq \mathcal{F}^{\text{Tree}}_{T}$

Proof. For any oblique decision tree with depth T, it contains 2^T-1 weights and biases. We thus construct a locally constant network with $L=1$ and $M_1 = 2^T-1$ such that each pair of $(\boldsymbol{\omega}, \beta)$ in the oblique decision tree is equal to some $\boldsymbol{W}^1_{k,:}$ and $\boldsymbol{b}^1_k$ in the constructed locally constant network.

For each leaf node in the decision tree, it is associated with an output value $\boldsymbol{y} \in \mathbb{R}^Y$ and T decisions; the decisions can be written as $\boldsymbol{W}^1_{\texttt{idx}[j],:}\boldsymbol{x} + \boldsymbol{b}^1_{\texttt{idx}[j]} \geq 0$ for $j \in \{1,2,\dots,T'\}$ and $\boldsymbol{W}^1_{\texttt{idx}[j],:}\boldsymbol{x} + \boldsymbol{b}^1_{\texttt{idx}[j]} < 0$ for $j \in \{T'+1, T'+2, \dots, T\}$ for some index function $\texttt{idx} : [T] \to [2^T-1]$ and some $T' \in \{0,1,\dots,T\}$. We can set the table $g(\cdot)$ of the locally constant network as

$$\boldsymbol{y}, \text{ if} \begin{cases} \boldsymbol{o}^1_{\texttt{idx}[j]} = 1 (\Leftrightarrow \boldsymbol{W}^1_{\texttt{idx}[j],:}\boldsymbol{x} + \boldsymbol{b}^1_{\texttt{idx}[j]} \geq 0), \text{for } j \in \{1,2,\dots,T'\}, \text{ and} \\ \boldsymbol{o}^1_{\texttt{idx}[j]} = 0 (\Leftrightarrow \boldsymbol{W}^1_{\texttt{idx}[j],:}\boldsymbol{x} + \boldsymbol{b}^1_{\texttt{idx}[j]} < 0), \text{for } j \in \{T'+1, T'+2, \dots, T\}. \end{cases}$$

As a result, the constructed locally constant network yields the same output as the given oblique decision tree for all the inputs that are routed to each leaf node, which concludes the proof. □

Since the main idea of the proof is to map the decision nodes to the same number of neurons, we can easily generalize the result to a more fine-grained statement. Below a non-dummy decision node means that both decision outcomes $\{0,1\}$ are feasible and uniqueness is with respect to the linear classification (i.e., invariant to scaling).

Corollary 4.3. $\mathcal{F}^{\text{Net}}_{T} \supseteq \{f \in \mathcal{F}^{\text{Net}}_{T} : L = 1\} \supseteq \{f \in \mathcal{F}^{\text{Tree}}_{T}$: f contains at most T non-dummy and unique decision nodes$\}$.

The proof is omitted due to the extreme similarity to the proof of Theorem 4.2. Then we prove that locally constant networks with T neurons can represent an oblique decision tree with an exponential number of decision nodes, which simply follows the construction of the toy examples in Fig. 4.3.1.

Theorem 4.4. $\mathcal{F}_T^{\text{Tree}} \supseteq \mathcal{F}_T^{\text{Net}}$

Proof. For any locally constant network with T neurons, it can be re-written to have 1 neuron per layer, by expanding any layer with $M_i > 1$ neurons to be M_i different layers such that they do not have effective intra-connections. Below the notation refers to the converted locally constant network with 1 neuron per layer. We define the following oblique decision tree with depth T for $\boldsymbol{x} \in \mathbb{R}^D$:

1. $\boldsymbol{r}_1 \triangleq \boldsymbol{o}_1^1 = \mathbb{I}[\boldsymbol{\omega}_\varnothing^\top \boldsymbol{x} + \beta_\varnothing \geq 0]$ with $\boldsymbol{\omega}_\varnothing = \boldsymbol{W}_{1,:}^1$ and $\beta_\varnothing = \boldsymbol{b}_1^1$.
2. For $i \in (2, 3, \ldots, T), \boldsymbol{r}_i \triangleq \mathbb{I}[\boldsymbol{\omega}_{\boldsymbol{r}_{1:i-1}}^\top \boldsymbol{x} + \beta_{\boldsymbol{r}_{1:i-1}} \geq 0]$, where $\boldsymbol{\omega}_{\boldsymbol{r}_{1:i-1}} = \nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i$ and $\beta_{\boldsymbol{r}_{1:i-1}} = \boldsymbol{z}_1^i - (\nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i)^\top \boldsymbol{x}$. Note that $\boldsymbol{r}_i = \mathbb{I}[\boldsymbol{z}_1^i \geq 0] = \boldsymbol{o}_1^i$.
3. $v = g$.

Note that, in order to be a valid decision tree, $\boldsymbol{\omega}_{1:\boldsymbol{r}_{i-1}}$ and $\beta_{1:\boldsymbol{r}_{i-1}}$ have to be unique for all $\boldsymbol{x}$ that yield the same decision pattern $\boldsymbol{r}_{1:i-1}$. To see this, for $i \in (2, 3, \ldots, T)$, as $\boldsymbol{r}_{1:i-1} = (\boldsymbol{o}_1^1, \ldots, \boldsymbol{o}_1^{i-1})$, we know each $\boldsymbol{z}_1^i$ is a fixed affine function given an activation pattern for the preceding neurons, so $\nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i$ and $\boldsymbol{z}_1^i - \boldsymbol{x}^\top \nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i$ are fixed quantities given a decision pattern $\boldsymbol{r}_{1:i-1}$.

Since $\boldsymbol{r}_{1:M} = \tilde{\boldsymbol{o}}$ and $v = g$, we conclude that they yield the same mapping. □

Despite the simplicity of the proof, it has some practical implications:

Remark 4.5. *The proof of Theorem 4.4 implies that we can train a locally constant network with $\tilde{M}$ neurons, and convert it to an oblique decision tree with depth $\tilde{M}$.*

Remark 4.6. *The proof of Theorem 4.4 establishes that, given a fixed number of neurons, it suffices (representationally) to only consider the locally constant networks with one neuron per layer (i.e., $M_i = 1, \forall i \in [L]$).*

Remark 4.6 is important for learning small locally constant networks (which can be converted to shallow decision trees for interpretability), since representation capacity is critical for low capacity models. As a result, *in the remainder of the chapter, we will only consider the setting with $M_i = 1, \forall i \in [L]$ (and thus $\tilde{M} = L$).*

To conclude this section, we make a simple observation that by taking the union of across $T \in \mathbb{Z}_{>0}$ in Theorem 4.2 and 4.4, we can derive the class-level equivalence between oblique decision trees and locally constant networks *without depth constraints*:

Corollary 4.7. $\cup_{T=1}^{\infty} \mathcal{F}_T^{\text{Net}} \equiv \cup_{T=1}^{\infty} \mathcal{F}_T^{\text{Tree}}$.

4.3.3 Structurally Shared Parameterization

As briefly highlighted in the previous section, if we restrict the number of neurons in the locally constant networks with some $\tilde{M} > 1$, it cannot reproduce all the decision trees with depth $T = \tilde{M}$. The result can be intuitively understood by the following reason: we are effectively using $\tilde{M}$ pairs of (weight, bias) in the locally constant network to implicitly realize $2^{\tilde{M}} - 1$ pairs of (weight, bias) in the corresponding oblique decision tree. Such exponential reduction on the effective parameters in the representation of oblique decision trees yields "dimension reduction" of the model capacity. This section aims to reveal the implied shared parameterization embedded in the oblique decision trees derived from locally constant networks.

In this section, the oblique decision trees and the associated parameters refer to *the decision trees obtained via the proof of Theorem 4.4*. We start the analysis by a decomposition of $\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}}$ among the preceding weights $\boldsymbol{\omega}_{\varnothing}, \boldsymbol{\omega}_{\boldsymbol{r}_{1:1}}, \ldots, \boldsymbol{\omega}_{\boldsymbol{r}_{1:i-1}}$. To simplify notation, we denote $\boldsymbol{\omega}_{\boldsymbol{r}_{1:0}} \triangleq \boldsymbol{\omega}_{\varnothing}$. Since $\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}} = \nabla_{\boldsymbol{x}} \boldsymbol{z}_1^{i+1}$ and $\boldsymbol{z}_1^{i+1}$ is an affine transformation of the vector $\texttt{CONCATENATE}[\boldsymbol{a}_0, \boldsymbol{a}_1^1, \ldots, \boldsymbol{a}_1^i]$,

$$\begin{aligned}
\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}} = \nabla_{\boldsymbol{x}} \boldsymbol{z}_1^{i+1} &= \boldsymbol{W}_{1,1:D}^{i+1} + \sum_{k=1}^{i} \boldsymbol{W}_{1,D+k}^{i+1} \times \frac{\partial \boldsymbol{a}_1^k}{\partial \boldsymbol{z}_1^k} \times \nabla_{\boldsymbol{x}} \boldsymbol{z}_1^k \\
&= \boldsymbol{W}_{1,1:D}^{i+1} + \sum_{k=1}^{i} \boldsymbol{W}_{1,D+k}^{i+1} \times \boldsymbol{r}_k \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:k-1}},
\end{aligned}$$

where we simply rewrite the derivatives in terms of tree parameters. Since $\boldsymbol{W}_{1,1:D}^{i+1}$ is fixed for all the $\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}}$, the above decomposition implies that, in the induced tree, all the weights $\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}}$ in *the same depth* i are restricted to be a linear combination of the fixed basis $\boldsymbol{W}_{1,1:D}^{i+1}$ and the corresponding preceding weights $\boldsymbol{\omega}_{\boldsymbol{r}_{1:0}}, \ldots, \boldsymbol{\omega}_{\boldsymbol{r}_{1:i-1}}$. We can extend this analysis to compare weights in same layer, beginning from comparing weights whose ℓ_0 distance in decision pattern is 1. To help interpret the statement, note that $\boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}}$ is the weight that leads to the decision $\boldsymbol{r}_j$ (or $\boldsymbol{r}'_j$; see below).

Lemma 4.8. *For an oblique decision tree with depth $T > 1$, $\forall i \in [T-1]$ and any $\boldsymbol{r}_{1:i}$, $\boldsymbol{r}'_{1:i}$ such that $\boldsymbol{r}_k = \boldsymbol{r}'_k$ for all $k \in [i]$ except that $\boldsymbol{r}_j \neq \boldsymbol{r}'_j$ for some $j \in [i]$, we have*

$$\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}} - \boldsymbol{\omega}_{\boldsymbol{r}'_{1:i}} = \alpha \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}}, \text{ for some } \alpha \in \mathbb{R}.$$

The proof involves some algebraic manipulation, and is deferred to Appendix 4.A.1. Lemma 4.8 characterizes an interesting structural constraint embedded in the oblique decision trees realized by locally constant networks, where the structural discrepancy $\boldsymbol{r}_j$ in decision patterns ($\boldsymbol{r}_{1:i}$ versus $\boldsymbol{r}'_{1:i}$) is reflected on the discrepancy of the corresponding weights (up to a scaling factor α). The analysis can be generalized for all the weights in the same layer, but the message is similar.

Proposition 4.9. *For the oblique decision tree with depth $T > 1$, $\forall i \in [T-1]$ and any $\boldsymbol{r}_{1:i}$, $\boldsymbol{r}'_{1:i}$ such that $\boldsymbol{r}_k = \boldsymbol{r}'_k$ for all $k \in [i]$ except for $n \in [i]$ coordinates $j_1, \ldots, j_n \in [i]$, we have*

$$\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}} - \boldsymbol{\omega}_{\boldsymbol{r}'_{1:i}} = \sum_{k=1}^{n} \alpha_k \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j_k-1}}, \text{ for some } \alpha_k \in \mathbb{R}, \forall k \in [n]. \tag{4.2}$$

The statement can be proved by applying Lemma 4.8 multiple times.

Discussion. Here we summarize this section and provide some discussion. Locally constant networks implicitly represent oblique decision trees with the same depth and structurally shared parameterization. In the implied oblique decision trees, the weight of each decision node is a linear combination of a shared weight across the whole layer and all the preceding weights. The analysis explains how locally constant networks use only $\tilde{M}$ weights to model a decision tree with $2^{\tilde{M}} - 1$ decision nodes; it yields a strong regularization effect to avoid overfitting, and helps computation by exponentially reducing the memory consumption on the weights.

4.3.4 Standard Locally Constant Networks and Extensions

The simple structure of the *canonical* locally constant networks is beneficial for theoretical analysis, but the structure is not practical for learning since the *discrete* activation pattern does not exhibit gradients for learning the networks. Indeed, $\nabla_{\tilde{\boldsymbol{o}}} g(\tilde{\boldsymbol{o}})$ is undefined, which implies that $\nabla_{\boldsymbol{W}^i} g(\tilde{\boldsymbol{o}})$ is also undefined. Here we present another architecture that is equivalent to the *canonical* architecture, but exhibits sub-gradients with respect to model parameters and is flexible for model extension.

Standard architecture. Let $\tilde{\boldsymbol{a}} : \mathbb{R}^D \to \mathbb{R}^{\tilde{M}}$ denote the concatenation of $(\boldsymbol{a}^1, \boldsymbol{a}^2, \dots, \boldsymbol{a}^L)$, $J_{\boldsymbol{x}}\tilde{\boldsymbol{a}} \in \mathbb{R}^{\tilde{M} \times D}$ denote the Jacobian of $\tilde{\boldsymbol{a}}$ with respect to the input, and $\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$ denote its vectorization. Then we define the standard locally constant networks as $g_\phi(\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}})$, where $g_\phi : \mathbb{R}^{(\tilde{M} \times D)} \to \mathbb{R}^Y$ is a fully-connected network.

We abbreviate the standard locally constant networks as **Lcn**. Note that each $\boldsymbol{a}_1^i$ is piecewise linear and thus the Jacobian $J_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$ is piecewise constant. We replace $\tilde{\boldsymbol{o}}$ with $J_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$ as the invariant representation for each locally linear region[1], and replace the table g with a differentiable function g_ϕ that takes as input real vectors. The gradients of Lcn with respect to parameters is thus established through the derivatives of g_ϕ and the mixed partial derivatives of the neurons (derivatives of $\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$).

Fortunately, all the previous analyses also apply to the standard architecture, due to a fine-grained equivalence between the two architectures. First, while it is easy to see that the Jacobian $J_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$ can be written as a function of the activation pattern $\tilde{\boldsymbol{o}}$, we prove that they are actually equally powerful.

Theorem 4.10. *There exists a bijection between the Jacobian $J_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$ and the activation pattern $\tilde{\boldsymbol{o}}$.*

The core idea of the proof is to show the following by induction. Since $\boldsymbol{o}_1^i = 0 \implies \nabla_{\boldsymbol{x}}\boldsymbol{a}^i = \boldsymbol{0}$ (the zero vector), we only have to show when $\boldsymbol{o}_1^i = 1$, the corresponding gradient $\nabla_{\boldsymbol{x}}\boldsymbol{a}^i$ cannot degenerate to $\boldsymbol{0}$ unless $\boldsymbol{o}_1^i$ is a constant, thus forming a bijection. This is easy see for $\boldsymbol{a}^1 : \nabla_{\boldsymbol{x}}\boldsymbol{a}^1 \neq \boldsymbol{0} \iff \boldsymbol{o}_1^1 = 1$ unless $\boldsymbol{W}^1 = 0$. The complete proof

[1]In practice, we also include each bias $\boldsymbol{a}_1^i - (\nabla_{\boldsymbol{x}}\boldsymbol{a}_1^i)^\top \boldsymbol{x}$, which is omitted here to simplify exposition.

is available in Appendix 4.A.2. Combining Theorem 4.10 with the universality of fully-connected networks [62] and tables, we have the corollary:

Corollary 4.11. *Given any fixed f_θ, any canonical architecture $g(\tilde{\boldsymbol{o}})$ can be equivalently represented by a standard architecture $g_\phi(\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}})$, and vice versa.*

The proof is available in Appendix 4.A.3. Since f_θ and g control the decision nodes and leaf nodes in the associated oblique decision tree, respectively (see the proof of Theorem 4.4), Corollary 4.11 essentially states that both architectures are equally competent for assigning leaf nodes. All the analyses in §4.3.2 and §4.3.3 are thus immediately inherited.

Discussion. The standard architecture yields a new property that is only partially exhibited in the canonical architecture. For all the decision and leaf nodes which no training data is routed to, there is no way to obtain learning signals in classic oblique decision trees. However, due to shared parameterization (see §4.3.3), locally constant networks can "learn" all the decision nodes in the implied oblique decision trees (if there is a way to optimize the networks), and the standard architecture can even "learn" all the leaf nodes due to the parameterized output function g_ϕ.

Extensions. The construction of (standard) locally constant networks enables several natural extensions due to the flexibility of the neural architecture and the interpretation of decision trees. The original locally linear networks (**Lln**) f_θ, which outputs a linear function instead of a constant function for each region, can be regarded as one extension. Here we discuss two examples.

- Approximately locally constant networks (**Alcn**): we can change the activation function while keeping the model architecture of Lcn. For example, we can replace ReLU $\max(0, x)$ with softplus $\log(1 + \exp(x))$, which will lead to an approximately locally constant network, as the softplus function has an approximately locally constant derivative for inputs with large absolute values. Note that the canonical architecture (tabular g) is not compatible with such extension.

- Ensemble locally constant networks (**ELCN**): since each LCN can only output 2^M different values, it is limited for complex tasks like regression (akin to decision trees). We can instead use an additive ensemble of LCN or ALCN to increase the capacity. We use $g_\phi^{[e]}(\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}}^{[e]})$ to denote a base model in the ensemble, and denote the ensemble with E models as $\sum_{e=1}^{E} g_\phi^{[e]}(\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}}^{[e]})$.

4.3.5 Computation and Learning

In this section, we discuss computation and learning algorithms for the proposed models. In the following complexity analyses, we assume g_ϕ to be a linear model.

Space complexity. The space complexity of LCN is $\Theta(LD + L^2)$ for representing decision nodes and $\Theta(LDY)$ for representing leaf nodes. In contrast, the space complexity for a classic oblique decision tree with the same depth is $\Theta((2^L - 1)D)$ for decision nodes and $\Theta(2^L Y)$ for leaf nodes. Hence, our representation improves the space complexity over classic oblique decision trees exponentially.

Computation and time complexity. LCN and ALCN are built on the gradients of all the neurons $\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}} = \texttt{CONCATENATE}[\nabla_{\boldsymbol{x}}\boldsymbol{a}_1^1, \dots, \nabla_{\boldsymbol{x}}\boldsymbol{a}_1^L]$, which can be computationally challenging to obtain. Automatic differentiation methods (e.g., back-propagation) only compute the gradient of a scalar output. Instead, here we propose an efficient dynamic programming procedure which only requires a forward pass:

1. $\nabla_{\boldsymbol{x}}\boldsymbol{a}_1^1 = \boldsymbol{o}_1^1 \times \boldsymbol{W}^1$.
2. $\forall i \in \{2, \dots, L\}, \nabla_{\boldsymbol{x}}\boldsymbol{a}_1^i = \boldsymbol{o}_1^i \times (\boldsymbol{W}_{1,1:D}^i + \sum_{k=1}^{i-1} \boldsymbol{W}_{1,D+k}^i \nabla_{\boldsymbol{x}}\boldsymbol{a}_1^k)$,

The complexity of the dynamic programming is $\Theta(L^2)$ due to the inner-summation inside each iteration. Straightforward back-propagation re-computes the partial solutions $\nabla_{\boldsymbol{x}}\boldsymbol{a}_1^k$ for each $\nabla_{\boldsymbol{x}}\boldsymbol{a}_1^i$, so the complexity is $\Theta(L^3)$. We can parallelize the inner-summation on a GPU, and the complexity of the dynamic programming and straightforward back-propagation will become $\Theta(L)$ and $\Theta(L^2)$, respectively. Note that the complexity of a forward pass of a typical network is also $\Theta(L)$ on a GPU. The time complexity of learning LCN by (stochastic) gradient descent is thus $\Theta(L\tau)$, where τ denotes the number of iterations. In contrast, the computation of existing oblique decision tree training algorithms is typically data-dependent and thus the complexity is hard to characterize.

Training LCN and ALCN. Even though LCN is sub-differentiable, whenever $\boldsymbol{o}_1^i = 0$, the network does not exhibit useful gradient information for learning each locally constant representation $\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i$ (note that $\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}} = \texttt{CONCATENATE}[\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^1, \ldots, \nabla_{\boldsymbol{x}} \boldsymbol{a}_1^L]$), since $\boldsymbol{o}_1^i = 0$ implies $\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i = \nabla_{\boldsymbol{x}} (\boldsymbol{o}_1^i \boldsymbol{z}_1^i) = \nabla_{\boldsymbol{x}} 0 = \boldsymbol{0}$, yielding no useful gradients. To alleviate the problem, we propose to leverage softplus as an infinitely differentiable approximation of ReLU to obtain meaningful learning signals for $\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i$. Concretely, we conduct the annealing during training:

$$\boldsymbol{a}_1^i = \lambda_t \max(0, \boldsymbol{z}_1^i) + (1 - \lambda_t) \log(1 + \exp(\boldsymbol{z}_1^i)), \forall i \in [M], \lambda_t \in [0, 1], \tag{4.3}$$

where λ_t is an iteration-dependent annealing parameter. Both LCN and ALCN can be constructed as a special case of Eq. (4.3). We train LCN with λ_t equal to the ratio between the current epoch and the total epochs, and ALCN with $\lambda_t = 0$. Both models are optimized via stochastic gradient descent.

We also include DropConnect [149] to the weight matrices $\boldsymbol{W}^i \leftarrow \text{drop}(\boldsymbol{W}^i)$ during training. Despite the simple structure of DropConnect in the locally constant networks, it entails a structural dropout on the weights in the corresponding oblique decision trees (see §4.3.3), which is challenging to reproduce in typical oblique decision trees. In addition, it also encourages the exploration of parameter space, which is easy to see for the raw LCN: the randomization enables the exploration that flips $\boldsymbol{o}_1^i = 0$ to $\boldsymbol{o}_1^i = 1$ to establish effective learning signal. Note that the standard DropOut [136] is not ideal for the low capacity models that we consider here.

Training ELCN. Since each ensemble component is sub-differentiable, we can directly learn the whole ensemble through gradient descent. However, the approach is not scalable due to memory constraints in practice. Instead, we propose to train the ensemble in a boosting fashion:

1. We first train an initial locally constant network $g_\phi^{[1]}(\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}}^{[1]})$.
2. For each iteration $e' \in \{2, 3, \ldots, E\}$, we incrementally optimize $\sum_{e=1}^{e'} g_\phi^{[e]}(\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}}^{[e]})$.

Table 4.4.1: Dataset statistics

Dataset	Bace	HIV	SIDER	Tox21	PDBbind
Task	(Multi-label) binary classification				Regression
Number of labels	1	1	27	12	1
Number of data	1,513	41,127	1,427	7,831	11,908

Note that, in the second step, only the latest model is optimized, and thus we can simply store the predictions of the preceding models without loading them into the memory. Each partial ensemble can be directly learned through gradient descent, without resorting to complex meta-algorithms such as adaptive boosting [45] or gradient boosting [46].

4.4 Experiments

Here we evaluate the efficacy of our models (Lcn, Alcn, and Elcn) using the chemical property prediction datasets from MoleculeNet [158], where random forest performs competitively. We include 4 (multi-label) binary classification datasets and 1 regression dataset. The statistics are available in Table 4.4.1. We follow the literature to construct the feature [158]. Concretely, we use the standard Morgan fingerprint (2,048 binary indicators of chemical substructures) [122] for the classification datasets, with additional 'grid features' (fingerprints of pairs between ligand and protein, see [158]) for the regression dataset. Each dataset is splitted into (train, validation, test) sets under the criterion specified in MoleculeNet.

We compare Lcn and its extensions (Lln, Alcn, and Elcn) with the following baselines:

- (Oblique) decision trees: Cart [18], Hhcart (oblique decision trees induced greedily on linear projections) [153], and Tao (oblique decision trees trained via alternating optimization) [23].
- Tree ensembles: Rf (random forest) [19] and Gbdt (gradient boosting decision trees) [46].

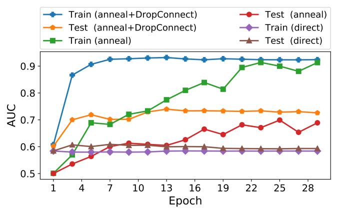
Table 4.4.2: Main results. The 1st section refers to (oblique) decision tree methods, the 2nd section refers to single model extensions of LCN, the 3rd section refers to ensemble methods, and the last section is GCN. The results of GCN are copied from [158], where the results in SIDER and Tox21 are not directly comparable due to lack of standard splittings. The best result in each section is in bold letters.

Dataset	Bace (AUC)	HIV (AUC)	SIDER (AUC)	Tox21 (AUC)	PDBbind (RMSE)
CART	0.652 ± 0.024	0.544 ± 0.009	0.570 ± 0.010	0.651 ± 0.005	1.573 ± 0.000
HHCART	0.545 ± 0.016	0.636 ± 0.000	0.570 ± 0.009	0.638 ± 0.007	1.530 ± 0.000
TAO	0.734 ± 0.000	0.627 ± 0.000	0.577 ± 0.004	0.676 ± 0.003	Not applicable
LCN	**0.839 ± 0.013**	**0.728 ± 0.013**	**0.624 ± 0.044**	**0.781 ± 0.017**	**1.508 ± 0.017**
LLN	0.818 ± 0.007	0.737 ± 0.009	**0.677 ± 0.014**	0.813 ± 0.009	1.627 ± 0.008
ALCN	**0.854 ± 0.007**	**0.738 ± 0.009**	0.653 ± 0.044	**0.814 ± 0.009**	**1.369 ± 0.007**
RF	0.869 ± 0.003	**0.796 ± 0.007**	0.685 ± 0.011	**0.839 ± 0.007**	1.256 ± 0.002
GBDT	0.859 ± 0.005	0.748 ± 0.001	0.668 ± 0.014	0.812 ± 0.011	1.247 ± 0.002
ELCN	**0.874 ± 0.005**	0.757 ± 0.011	**0.685 ± 0.010**	0.822 ± 0.006	**1.219 ± 0.007**
GCN	0.783 ± 0.014	0.763 ± 0.016	*0.638 ± 0.012	*0.829 ± 0.006	1.44 ± 0.12

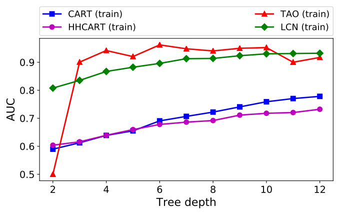
- Graph networks: GCN (graph convolutional networks on molecules) [37].

For decision trees, LCN, LLN, and ALCN, we tune the tree depth in $\{2, 3, \ldots, 12\}$. For LCN, LLN, and ALCN, we also tune the DropConnect probability in $\{0, 0.25, 0.5, 0.75\}$. Since regression tasks require precise estimations of the prediction values while classification tasks do not, we tune the number of hidden layers of g_ϕ in $\{0, 1, 2, 3, 4\}$ (each with 256 neurons) for the regression task, and simply use a linear model g_ϕ for the classification tasks. For ELCN, we use ALCN as the base model, tune the ensemble size $E \in \{2^0, 2^1, \ldots, 2^6\}$ for the classification tasks, and $E \in \{2^0, 2^1, \ldots, 2^9\}$ for the regression task. To train our models, we use the cross entropy loss for the classification tasks, and mean squared error for the regression task. Other minor details are available in Appendix 4.B.

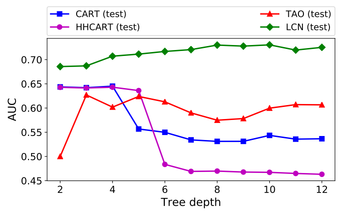
We follow the chemistry literature [158] to measure the performance by AUC for classification, and root-mean-squared error (RMSE) for regression. For each dataset, we train a model for each label, compute the mean and standard deviation of the performance across 10 different random seeds, and report their average across all the labels within the dataset. The results are in Table 4.4.2.



(a) Learning curve of LCN



(b) Training performance



(c) Testing performance

Figure 4.4.1: Empirical analysis for oblique decision trees on the HIV dataset, including (a) an ablation study for LCN and (b-c) comparisons of learning algorithms.

Among the (oblique) decision tree training algorithms, our LCN achieves the state-of-the-art performance. The continuous extension (ALCN) always improves the empirical performance of LCN, which is expected since LCN is limited for the number of possible outputs (leaf nodes). Among the ensemble methods, the proposed ELCN always outperforms the classic counterpart, GBDT, and sometimes outperforms RF. Overall, LCN is the state-of-the-art method for learning oblique decision trees, and ELCN performs competitively against other alternatives for training tree ensembles.

Empirical analysis. Here we analyze the proposed LCN in terms of the optimization and generalization performance in the large HIV dataset. We conduct an ablation study on the proposed method for training LCN in Figure 4.4.1a. Direct training (without annealing) does not suffice to learn LCN, while the proposed annealing succeed in optimization; even better optimization and generalization performance can be achieved by introducing DropConnect, which corroborates our hypothesis on the exploration effect during training in §4.3.5 and its well-known regularization effect. Compared to other methods (Fig. 4.4.1b), only TAO has a comparable training performance. In terms of generalization (Fig. 4.4.1c), all of the competitors do not perform well and overfit fairly quickly. In stark contrast, LCN outperforms the competitors by a large margin and gets even more accurate as the depth increases. This is expected due to the strong regularization of LCN that uses a linear number of effective weights to construct an exponential number of decision nodes, as discussed in §4.3.3. Some additional analysis and the visualization of the tree converted from LCN are included in Appendix 4.C.

4.5 Discussion and Conclusion

We create a novel neural architecture by casting the derivatives of deep networks as the representation, which realizes a new class of neural models that is equivalent to oblique decision trees. The induced oblique decision trees embed rich structures and are compatible with deep learning methods. This work can be used to interpret methods that utilize derivatives of a network, such as training a generator through the gradient of a discriminator [49]. The work opens up many avenues for future work, from building representations from the derivatives of neural models to the incorporation of more structures, such as the inner randomization of random forest.

4.A Proofs

4.A.1 Proof of Lemma 4.8

Proof. We fix j and do induction on i. Without loss of generality, we assume $1 = \boldsymbol{r}_j \neq \boldsymbol{r}'_j = 0$.

If $i = j$, since $\boldsymbol{r}'_j = 0$, we have

$$\begin{cases} \boldsymbol{\omega}_{\boldsymbol{r}_{1:i}} = \boldsymbol{W}^{i+1}_{1,1:D} + \sum_{k=1}^{i} \boldsymbol{W}^{i+1}_{1,D+k} \times \boldsymbol{r}_k \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:k-1}}, \\ \boldsymbol{\omega}_{\boldsymbol{r}'_{1:1}} = \boldsymbol{W}^{i+1}_{1,1:D} + \sum_{k=1}^{i-1} \boldsymbol{W}^{i+1}_{1,D+k} \times \boldsymbol{r}_k \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:k-1}}. \end{cases}$$

Hence, we have $\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}} - \boldsymbol{\omega}_{\boldsymbol{r}'_{1:1}} = (\boldsymbol{W}^{i+1}_{1,D+i} \times \boldsymbol{r}_i) \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:i-1}} = \alpha \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}}$.

We assume the statement holds for *up to* some integer $i \geq j$:

$$\boldsymbol{\omega}_{\boldsymbol{r}_{1:i}} - \boldsymbol{\omega}_{\boldsymbol{r}'_{1:i}} = \alpha \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}}, \text{ for some } \alpha \in \mathbb{R}.$$

For $i+1$, we have

$$\begin{aligned}
\boldsymbol{\omega}_{\boldsymbol{r}_{1:i+1}} =& \boldsymbol{W}^{i+2}_{1,1:D} + \sum_{k=1}^{i+1} \boldsymbol{W}^{i+2}_{1,D+k} \times \boldsymbol{r}_k \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:k-1}} \\
=& \boldsymbol{W}^{i+2}_{1,1:D} + \sum_{k=1}^{j-1} \boldsymbol{W}^{i+2}_{1,D+k} \times \boldsymbol{r}_k \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:k-1}} + \boldsymbol{W}^{i+2}_{1,D+j} \times \boldsymbol{r}_j \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}} \\
& + \sum_{k=j+1}^{i+1} \boldsymbol{W}^{i+2}_{1,D+k} \times \boldsymbol{r}_k \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:k-1}} \\
=& \boldsymbol{W}^{i+2}_{1,1:D} + \sum_{k=1}^{j-1} \boldsymbol{W}^{i+2}_{1,D+k} \times \boldsymbol{r}'_k \times \boldsymbol{\omega}_{\boldsymbol{r}'_{1:k-1}} + \boldsymbol{W}^{i+2}_{1,D+j} \times \boldsymbol{r}_j \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}} \\
& + \sum_{k=j+1}^{i+1} \boldsymbol{W}^{i+2}_{1,D+k} \times \boldsymbol{r}'_k \times (\boldsymbol{\omega}_{\boldsymbol{r}'_{1:k-1}} + \alpha_k \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}}), \text{for some } \alpha_k \in \mathbb{R} \\
=& \boldsymbol{W}^{i+2}_{1,1:D} + \sum_{k=1}^{i+1} \boldsymbol{W}^{i+2}_{1,D+k} \times \boldsymbol{r}'_k \times \boldsymbol{\omega}_{\boldsymbol{r}'_{1:k-1}} \\
& + (\boldsymbol{W}^{i+2}_{1,D+j} \times \boldsymbol{r}_j + \sum_{k=j+1}^{i+1} \boldsymbol{W}^{i+2}_{1,D+k} \times \boldsymbol{r}_k \times \alpha_k) \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}} \\
=& \boldsymbol{\omega}_{\boldsymbol{r}'_{1:i+1}} + \alpha' \times \boldsymbol{\omega}_{\boldsymbol{r}_{1:j-1}}, \text{for some } \alpha' \in \mathbb{R}
\end{aligned}$$

The proof follows by induction. □

4.A.2 Proof of Theorem 4.10

Proof. For any $\boldsymbol{x}$ mapping to the same activation pattern $\tilde{\boldsymbol{o}}$, the Jacobian $\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$ is constant, thus there is a function from the activation pattern to the Jacobian.

We prove the other direction by layer-wise induction (note that $\tilde{\boldsymbol{o}}$ is the concatenation of $(\boldsymbol{o}^1_1, \dots, \boldsymbol{o}^L_1)$ and $\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$ is the concatenation of $(\nabla_{\boldsymbol{x}}\boldsymbol{a}^1_1, \dots, \nabla_{\boldsymbol{x}}\boldsymbol{a}^L_1)$):

1. The induction hypothesis is that $(\boldsymbol{o}^1_1, \dots, \boldsymbol{o}^i_1)$ is a function of $(\nabla_{\boldsymbol{x}}\boldsymbol{a}^1_1, \dots, \nabla_{\boldsymbol{x}}\boldsymbol{a}^i_1)$.

2. $(i = 1)$ If $\boldsymbol{W}^1 = \boldsymbol{0}$ (zero vector), $\boldsymbol{z}^1_1$, $\boldsymbol{a}^1_1$, and $\boldsymbol{o}^1_1$ are constant (thus being a function of $\nabla_{\boldsymbol{x}}\boldsymbol{a}^1_1$). Otherwise, $\nabla_{\boldsymbol{x}}\boldsymbol{a}^1_1 = \boldsymbol{0} \Leftrightarrow \boldsymbol{o}^1_1 = 0$ and $\nabla_{\boldsymbol{x}}\boldsymbol{a}^1_1 = \boldsymbol{W}^1 \Leftrightarrow \boldsymbol{o}^1_1 = 1$, so $\boldsymbol{o}^1_1$ can be written as a function of $\nabla_{\boldsymbol{x}}\boldsymbol{a}^1_1$.

3. ($i > 1$) Assume that we are given $(\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^1, \ldots, \nabla_{\boldsymbol{x}} \boldsymbol{a}_1^{i-1})$ and the corresponding $(\boldsymbol{o}_1^1, \ldots, \boldsymbol{o}_1^{i-1})$.

   If either $\boldsymbol{o}_1^i = 0$ or $\boldsymbol{o}_1^i = 1$ is infeasible (but not both), by induction hypothesis, $(\boldsymbol{o}_1^1, \ldots, \boldsymbol{o}_1^i)$ can be written as a function of $(\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^1, \ldots, \nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i)$.

   If $\boldsymbol{o}_1^i = 1$ for some $\boldsymbol{x}'$ and $\boldsymbol{o}_1^i = 0$ for some $\boldsymbol{x}''$, we claim that $\boldsymbol{o}_1^i = 1 \Rightarrow \nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i \neq \mathbf{0}$:

   If $\boldsymbol{o}_1^i = 1$ and $\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i = \mathbf{0}$, we have $\boldsymbol{o}_1^i = 1 \Rightarrow \boldsymbol{a}_1^i = \boldsymbol{z}_1^i \geq 0$ and $\mathbf{0} = \nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i = \nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i$, which implies that the bias (of $\boldsymbol{z}^i$, given $(\boldsymbol{o}_1^1, \ldots, \boldsymbol{o}_1^{i-1})$) $\boldsymbol{z}_1^i - (\nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i)^\top \boldsymbol{x} \geq 0$. Note that both $\nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i$ and $\boldsymbol{z}_1^i - (\nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i)^\top \boldsymbol{x}$ are constant given $(\boldsymbol{o}_1^1, \ldots, \boldsymbol{o}_1^{i-1})$, regardless of $\boldsymbol{o}_1^i$. Hence, given $(\boldsymbol{o}_1^1, \ldots, \boldsymbol{o}_1^{i-1})$, we have $\boldsymbol{z}_1^i = \boldsymbol{z}_1^i - (\nabla_{\boldsymbol{x}} \boldsymbol{z}_1^i)^\top \boldsymbol{x} \geq 0$ and $\boldsymbol{o}_1^i = 0$ is infeasible ($\Rightarrow\!\Leftarrow$).

   Note that, given fixed $(\boldsymbol{o}_1^1, \ldots, \boldsymbol{o}_1^{i-1})$, $\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i \neq \mathbf{0}$ has a unique value in $\mathbb{R}^D$. Combining the result $\boldsymbol{o}_1^i = 1 \Rightarrow \nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i \neq \mathbf{0}$ with $\boldsymbol{o}_1^i = 0 \Rightarrow \nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i = \mathbf{0}$, there is a bijection between $\boldsymbol{o}_1^i$ and $\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i$ in this case, which implies that $(\boldsymbol{o}_1^1, \ldots, \boldsymbol{o}_1^i)$ can be written as a function of $(\nabla_{\boldsymbol{x}} \boldsymbol{a}_1^1, \ldots, \nabla_{\boldsymbol{x}} \boldsymbol{a}_1^i)$.

□

4.A.3 Proof of Corollary 4.11

Proof. We first prove that we can represent any $g_\phi(\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}})$ as $g(\tilde{\boldsymbol{o}})$. By Theorem 4.10, we can write $\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}}$ as a function of the activation pattern $J'(\tilde{\boldsymbol{o}})$, thus we can set set the table as $g(\cdot) \triangleq g_\phi(J'(\cdot))$.

To prove the other direction, we can similarly write $\tilde{\boldsymbol{o}}$ as $\boldsymbol{o}'(\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}})$ by Theorem 4.10. Then it remains to show that there exists a feed-forward network g_ϕ such that $g_\phi(\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}}) = g(\tilde{\boldsymbol{o}}(\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}}))$ for at most $2^{\tilde{M}}$ distinct inputs $\vec{J}_{\boldsymbol{x}} \tilde{\boldsymbol{a}}$, which can be proved by the Theorem 2.5 of [62] or the Theorem 1 of [162].

□

4.B Implementation Details

Here we provide the full version of the implementation details.

For the baseline methods:

- Cart, Hhcart, and Tao: we tune the tree depth in $\{2, 3, \dots, 12\}$.
- Rf: we use the `scikit-learn` [114] implementation of random forest. We set the number of estimators as 500.
- Gbdt: we use the `scikit-learn` [114] implementation of gradient boosting trees. We tune the number of estimators in $\{2^3, 2^4, \dots, 2^{10}\}$.

For Lcn, Lln, and Alcn, we run the same training procedure. For all the datasets, we tune the depth in $\{2, 3, \dots, 12\}$ and the DropConnect probability in $\{0, 0.25, 0.5, 0.75\}$. The models are optimized with mini-batch stochastic gradient descent with batch size set to 64. For all the classification tasks, we set the learning rate as 0.1, which is annealed by a factor of 10 for every 10 epochs (30 epochs in total). For the regression task, we set the learning rate as 0.0001, which is annealed by a factor of 10 for every 30 epochs (60 epochs in total).

Both Lcn and Alcn have an extra fully-connected network g_ϕ, which transforms the derivatives $\vec{J}_{\boldsymbol{x}}\tilde{\boldsymbol{a}}$ to the final outputs. Since regression tasks require precise estimation of prediction values while classification tasks do not, we tune the number of hidden layers of g_ϕ in $\{0, 1, 2, 3, 4\}$ (each with 256 neurons) for the regression dataset, and simply use a linear g_ϕ for the classification datasets.

For Elcn, we fix the depth to 12 and tune the number of base models $E \in \{2^0, 2^1, \dots, 2^6\}$ for the classification tasks, and $E \in \{2^0, 2^1, \dots, 2^9\}$ for the regression task. We set the DropConnect probability as 0.75 to encourage strong regularization for the classification tasks, and as 0.25 to impose mild regularization for the regression task (because regression is hard to fit). We found stochastic gradient descent does not suffice to incrementally learn the Elcn, so we use the AMSGrad optimizer [120] instead. We set the batch size as 256 and train each partial ensemble for 30 epochs. The learning rate is 0.01 for the classification tasks, and 0.0001 for the regression task.

To train our models, we use the cross entropy loss for the classification tasks, and mean squared error for the regression task.

Table 4.C.1: Analysis for "unobserved decision patterns" of LCN in the Bace dataset.

Depth	8	9	10	11	12
# of possible patterns	256	512	1024	2048	4096
# of training patterns	72	58	85	103	86
# of testing patterns	32	31	48	49	40
# of testing patterns - training patterns	5	2	11	8	11
Ratio of testing points w/ unobserved patterns	0.040	0.013	0.072	0.059	0.079
Testing performance - observed patterns	0.8505	0.8184	0.8270	0.8429	0.8390
Testing performance - unobserved patterns	0.8596	0.9145	0.8303	0.7732	0.8894

4.C Empirical Analysis and Visualization

4.C.1 Supplementary Empirical Analysis

In this section, we investigate the learning of "unobserved branching / leaves" discussed in §4.3.4. The "unobserved branching / leaves" refer to the decision and leaf nodes of the oblique decision tree converted from LCN, such that there is no training data that are routed to the nodes. It is impossible for traditional (oblique) decision tree training algorithms to learn the values of such nodes (e.g., the output value of a leaf node in the traditional framework is based on the training data that are routed to the leaf node). However, the shared parameterization in our oblique decision tree provides a means to update such unobserved nodes during training (see the discussion in §4.3.4).

Since the above scenario in general happens more frequently in small datasets than in large datasets, we evaluate the scenario on the small Bace dataset (binary classification task). Here we empirically analyze a few things pertaining to the unobserved nodes:

- # of training patterns: the number of distinct end-to-end activation / decision patterns $\boldsymbol{r}_{1:L}$ encountered in the training data.
- # of testing patterns: the number of distinct end-to-end activation / decision patterns $\boldsymbol{r}_{1:L}$ encountered in the testing data.
- # of testing patterns - training patterns: the number of distinct end-to-end

activation / decision patterns $\boldsymbol{r}_{1:L}$ that is only encountered in the testing data but not in the training data.

- Ratio of testing points w/ unobserved patterns: the number of testing points that yield unobserved patterns divided by the total number of testing points.
- Testing performance - observed patterns: here we denote the number of testing data as N, the prediction and label of the i^{th} data point as $\hat{y}_i \in [0,1]$ and $y_i \in \{0,1\}$, respectively. We collect the subset of indices I of the testing data such that their activation / decision patterns $\boldsymbol{r}_{1:L}$ are observed in the training data, and then compute the performance of their predictions. Since the original performance is measured by AUC, here we generalize AUC to measure a subset of points I as:

$$\frac{\sum_{i\in I}\sum_{j=1}^{N}\Big(\mathbb{I}[y_i > y_j]\Big(\mathbb{I}[\hat{y}_i > \hat{y}_j] + 0.5\mathbb{I}[\hat{y}_i = \hat{y}_j]\Big) + \mathbb{I}[y_i < y_j]\Big(\mathbb{I}[\hat{y}_i < \hat{y}_j] + 0.5\mathbb{I}[\hat{y}_i = \hat{y}_j]\Big)\Big)}{\sum_{i\in I}\sum_{j=1}^{N}\Big(\mathbb{I}[y_i > y_j] + \mathbb{I}[y_i < y_j]\Big)}.$$

  When $I = [N]$, the above measure recovers AUC.
- Testing performance - unobserved patterns: the same as above, but use I for the testing data such that their activation / decision patterns $\boldsymbol{r}_{1:L}$ are *unobserved* in the training data.

The results are in Table 4.C.1. There are some interesting findings. For example, there is an exponential number of possible patterns, but the number of patterns that appear in the dataset is quite small. The ratio of testing points with unobserved patterns is also small, but these unobserved branching / leaves seem to be controlled properly. They do not lead to completely different performance compared to those that are observed during training.

4.C.2 Visualization

Here we visualize the learned locally constant network on the HIV dataset in the representation of its equivalent oblique decision tree in Fig. 4.C.1. Since the dimension of Morgan fingerprint [122] is quite high (2,048), here we only visualize the top-K weights (in terms of the absolute value) for each decision node. We also normalize each weight such that the ℓ_1 norm of each weight is 1. Since the task is evaluated by ranking (AUC), we visualize the leaf nodes in terms of the ranking of output probability among the leaf nodes (the higher the more likely).

Note that a complete visualization requires some engineering efforts. Our main contribution here is the algorithm that transforms an LCN to an oblique decision tree, rather than the visualization of oblique decision trees, so we only provide the initial visualization as a proof of concept.

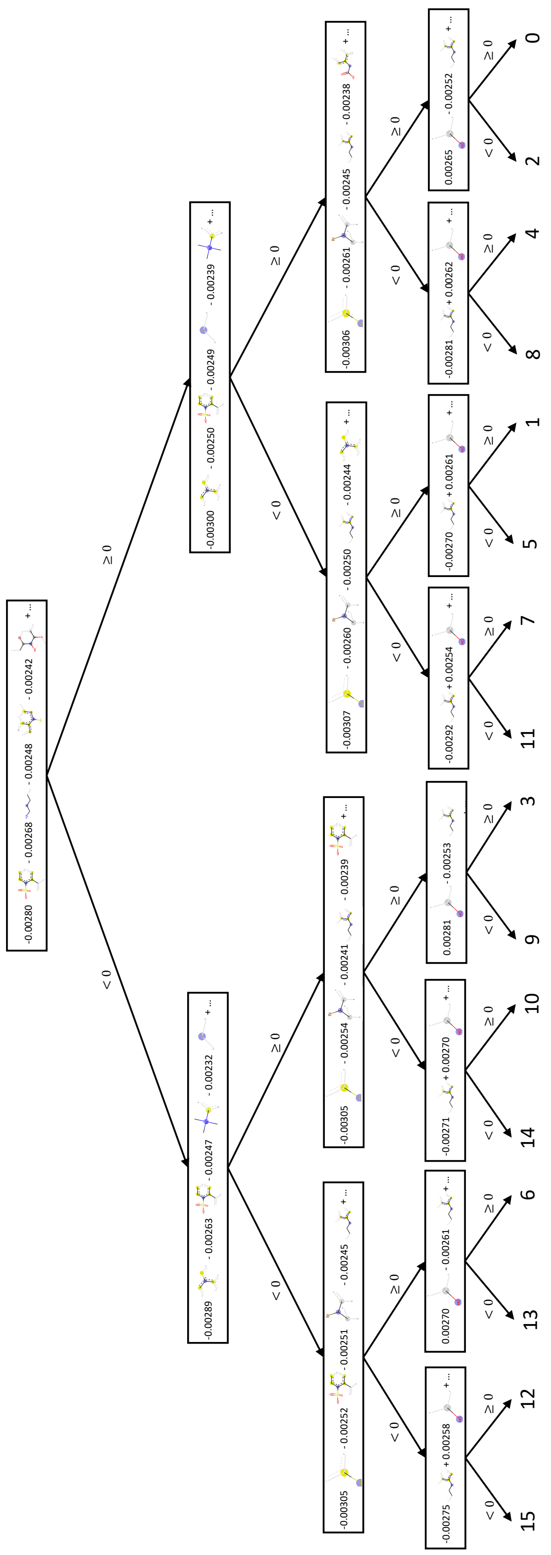


Figure 4.C.1: Visualization of learned locally constant network in the representation of oblique decision trees using the proof of Theorem 4.4. The number in the leaves indicates the ranking of output probability among the 16 leaves (the exact value is not important). See the descriptions in Appendix 4.C.2.

Chapter 5

Tight Certificates of Adversarial Robustness for Randomly Smoothed Classifiers

5.1 Introduction

Many powerful classifiers lack robustness in the sense that a slight, potentially unnoticeable manipulation of the input features, e.g., by an adversary, can cause the classifier to change its prediction [50]. The effect is clearly undesirable in decision critical applications. Indeed, a lot of recent work has gone into analyzing such failures together with providing certificates of robustness.

Robustness can be defined with respect to a variety of metrics that bound the magnitude or the type of adversarial manipulation. The most common approach to searching for violations is by finding an adversarial example within a small neighborhood of the example in question, e.g., using gradient-based algorithms [42, 50, 95]. The downside of such approaches is that failure to discover an adversarial example does not mean that another technique could not find one. For this reason, a recent line of work has instead focused on certificates of robustness, i.e., guarantees that ensure, for specific classes of methods, that no adversarial examples exist within a certified region. Unfortunately, obtaining exact guarantees can be computationally intractable [70, 93, 143], and guarantees that scale to realistic architectures have remained somewhat conservative [31, 99, 151, 154, 163].

Ensemble classifiers have recently been shown to yield strong guarantees of robustness [29]. The ensembles, in this case, are simply induced from randomly perturbing the input to a base classifier. The guarantees state that, given an additive isotropic Gaussian noise on the input example, an adversary cannot alter the prediction of the corresponding ensemble within an ℓ_2 radius, where the radius depends on the noise variance as well as the ensemble margin at the given point [29].

In this work, we substantially extend robustness certificates for such noise-induced ensembles. We provide guarantees for alternative metrics and noise distributions (e.g., uniform), develop a stratified likelihood ratio analysis that allows us to provide certificates of robustness over continuous spaces with respect to ℓ_1 and ℓ_∞ distances as well as discrete spaces with respect to ℓ_0 distance, which are *tight* and *applicable* to any measurable classifiers. We also introduce scalable algorithms for computing the certificates. The guarantees can be further tightened by introducing additional assumptions about the family of classifiers. We illustrate this in the context of ensembles derived from decision trees. Empirically, our ensemble classifiers yield the state-of-the-art certified guarantees with respect to ℓ_0 bounded adversaries across image and molecule datasets in comparison to the previous method adapted from continuous spaces.

5.2 Related Work

In a classification setting, the role of robustness certificates is to guarantee a constant classification within a local region; a certificate is always sufficient to claim robustness. When a certificate is both sufficient and necessary, it is called an exact certificate. For example, the exact ℓ_2 certificate of a linear classifier is the ℓ_2 distance between the classifier and a given point. Below we focus the discussions on the recent development of robustness guarantees for deep networks.

Most of the exact methods are derived on piecewise linear networks, whose mixed integer linear representations [83] enable the usage of mixed integer linear programming [24, 36, 44, 93, 143] or satisfiability modulo theories [22, 40, 70, 124] for finding the exact adversary under an ℓ_q radius. However, the exact methods are in general NP-complete, and thus do not scale to large problems [143].

A certificate that only holds a sufficient condition is conservative but can be more scalable than exact methods. Through relaxation, such guarantees may be derived as a linear program [154, 156], a semidefinite program [117, 118], or a dual optimization problem [38, 39]. Alternative approaches conduct layer-wise relaxations of feasible neuron values to derive the certificates [52, 99, 130, 151, 163]. Unfortunately, there is no empirical evidence of an effective certificate from the above methods in large scale problems. This does not entail that the certificates are not tight enough in practice; it might also be attributed to the fact that it is challenging to obtain a robust network in a large scale setting.

Recent works propose a new modeling scheme that ensembles a classifier by input randomization [21, 91], mostly done via an additive isotropic Gaussian noise. Lecuyer et al. [82] first propose a certificate based on differential privacy, which is improved by Li et al. [90] using Rényi divergence. Cohen et al. [29] proceed with the analysis by proving the *tight* certificate with respect to *all the measurable classifiers* based on the Neyman-Pearson Lemma [108], which yields the state-of-the-art provably robust classifier. However, the tight certificate is tailored to an isotropic Gaussian distribution and ℓ_2 metric, while we generalize the result across broad classes of distributions and metrics. In addition, we show that such tight guarantee can be tightened with assumptions about the classifier.

Our method of certification also yields the first tight and actionable ℓ_0 robustness certificates in discrete domains (cf. continuous domains where an adversary is easy to find [50]). Robustness guarantees in discrete domains are combinatorial in nature and thus challenging to obtain. Indeed, even for simple binary vectors, verifying robustness requires checking an exponential number of predictions for any black-box model.[1]

5.3 Certification Methodology

Given an input $\boldsymbol{x} \in \mathcal{X}$, a randomization scheme ϕ assigns a probability mass/density $\Pr(\phi(\boldsymbol{x}) = \boldsymbol{z})$ for each randomized outcome $\boldsymbol{z} \in \mathcal{X}$. We can define a probabilistic classifier either by specifying the associated conditional distribution $\mathbb{P}(y|\boldsymbol{x})$ for a class $y \in \mathcal{Y}$ or by viewing it as a random function $f(\boldsymbol{x})$ where the randomness in the output is independent for each $\boldsymbol{x}$. We compose the randomization scheme ϕ with a classifier f to get a randomly smoothed classifier $\mathbb{E}_{\phi}[\mathbb{P}(y|\phi(\boldsymbol{x}))]$, where the probability for outputting a class $y \in \mathcal{Y}$ is denoted as $\Pr(f(\phi(\boldsymbol{x})) = y)$ and abbreviated as p, whenever $f, \phi, \boldsymbol{x}$ and y are clear from the context. Under this setting, we first develop our framework for tight robustness certificates in §5.3.1, exemplify the framework in §5.3.2-5.3.4, and illustrate how the guarantees can be refined with further assumption in §5.3.5-5.3.6. We defer all the proofs to Appendix 5.A.

5.3.1 A Framework for Tight Certificates of Robustness

In this section, we develop our framework for deriving tight certificates of robustness for randomly smoothed classifiers, which will be instantiated in the following sections.

Point-wise certificate. Given p, we first identify a tight lower bound on the probability score $\Pr(f(\phi(\bar{\boldsymbol{x}})) = y)$ for another (neighboring) point $\bar{\boldsymbol{x}} \in \mathcal{X}$. Here we denote the set of measurable classifiers with respect to ϕ as $\mathcal{F}$. Without any additional assumptions on f, a lower bound can be found by the minimization problem:

$$\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p) \triangleq \min_{\bar{f}\in\mathcal{F}:\Pr(\bar{f}(\phi(\boldsymbol{x}))=y)=p} \Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y) \leq \Pr(f(\phi(\bar{\boldsymbol{x}})) = y). \tag{5.1}$$

Note that the bound is tight since f satisfies the constraint.

Regional certificate. We can extend the point-wise certificate $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$ to a regional certificate by examining the worst case $\bar{\boldsymbol{x}}$ over the neighboring region around $\boldsymbol{x}$.

[1]We are aware of two concurrent works also yielding certificates in discrete domain [67, 68].

Formally, given an ℓ_q metric $\|\cdot\|_q$, the neighborhood around $\boldsymbol{x}$ with radius r is defined as $\mathcal{B}_{r,q}(\boldsymbol{x}) \triangleq \{\bar{\boldsymbol{x}} \in \mathcal{X} : \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_q \le r\}$. Assuming $p = \Pr(f(\phi(\boldsymbol{x})) = y) > 0.5$ for a $y \in \mathcal{Y}$, a robustness certificate on the ℓ_q radius can be found by

$$R(\boldsymbol{x}, p, q) \triangleq \sup r, \;\; s.t. \min_{\bar{\boldsymbol{x}} \in \mathcal{B}_{r,q}(\boldsymbol{x})} \rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p) > 0.5. \tag{5.2}$$

Essentially, the certificate $R(\boldsymbol{x}, p, q)$ entails the following robustness guarantee:

$$\forall \bar{\boldsymbol{x}} \in \mathcal{X} : \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_q < R(\boldsymbol{x}, p, q), \text{we have } \Pr(f(\phi(\bar{\boldsymbol{x}})) = y) > 0.5. \tag{5.3}$$

When the maximum can be attained in Eq. (5.2) (which will be the case in ℓ_0 norm), the above $<$ can be replaced with $\le$. Note that here we assume $\Pr(f(\phi(\boldsymbol{x})) = y) > 0.5$ and ignore the case that $0.5 \ge \Pr(f(\phi(\bar{\boldsymbol{x}})) = y) > \max_{y' \neq y} \Pr(f(\phi(\bar{\boldsymbol{x}})) = y')$. By definition, the certified radius $R(\boldsymbol{x}, p, q)$ is tight for binary classification, and provides a reasonable sufficient condition to guarantee robustness for $|\mathcal{Y}| > 2$. The tight guarantee for $|\mathcal{Y}| > 2$ will involve the maximum prediction probability over all the remaining classes (see Theorem 1 of [29]). However, when the prediction probability $p = \Pr(f(\phi(\boldsymbol{x})) = y)$ is intractable to compute and relies on statistical estimation for each class y (e.g., when f is a deep network), the tight guarantee is statistically challenging to obtain. The actual algorithm used by Cohen et al. [29] is also a special case of Eq. (5.2).

5.3.2 A Warm-up Example: the Uniform Distribution

To illustrate the framework, we show a simple (but new) scenario when $\mathcal{X} = \mathbb{R}^d$ and ϕ is an additive uniform noise with a parameter $\gamma \in \mathbb{R}_{>0}$:

$$\phi(\boldsymbol{x})_i = \boldsymbol{x}_i + \boldsymbol{\epsilon}_i, \boldsymbol{\epsilon}_i \overset{i.i.d.}{\sim} \text{Uniform}([-\gamma, \gamma]), \forall i \in \{1, \ldots, d\}. \tag{5.4}$$

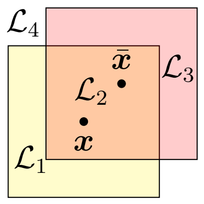


Figure 5.3.1: Uniform distributions.

Given two points $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$, as illustrated in Fig. 5.3.1, we can partition the space $\mathbb{R}^d$ into 4 disjoint regions: $\mathcal{L}_1 = \mathcal{B}_{\gamma,\infty}(\boldsymbol{x}) \backslash \mathcal{B}_{\gamma,\infty}(\bar{\boldsymbol{x}})$, $\mathcal{L}_2 = \mathcal{B}_{\gamma,\infty}(\boldsymbol{x}) \cap \mathcal{B}_{\gamma,\infty}(\bar{\boldsymbol{x}})$, $\mathcal{L}_3 =$

$\mathcal{B}_{\gamma,\infty}(\bar{\boldsymbol{x}})\backslash\mathcal{B}_{\gamma,\infty}(\boldsymbol{x})$ and $\mathcal{L}_4 = \mathbb{R}^d\backslash(\mathcal{B}_{\gamma,\infty}(\bar{\boldsymbol{x}}) \cup \mathcal{B}_{\gamma,\infty}(\boldsymbol{x}))$. Accordingly, $\forall \bar{f} \in \mathcal{F}$, we can rewrite $\Pr(\bar{f}(\phi(\boldsymbol{x})) = y)$ and $\Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y)$ as follows:

$$\Pr(\bar{f}(\phi(\boldsymbol{x})) = y) = \sum_{i=1}^{4}\int_{\mathcal{L}_i} \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) \Pr(\bar{f}(\boldsymbol{z}) = y))\mathrm{d}\boldsymbol{z} = \sum_{i=1}^{4}\pi_i\int_{\mathcal{L}_i} \Pr(\bar{f}(\boldsymbol{z}) = y))\mathrm{d}\boldsymbol{z},$$
$$\Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y) = \sum_{i=1}^{4}\int_{\mathcal{L}_i} \Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z}) \Pr(\bar{f}(\boldsymbol{z}) = y))\mathrm{d}\boldsymbol{z} = \sum_{i=1}^{4}\bar{\pi}_i\int_{\mathcal{L}_i} \Pr(\bar{f}(\boldsymbol{z}) = y))\mathrm{d}\boldsymbol{z},$$

where $\pi_{1:4} = ((2\gamma)^{-d}, (2\gamma)^{-d}, 0, 0)$, and $\bar{\pi}_{1:4} = (0, (2\gamma)^{-d}, (2\gamma)^{-d}, 0)$. With this representation, it is clear that, in order to solve Eq. (5.1), we only have to consider the integral behavior of $\bar{f}$ within each region $\mathcal{L}_1, \ldots, \mathcal{L}_4$. Concretely, we have:

$$\begin{aligned}\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p) &= \min_{\bar{f}\in\mathcal{F}:\sum_{i=1}^{4}\pi_i\int_{\mathcal{L}_i}\Pr(\bar{f}(\boldsymbol{z})=y))\mathrm{d}\boldsymbol{z}=p} \sum_{i=1}^{4}\bar{\pi}_i\int_{\mathcal{L}_i}\Pr(\bar{f}(\boldsymbol{z}) = y))\mathrm{d}\boldsymbol{z} \\ &= \min_{\substack{g:\{1,2,3,4\}\to[0,1],\\ \pi_1|\mathcal{L}_1|g(1)+\pi_2|\mathcal{L}_2|g(2)=p}} \bar{\pi}_2|\mathcal{L}_2|g(2) + \bar{\pi}_3|\mathcal{L}_3|g(3) = \min_{\substack{g:\{1,2,3,4\}\to[0,1],\\ \pi_1|\mathcal{L}_1|g(1)+\pi_2|\mathcal{L}_2|g(2)=p}} \bar{\pi}_2|\mathcal{L}_2|g(2),\end{aligned}$$

where the second equality filters the components with $\pi_i = 0$ or $\bar{\pi}_i = 0$, and the last equality is due to the fact that $g(3)$ is unconstrained and minimizes the objective when $g(3) = 0$. Since $\pi_2 = \bar{\pi}_2$,

$$\begin{cases}\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p) = 0, & \text{if } 0 \le p \le \pi_1|\mathcal{L}_1| = \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_1), \\ \rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p) = p - \pi_1|\mathcal{L}_1|, & \text{if } 1 \ge p > \pi_1|\mathcal{L}_1| = \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_1).\end{cases}$$

To obtain the regional certificate, the minimizers of $\min_{\bar{\boldsymbol{x}}\in\mathcal{B}_{r,q}(\boldsymbol{x})}\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$ are simply the points that maximize the volume of $\mathcal{L}_1 = \mathcal{B}_1\backslash\mathcal{B}_2$. Accordingly,

Proposition 5.1. *If $\phi(\cdot)$ is defined as Eq. (5.4), we have $R(\boldsymbol{x}, p, q = 1) = 2p\gamma - \gamma$ and $R(\boldsymbol{x}, p, q = \infty) = 2\gamma - 2\gamma(1.5 - p)^{1/d}$.*

Discussion. Our goal here was to illustrate how certificates can be computed with the uniform distribution using our technique. However, the certificate radius itself is inadequate in this case. For example, $R(\boldsymbol{x}, p, q = 1) \le \gamma$, which arises from the bounded support in the uniform distribution. The derivation nevertheless provides some insights about how one can compute the point-wise certificate $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$. The key step is to partition the space into regions $\mathcal{L}_1, \dots, \mathcal{L}_4$, where the likelihoods $\Pr(\phi(\boldsymbol{x}) = \boldsymbol{z})$ and $\Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z})$ are both constant within each region $\mathcal{L}_i$. The property allows us to substantially reduce the optimization problem in Eq. (5.1) to finding a single probability value $g(i) \in [0, 1]$ for each region $\mathcal{L}_i$.

5.3.3 A General Lemma for Point-wise Certificate

In this section, we generalize the idea in §5.3.2 to find the point-wise certificate $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$. For each point $\boldsymbol{z} \in \mathcal{X}$, we define the likelihood ratio $\eta_{\boldsymbol{x},\bar{\boldsymbol{x}}}(\boldsymbol{z}) \triangleq \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) / \Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z})$.[2] If we can partition $\mathcal{X}$ into n regions $\mathcal{L}_1, \dots, \mathcal{L}_n : \cup_{i=1}^{n} \mathcal{L}_i = \mathcal{X}$ for some $n \in \mathbb{Z}_{>0}$, such that the likelihood ratio within each region $\mathcal{L}_i$ is a constant $\eta_i \in [0, \infty]$: $\eta_{\boldsymbol{x},\bar{\boldsymbol{x}}}(\boldsymbol{z}) = \eta_i, \forall \boldsymbol{z} \in \mathcal{L}_i$, then we can sort the regions such that $\eta_1 \geq \eta_2 \geq \cdots \geq \eta_n$. Note that $\mathcal{X}$ can still be uncountable (see the example in §5.3.2).

Informally, we can always "normalize" $\bar{f}$ so that it predicts a constant probability value $g(i) \in [0, 1]$ within each likelihood ratio region $\mathcal{L}_i$. This preserves the integral over $\mathcal{L}_i$ and thus over $\mathcal{X}$, generalizing the scenario in §5.3.2. Moreover, to minimize $\Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y)$ under a fixed budget $\Pr(\bar{f}(\phi(\boldsymbol{x})) = y)$, as in Eq. (5.1), it is advantageous to set $\bar{f}(\boldsymbol{z})$ to y in regions with high likelihood ratio. These arguments suggest a greedy algorithm for solving Eq. (5.1) by iteratively assigning $\bar{f}(\boldsymbol{z}) = y, \forall \boldsymbol{z} \in \mathcal{L}_i$ for $i \in (1, 2, \dots)$ until the budget constraint is met. Formally,

Lemma 5.2. *$\forall \boldsymbol{x}, \bar{\boldsymbol{x}} \in \mathcal{X}$, $p \in [0, 1]$, let $H^* \triangleq \min_{H \in \{1,\dots,n\} : \sum_{i=1}^{H} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) \geq p} H$, then $\eta_{H^*} > 0$, any f^* satisfying Eq. (5.5) is a minimizer of Eq. (5.1),*

$$\forall i \in \{1, 2, \dots, n\}, \forall \boldsymbol{z} \in \mathcal{L}_i, \Pr(f^*(\boldsymbol{z}) = y) = \begin{cases} 1, & \text{if } i < H^*, \\ \frac{p - \sum_{i=1}^{H^*-1} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i)}{\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_{H^*})}, & \text{if } i = H^*, \\ 0, & \text{if } i > H^*. \end{cases} \quad (5.5)$$

and $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p) = \sum_{i=1}^{H^*-1} \Pr(\phi(\bar{\boldsymbol{x}}) \in \mathcal{L}_i) + (p - \sum_{i=1}^{H^*-1} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i)) / \eta_{H^*}$

[2]If $\Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z}) = \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) = 0$, $\eta_{\boldsymbol{x},\bar{\boldsymbol{x}}}(\boldsymbol{z})$ can be defined arbitrarily in $[0, \infty]$ without affecting the solution in Lemma 5.2.

We remark that Eq. (5.1) and Lemma 5.2 can be interpreted as a likelihood ratio testing [108], by casting $\Pr(\phi(\boldsymbol{x}) = \boldsymbol{z})$ and $\Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z})$ as likelihoods for two hypothesis with the significance level p. We refer the readers to [144] to see a similar Lemma derived under the language of hypothesis testing.

Remark 5.3. *$\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$ is an increasing continuous function of p; if $\eta_1 < \infty$, $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$ is a strictly increasing continuous function of p; if $\eta_1 < \infty$ and $\eta_n > 0$, $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}} : [0,1] \to [0,1]$ is a bijection.*

Remark 5.3 will be used in §5.3.4 to derive an efficient algorithm to compute robustness certificates.

Discussion. Given $\mathcal{L}_i$, $\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i)$, and $\Pr(\phi(\bar{\boldsymbol{x}}) \in \mathcal{L}_i), \forall i \in [n]$, Lemma 5.2 provides an $O(n)$ method to compute $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$. For any actual randomization ϕ, the key is to find a partition $\mathcal{L}_1, \ldots, \mathcal{L}_n$ such that $\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i)$ and $\Pr(\phi(\bar{\boldsymbol{x}}) \in \mathcal{L}_i)$ are easy to compute. Having constant likelihoods in each $\mathcal{L}_i : \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) = \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}'), \forall \boldsymbol{z}, \boldsymbol{z}' \in \mathcal{L}_i$ (cf. only having constant likelihood ratio η_i) is a way to simplify $\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) = |\mathcal{L}_i| \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z})$, and similarly for $\Pr(\phi(\bar{\boldsymbol{x}}) \in \mathcal{L}_i)$.

5.3.4 A Discrete Distribution for ℓ_0 Robustness

We consider ℓ_0 robustness guarantees in a discrete space $\mathcal{X} = \left\{0, \frac{1}{K}, \frac{2}{K}, \ldots, 1\right\}^d$ for some $K \in \mathbb{Z}_{>0}$;[3] we define the following discrete distribution with a parameter $\alpha \in (0,1)$, independent and identically distributed for each dimension $i \in \{1, 2, \ldots, d\}$:

$$\begin{cases} \Pr(\phi(\boldsymbol{x})_i = \boldsymbol{x}_i) = \alpha, \\ \Pr(\phi(\boldsymbol{x})_i = z) = (1-\alpha)/K \triangleq \beta \in (0, 1/K), \quad \text{if } z \in \left\{0, \frac{1}{K}, \frac{2}{K}, \ldots, 1\right\} \text{ and } z \neq \boldsymbol{x}_i. \end{cases} \tag{5.6}$$

Here $\phi(\cdot)$ can be regarded as a composition of a Bernoulli random variable and a uniform random variable. Due to the symmetry of the randomization with respect

[3]More generally, the method applies to the ℓ_0 / Hamming distance in a Hamming space (i.e., fixed length sequences of tokens from a discrete set, e.g., $(\spadesuit 10, \spadesuit J, \spadesuit Q, \spadesuit K, \spadesuit A) \in \{\spadesuit A, \spadesuit K, ..., \clubsuit 2\}^5$).

to all the configurations of $\boldsymbol{x}, \bar{\boldsymbol{x}} \in \mathcal{X}$ such that $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_0 = r$ (for some $r \in \mathbb{Z}_{\geq 0}$), we have the following Lemma for the equivalence of $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}$:

Lemma 5.4. *If $\phi(\cdot)$ is defined as Eq. (5.6), given $r \in \mathbb{Z}_{\geq 0}$, define the canonical vectors $\boldsymbol{x}_C \triangleq (0, 0, \cdots, 0)$ and $\bar{\boldsymbol{x}}_C \triangleq (1, 1, \cdots, 1, 0, 0, \cdots, 0)$, where $\|\bar{\boldsymbol{x}}_C\|_0 = r$. Let $\rho_r \triangleq \rho_{\boldsymbol{x}_C,\bar{\boldsymbol{x}}_C}$. Then for all $\boldsymbol{x}, \bar{\boldsymbol{x}}$ such that $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_0 = r$, we have $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}} = \rho_r$.*

Based on Lemma 5.4, finding $R(\boldsymbol{x}, p, q)$ for a given p, it suffices to find the maximum r such that $\rho_r(p) > 0.5$. Since the likelihood ratio $\eta_{\boldsymbol{x},\bar{\boldsymbol{x}}}(\boldsymbol{z})$ is always positive and finite, the inverse ρ_r^{-1} exists (due to Remark 5.3), which allows us to pre-compute $\rho_r^{-1}(0.5)$ and check $p > \rho_r^{-1}(0.5)$ for each $r \in \mathbb{Z}_{\geq 0}$, instead of computing $\rho_r(p)$ for each given p and r. Then $R(\boldsymbol{x}, p, q)$ is simply the maximum r such that $p > \rho_r^{-1}(0.5)$. Below we discuss how to compute $\rho_r^{-1}(0.5)$ in a scalable way. Our first step is to identify a set of likelihood ratio regions $\mathcal{L}_1, \ldots, \mathcal{L}_n$ such that $\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i)$ and $\Pr(\phi(\bar{\boldsymbol{x}}) \in \mathcal{L}_i)$ as used in Lemma 5.2 can be computed efficiently. Note that, due to Lemma 5.4, it suffices to consider $\boldsymbol{x}_C, \bar{\boldsymbol{x}}_C$ such that $\|\bar{\boldsymbol{x}}_C\|_0 = r$ throughout the derivation.

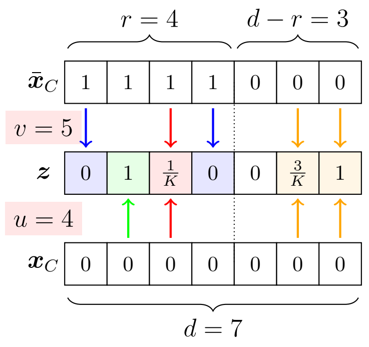


Figure 5.3.2: Illustration for Eq. (5.7)

For an ℓ_0 radius $r \in \mathbb{Z}_{\geq 0}$, $\forall (u, v) \in \{0, 1, \ldots, d\}^2$, we construct the region

$$\mathcal{L}(u, v; r) \triangleq \{\boldsymbol{z} \in \mathcal{X} : \Pr(\phi(\boldsymbol{x}_C) = \boldsymbol{z}) = \alpha^{d-u}\beta^u, \Pr(\phi(\bar{\boldsymbol{x}}_C) = \boldsymbol{z}) = \alpha^{d-v}\beta^v\}, \quad (5.7)$$

which contains points that can be obtained by "flipping" u coordinates from $\boldsymbol{x}_C$ or v coordinates from $\bar{\boldsymbol{x}}_C$. See Figure 5.3.2 for an illustration, where different colors represent different types of coordinates: orange means both $\boldsymbol{x}_C, \bar{\boldsymbol{x}}_C$ are flipped on this coordinate and they were initially the same; red means both are flipped and were initially different; green means only $\boldsymbol{x}_C$ is flipped and blue means only $\bar{\boldsymbol{x}}_C$ is flipped. By denoting the numbers of these coordinates as $i, j^*, u - i - j^*, v - i - j^*$, respectively, we have the following formula for computing the cardinality of each region $|\mathcal{L}(u, v; r)|$.

Algorithm 1 Computing $\rho_r^{-1}(0.5)$

```
 1: sort $\{(u_i, v_i)\}_{i=1}^n$ by likelihood ratio
 2: $p, \rho_r = 0, 0$
 3: for $i = 1, \ldots, n$ do
 4:     $p' = \alpha^{d-u_i}\beta^{u_i}$
 5:     $\rho'_r = \alpha^{d-v_i}\beta^{v_i}$
 6:     $\Delta\rho_r = \rho'_r \times |\mathcal{L}(u_i, v_i; r)|$
 7:     if $\rho_r + \Delta\rho_r < 0.5$ then
 8:         $\rho_r = \rho_r + \Delta\rho_r$
 9:         $p = p + p' \times |\mathcal{L}(u_i, v_i; r)|$
10:     else
11:         $p = p + p' \times (0.5 - \rho_r)/\rho'_r$
12:         return $p$
13:     end if
14: end for
```

Lemma 5.5. *For any $u, v \in \{0, 1, \ldots, d\}, u \leq v, r \in \mathbb{Z}_{\geq 0}$ we have $|\mathcal{L}(u, v; r)| = |\mathcal{L}(v, u; r)|$, and*

$$|\mathcal{L}(u, v; r)| = \sum_{i=\max\{0, v-r\}}^{\min\left(u, d-r, \lfloor \frac{u+v-r}{2} \rfloor\right)} \frac{(K-1)^{j^*} r!}{(u-i-j^*)!(v-i-j^*)!j^*!} \frac{K^i (d-r)!}{(d-r-i)!i!},$$

where $j^* \triangleq u + v - 2i - r$.

Therefore, for a fixed r, the complexity of computing all the cardinalities $|\mathcal{L}(u, v; r)|$ is $\Theta(d^3)$. Since each region $\mathcal{L}(u, v; r)$ has a constant likelihood ratio $\alpha^{v-u}\beta^{u-v}$ and we have $\cup_{u=0}^d \cup_{v=0}^d \mathcal{L}(u, v; r) = \mathcal{X}$, we can apply the regions to find the function $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}} = \rho_r$ via Lemma 5.2. Under this representation, the number of nonempty likelihood ratio regions n is bounded by $(d + 1)^2$, the perturbation probability $\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}(u, v; r))$ used in Lemma 5.2 is simply $\alpha^{d-u}\beta^u|\mathcal{L}(u, v; r)|$, and similarly for the $\Pr(\phi(\bar{\boldsymbol{x}}) \in \mathcal{L}(u, v; r))$. Based on Lemma 5.2 and Lemma 5.5, we may use a for-loop to compute the bijection $\rho_r(\cdot)$ for the input p until $\rho_r(p) = 0.5$, and return the corresponding p as $\rho_r^{-1}(0.5)$. The procedure is illustrated in Algorithm 1.

Scalable implementation. In practice, Algorithm 1 can be challenging to implement; the probability values (e.g., $\alpha^{d-u}\beta^u$) can be extremely small, which is infeasible to be computationally represented using floating points. If we set α to be a rational number, both α and β can be represented in fractions, and thus all the corresponding probability values can be represented by two (large) integers; we also observe that computing the (large) cardinality $|\mathcal{L}(u, v; r)|$ is feasible in modern large integer computation frameworks in practice (e.g., `python`), which motivates us to adapt the computation in Algorithm 1 to large integers.

For simplicity, we assume $\alpha = \alpha'/100$ with some $\alpha' \in \mathbb{Z} : 100 \geq \alpha' \geq 0$. If we define $\tilde{\alpha} \triangleq 100K\alpha \in \mathbb{Z}, \tilde{\beta} \triangleq 100K\beta \in \mathbb{Z}$, we may implement Algorithm 1 in terms of the non-normalized, integer version $\tilde{\alpha}, \tilde{\beta}$. Specifically, we replace α, β and the constant 0.5 with $\tilde{\alpha}, \tilde{\beta}$ and $50K \times (100K)^{d-1}$, respectively. Then all the computations in Algorithm 1 can be trivially adapted except the division $(0.5 - \rho_r)/\rho'_r$. Since the division is bounded by $|\mathcal{L}(u_i, v_i; r)|$ (see the comparison between line 9 and line 11), we can implement the division by a binary search over $\{1, 2 \ldots, |\mathcal{L}\{u_i, v_i; r\}|\}$, which will result in an upper bound with an error bounded by p' in the original space, which is in turn bounded by α^d assuming $\alpha > \beta$. Finally, to map the computed, unnormalized $\rho_r^{-1}(0.5)$, denoted as $\tilde{\rho}_r^{-1}(0.5)$, back to the original space, we find an upper bound of $\rho_r^{-1}(0.5)$ up to the precision of 10^{-c} for some $c \in \mathbb{Z}_{>0}$ (we set $c = 20$ in the experiments): we find the smallest upper bound of $\tilde{\rho}_r^{-1}(0.5) \leq \hat{\rho} \times (10K)^c (100K)^{d-c}$ over $\hat{\rho} \in \{1, 2, \ldots, 10^c\}$ via binary search, and report an upper bound of $\rho_r^{-1}(0.5)$ as $\hat{\rho} \times 10^{-c}$ with an error bounded by $10^{-c} + \alpha^d$ in total. Note that an upper bound of $\rho_r^{-1}(0.5)$ is still a valid certificate.

As a side note, simply computing the probabilities in the log-domain will lead to uncontrollable approximate results due to floating point arithmetic; using large integers to ensure a verifiable approximation error in Algorithm 1 is necessary to ensure a computationally accurate certificate.

5.3.5 Connection Between the Discrete Distribution and an Isotropic Gaussian Distribution

When the inputs are binary vectors $\mathcal{X} = \{0,1\}^d$, one may still apply the prior work [29] using an additive isotropic Gaussian noise ϕ to obtain an ℓ_0 certificates since there is a bijection between ℓ_0 and ℓ_2 distance in $\{0,1\}^d$. If one uses a denoising function $\zeta(\cdot)$ that projects each randomized coordinate $\phi(\boldsymbol{x})_i \in \mathbb{R}$ back to the space $\{0,1\}$ using the (likelihood ratio testing) rule

$$\zeta(\phi(\boldsymbol{x}))_i = \mathbb{I}\{\phi(\boldsymbol{x})_i > 0.5\}, \forall i \in [d],$$

then the composition $\zeta \circ \phi$ is equivalent to our discrete randomization scheme with $\alpha = \Phi(0.5; \mu = 0, \sigma^2)$, where Φ is the CDF function of the Gaussian distribution with mean μ and variance σ^2.

If we apply a classifier upon the composition (or, equivalently, the discrete randomization scheme), then the certificates obtained via the discrete distribution is always tighter than the one via the Gaussian distribution. Concretely, we denote $\mathcal{F}_\zeta \subset \mathcal{F}$ as the set of measurable functions with respect to the Gaussian distribution that can be written as the composition $\bar{f}' \circ \zeta$ for some $\bar{f}'$, and we have

$$\min_{\bar{f} \in \mathcal{F}_\zeta : \Pr(\bar{f}(\phi(\boldsymbol{x}))=y)=p} \Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y) \geq \min_{\bar{f} \in \mathcal{F} : \Pr(\bar{f}(\phi(\boldsymbol{x}))=y)=p} \Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y),$$

where the LHS corresponds to the certificate derived from the discrete distribution (i.e., applying ζ to an isotropic Gaussian), and the RHS corresponds to the certificate from the Gaussian distribution.

5.3.6 A Certificate with Additional Assumptions

In the previous analyses, we assume nothing but the measurability of the classifier. If we further make assumptions about the functional class of the classifier, we can obtain a tighter certificate than the ones outlined in §5.3.1. Assuming an extra denoising step in the classifier over an additive Gaussian noise as illustrated in §5.3.5 is one example.

Here we illustrate the idea with another example. We assume that the inputs are binary vectors $\mathcal{X} = \{0,1\}^d$, the outputs are binary $\mathcal{Y} = \{0,1\}$, and that the classifier is a decision tree that each input coordinate can be used at most once in the entire tree. Under the discrete randomization scheme, the prediction probability can be computed via tree recursion, since a decision tree over the discrete randomization scheme can be interpreted as assigning a probability of visiting the left child and the right child for each decision node. To elaborate, we denote $\texttt{idx}[i], \texttt{left}[i]$, and $\texttt{right}[i]$ as the split feature index, the left child and the right child of the i^{th} node. Without loss of generality, we assume that each decision node i routes its input to the right branch if $\boldsymbol{x}_{\texttt{idx}[i]} = 1$. Then $\Pr(f(\phi(\boldsymbol{x})) = 1)$ can be found by the recursion

$$\texttt{pred}[i] = \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=1\}}\beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=0\}}\texttt{pred}[\texttt{right}[i]] + \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=0\}}\beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=1\}}\texttt{pred}[\texttt{left}[i]], \tag{5.8}$$

where the boundary condition is the output of the leaf nodes. Effectively, we are recursively aggregating the partial solutions found in the left subtree and the right subtree rooted at each node i, and $\texttt{pred}[\texttt{root}]$ is the final prediction probability. Note that changing one input coordinate in $\boldsymbol{x}_k$ is equivalent to changing the recursion in the corresponding unique node i' (if exists) that uses feature k as the splitting index, which gives

$$\texttt{pred}[i'] = \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i']}=0\}}\beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i']}=1\}}\texttt{pred}[\texttt{right}[i']] + \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i']}=1\}}\beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i']}=0\}}\texttt{pred}[\texttt{left}[i']].$$

In addition, changes in the left subtree do not affect the partial solution found in the right subtree, and vice versa. Hence, we may use dynamic programming to find the *exact* adversary under each ℓ_0 radius r by aggregating the worst case changes found in the left subtree and the right subtree rooted at each node i. See Appendix 5.B.1 for details.

5.4 Learning and Prediction in Practice

Since we focus on the development of certificates, here we only briefly discuss how we train the classifiers and compute the prediction probability $\Pr(f(\phi(\boldsymbol{x})) = y)$ in practice.

Deep networks. We follow the approach proposed by the prior work [82]: training is conducted on samples drawn from the randomization scheme via a cross entropy loss. The prediction probability $\Pr(f(\phi(\boldsymbol{x})) = y)$ is estimated by the lower bound of the Clopper-Pearson Bernoulli confidence interval [28] with 100K samples drawn from the distribution and the 99.9% confidence level. Since $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$ is an increasing function of p (Remark 5.3), a lower bound of p entails a valid certificate.

Decision trees. We train the decision tree greedily in a breadth-first ordering with a depth limit; for each split, we only search coordinates that are not used before to enforce the functional constraint in §5.3.6, and optimize a weighted gini index, which weights each training example $\boldsymbol{x}$ by the probability that it is routed to the node by the discrete randomization. The details of the training algorithm is in Appendix 5.B.2. The prediction probability is computed by Eq. (5.8).

5.5 Experiments

In this section, we validate the robustness certificate of the proposed discrete distribution ($\mathcal{D}$) in ℓ_0 norm. We compare to the state-of-the-art additive isotropic Gaussian noise ($\mathcal{N}$) [29], since an ℓ_0 certificate with radius r in $\mathcal{X} = \{0, \frac{1}{K}, \ldots, 1\}^d$ can be obtained from an ℓ_2 certificate with radius $\sqrt{r}$. Note that the derived ℓ_0 certificate from Gaussian distribution is still tight with respect to all the measurable classifiers (see Theorem 1 in [29]). We consider the following evaluation measures:

- $\mu(R)$: the average certified ℓ_0 radius $R(\boldsymbol{x}, p, q)$ (with respect to the labels) across the testing set.
- ACC@r: the certified accuracy within a radius r (the average $\mathbb{I}\{R(\boldsymbol{x}, p, q) \geq r\}$ in the testing set).

Table 5.5.1: Randomly smoothed CNN models on the MNIST dataset. The first two rows refer to the same model with certificates computed via different methods (see details in §5.3.5).

ϕ	Certificate	$\mu(R)$	ACC@r						
			$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$	$r=7$
$\mathcal{D}$	$\mathcal{D}$	**3.456**	**0.921**	**0.774**	**0.539**	**0.524**	**0.357**	**0.202**	**0.097**
$\mathcal{D}$	$\mathcal{N}$ [29]	1.799	0.830	0.557	0.272	0.119	0.021	0.000	0.000
$\mathcal{N}$	$\mathcal{N}$ [29]	2.378	0.884	0.701	0.464	0.252	0.078	0.000	0.000

5.5.1 Binarized MNIST

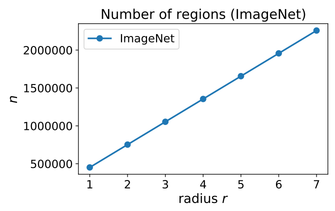
We use a $55,000/5,000/10,000$ split of the MNIST dataset for training/validation/testing. For each data point $\boldsymbol{x}$ in the dataset, we binarize each coordinate by setting the threshold as 0.5. Experiments are conducted on randomly smoothed CNN models and the implementation details are in Appendix 5.C.1.

The results are shown in Table 5.5.1. For the same randomly smoothed CNN model (the 1st and 2nd rows in Table 5.5.1), our certificates are consistently better than the ones derived from the Gaussian distribution (see §5.3.5). The gap between the average certified radius is about 1.7 in ℓ_0 distance, and the gap between the certified accuracy can be as large as 0.4. Compared to the model trained with Gaussian noise (the 3rd row in Table 5.5.1), our model is also consistently better in terms of all the measures.

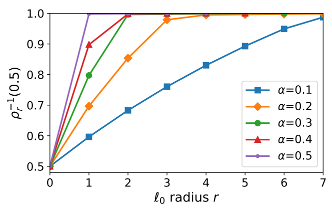
Since the above comparison between our certificates and the Gaussian-based certificates is *relative*, we conduct an exhaustive search over all the possible adversary within ℓ_0 radii 1 and 2 to study the tightness against the *exact* certificate. The resulting certified accuracies at radii 1 and 2 are 0.954 and 0.926, respectively, which suggest that our certificate is reasonably tight when $r = 1$ (0.954 vs. 0.921), but still too pessimistic when $r = 2$ (0.926 vs. 0.774). The phenomenon is expected since the certificate is based on *all the measurable functions* for the discrete distribution. A tighter certificate requires additional assumptions on the classifier such as the example in §5.3.6.

Table 5.5.2: The guaranteed accuracy of randomly smoothed ResNet50 models on ImageNet.

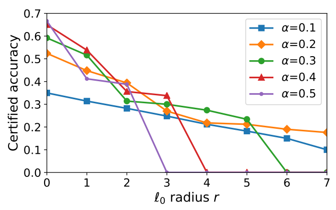
ϕ and certificate	ACC@r						
	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$	$r = 6$	$r = 7$
$\mathcal{D}$	**0.538**	**0.394**	**0.338**	**0.274**	**0.234**	**0.190**	**0.176**
$\mathcal{N}$ [29]	0.372	0.292	0.226	0.194	0.170	0.154	0.138



(a) # of nonempty $\mathcal{L}(u, v; r)$



(b) $\rho_r^{-1}(0.5)$ for an α



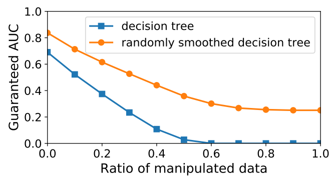
(c) The certified accuracy for an α

Figure 5.5.1: Analysis of the proposed method in the ImageNet dataset.

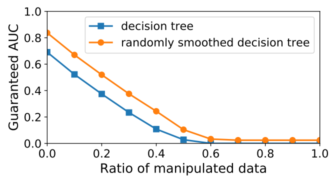
5.5.2 ImageNet

We conduct experiments on ImageNet [33], a large scale image dataset with $1,000$ labels. Following common practice, we consider the input space $\mathcal{X} = \{0, 1/255, \ldots, 1\}^{224\times224\times3}$ by scaling the images. We consider the same ResNet50 classifier [59] and learning procedure as Cohen et al. [29] with the only modification on the noise distribution. The details and visualizations can be found in Appendix 5.C.2. For comparison, we report the best guaranteed accuracy of each method for each ℓ_0 radius r in Table 5.5.2. Our model outperforms the competitor by a large margin at $r = 1$ (0.538 vs. 0.372), and consistently outperforms the baseline across different radii.

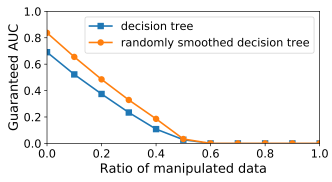
Analysis. We analyze our method in ImageNet in terms of 1) the number n of nonempty likelihood ratio region $\mathcal{L}(u, v; r)$ in Algorithm 1, 2) the pre-computed $\rho_r^{-1}(0.5)$, and 3) the certified accuracy at each α. The results are in Figure 5.5.1. For reproducability, the detailed accuracy numbers of 3) is available in Table 5.C.1 in Appendix 5.C.2, and the pre-computed $\rho_r^{-1}(0.5)$ is available at our code repository. 1) The number n of nonempty likelihood ratio regions is much smaller than the bound $(d+1)^2 = (3 \times 224 \times 224)^2$ for small radii. 2) The value $\rho_r^{-1}(0.5)$ approaches 1 more rapidly for a higher α value than a lower one. Note that $\rho_r^{-1}(0.5)$ only reaches 1 when



(a) $r = 1$



(b) $r = 2$



(c) $r = 3$

Figure 5.5.2: The guaranteed AUC in the Bace dataset across different ℓ_0 radius r and the ratio of testing data that the adversary can manipulate.

$r = d$ due to Remark 5.3. Computing $\rho_r^{-1}(0.5)$ in large integer is time-consuming, which takes about 4 days for each α and r, but this can be trivially parallelized across different α and r.[4] For each radius r and randomization parameter α, note that the 4-day computation only has to be done *once*, and the pre-computed $\rho_r^{-1}(0.5)$ can be applied to any ImageNet scale images and models. 3) The certified accuracy behaves nonlinearly across different radii; relatively, a high α value exhibits a high certified accuracy at small radii and low certified accuracy at large radii, and vice versa.

5.5.3 Chemical Property Prediction

The experiment is conducted on the Bace dataset [138], a binary classification dataset for biophysical property prediction on molecules. We use Morgan fingerprints [122] to represent molecules, which are commonly used binary features [158] indicating the presence of various chemical substructures. The dimension of the features (fingerprints) is $1,024$. Here we focus on an ablation study comparing the proposed randomly smoothed decision tree with a vanilla decision tree, where the adversary is found by dynamic programming in §5.3.6 (thus the exact worse case) and a greedy search, respectively. More details can be found in Appendix 5.C.3.

Since chemical property classification datasets are typically evaluated by AUC due to the unbalanced labels [158], we define a robust version of AUC that takes account of the radius of the adversary as well as the ratio of testing data that can be manipulated. Note that to maximally decrease the score of AUC via a positive (negative) example, the adversary only has to maximally decrease (increase) its prediction probability,

[4]As a side note, computing $\rho_r^{-1}(0.5)$ in MNIST takes less than 1 second for each α and r.

regardless of the scores of the other examples. Hence, given an ℓ_0 radius r and a ratio of testing data, we first compute the adversary for each testing data, and then find the combination of adversaries and the clean data under the ratio constraint that leads to the worst AUC score. See details in Appendix 5.C.4.

The results are in Figure 5.5.2. Empirically, the adversary of the decision tree at $r = 1$ always changes the prediction probability of a positive (negative) example to 0 (1). Hence, the plots of the decision tree model are constant across different ℓ_0 radii. The randomly smoothed decision tree is consistently more robust than the vanilla decision tree model. We also compare the exact certificate of the prediction probability with the one derived from Lemma 5.2; the average difference across the training data is 0.358 and 0.402 when r equals to 1 and 2, respectively. The phenomenon encourages the development of a classifier-aware guarantee that is tighter than the classifier-agnostic guarantee.

5.6 Discussion and Conclusion

We present a stratified approach to certifying the robustness of randomly smoothed classifiers, where the robustness guarantees can be obtained in various resolutions and perspectives, ranging from a point-wise certificate to a regional certificate and from general results to specific examples. The hierarchical investigation opens up many avenues for future extensions at different levels.

Following the publication of this work as a conference paper, Bojchevski *et al.* [17] extended the discrete randomization to be data-dependent and accelerated the certification algorithm by partitioning the space into only $2r+1$ likelihood ratio regions. In practice, given the ensemble prediction value p, their certification algorithm improves the computation of $\rho_r^{-1}(0.5)$ on ImageNet data from 4 days to less than 1 second. The ℓ_0 certificate has also been applied to label-flipping attacks by Rosenfeld *et al.* [123].

5.A Proofs

To simplify exposition, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

5.A.1 Proof of Proposition 5.1

Proof. We have

$$\begin{cases} \rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p) = 0, & \text{if } 0 \leq p \leq \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_1), \\ \rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p) = p - \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_1), & \text{if } 1 \geq p > \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_1), \end{cases}$$

where $\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_1) = \mathrm{Vol}(\mathcal{B}_1 \backslash \mathcal{B}_2)/\mathrm{Vol}(\mathcal{B}_1)$ and $\mathrm{Vol}(\mathcal{B}_1)$ is a constant given γ. Hence, the minimizers of $\min_{\bar{\boldsymbol{x}} \in \mathcal{B}_{r,q}(\boldsymbol{x})} \rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$ are simply the points that maximize the volume of $\mathcal{B}_1 \backslash \mathcal{B}_2$, or, equivalently, minimize the volume of $\mathcal{B}_1 \cap \mathcal{B}_2$. Below we re-write $\bar{\boldsymbol{x}}$ as $\boldsymbol{x} + \boldsymbol{\delta}$.

Case $q = 1$: $\forall r > 0$, we want to find a $\boldsymbol{\delta}$ s.t., $\|\boldsymbol{\delta}\|_1 = r$ and the overlapping region is minimized: (By symmetry, we assume that $\boldsymbol{\delta}_i \geq 0$ for all i)

$$\operatorname*{arg\,min}_{\boldsymbol{\delta} \geq 0 : \|\boldsymbol{\delta}\|_1 = r} \prod_{i=1}^{d} (2\gamma - \boldsymbol{\delta}_i). \tag{5.9}$$

Since $\forall i, j \in [d], i \neq j$, we know

$$(2\gamma - \boldsymbol{\delta}_i)(2\gamma - \boldsymbol{\delta}_j) = 4\gamma^2 - (\boldsymbol{\delta}_i + \boldsymbol{\delta}_j) + \boldsymbol{\delta}_i \boldsymbol{\delta}_j \geq 4\gamma^2 - (\boldsymbol{\delta}_i + \boldsymbol{\delta}_j). \tag{5.10}$$

So we can always move the mass of $\boldsymbol{\delta}_j$ to $\boldsymbol{\delta}_i$ to further decrease the product value. That means, $\boldsymbol{\delta}_1 = r, \boldsymbol{\delta}_i = 0, \forall i \neq 1$ minimizes Eq. (5.9) for a given r. As a result, we know

$$\begin{aligned} & \sup r, s.t. \min_{\boldsymbol{\delta} : \|\boldsymbol{\delta}\|_1 \leq r} \rho_{\boldsymbol{x},\boldsymbol{x}+\boldsymbol{\delta}}(p) > 0.5 && (5.11) \\ = & \sup r, s.t.\ p - \left(1 - \frac{(2\gamma)^{d-1}(2\gamma - r)}{(2\gamma)^d}\right) > 0.5 && (5.12) \\ = & 2p\gamma - \gamma && (5.13) \end{aligned}$$

Case $q = \infty$: Similarly, for $q = \infty$ case, we want to find a $\boldsymbol{\delta}$ with $\|\boldsymbol{\delta}\|_\infty = r$, and the following is minimized: (by symmetry we assume $\boldsymbol{\delta}_i \geq 0$ for all i)

$$\arg\min_{\boldsymbol{\delta}\geq 0:\|\boldsymbol{\delta}\|_\infty = r} \prod_{i=1}^{d} (2\gamma - \boldsymbol{\delta}_i).$$

In this case, we should set $\boldsymbol{\delta}_i = r$ for all i, which means

$$\sup r, s.t. \min_{\boldsymbol{\delta}:\|\boldsymbol{\delta}\|_\infty \leq r} \rho_{\boldsymbol{x},\boldsymbol{x}+\boldsymbol{\delta}}(p) > 0.5 \tag{5.14}$$

$$= \sup r, s.t.\ p - \left(1 - \frac{(2\gamma - r)^d}{(2\gamma)^d}\right) > 0.5 \tag{5.15}$$

$$= \sup r, s.t.\ \frac{(2\gamma - r)^d}{(2\gamma)^d} > 1.5 - p \tag{5.16}$$

It remains to see that

$$\frac{(2\gamma - r)^d}{(2\gamma)^d} > 1.5 - p$$
$$\iff 2\gamma - r > 2\gamma(1.5 - p)^{1/d}$$
$$\iff 2\gamma - 2\gamma(1.5 - p)^{1/d} > r. \qquad \square$$

5.A.2 Proof of Lemma 5.2

Proof. $\forall \bar{f} \in \mathcal{F}$, We may rewrite the probabilities in an integral form:

$$\Pr(\bar{f}(\phi(\boldsymbol{x})) = y) = \sum_{i=1}^{n} \int_{\mathcal{L}_i} \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) \Pr(\bar{f}(\boldsymbol{z}) = y) \mathrm{d}\boldsymbol{z},$$
$$\Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y) = \sum_{i=1}^{n} \int_{\mathcal{L}_i} \Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z}) \Pr(\bar{f}(\boldsymbol{z}) = y) \mathrm{d}\boldsymbol{z}$$

Note that for all possible $\bar{f} \in \mathcal{F}$, we can re-assign all the function output within a likelihood region to be *constant* without affecting $\Pr(\bar{f}(\phi(\boldsymbol{x})) = y)$ and $\Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) =$

y). Concretely, we define $\bar{f}'$ as

$$\Pr(\bar{f}'(\boldsymbol{z}') = y) = \frac{\int_{\mathcal{L}_i} \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) \Pr(\bar{f}(\boldsymbol{z}) = y) \mathrm{d}\boldsymbol{z}}{\int_{\mathcal{L}_i} \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) \mathrm{d}\boldsymbol{z}}, \forall \boldsymbol{z}' \in \mathcal{L}_i, \forall i \in [n],$$

then we have

$$\int_{\mathcal{L}_i} \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) \Pr(\bar{f}(\boldsymbol{z}) = y) \mathrm{d}\boldsymbol{z} = \int_{\mathcal{L}_i} \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) \Pr(\bar{f}'(\boldsymbol{z}) = y) \mathrm{d}\boldsymbol{z}$$

Since in $\mathcal{L}_i$, $\Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) / \Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z})$ is constant, we also have

$$\int_{\mathcal{L}_i} \Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z}) \Pr(\bar{f}(\boldsymbol{z}) = y) \mathrm{d}\boldsymbol{z} = \int_{\mathcal{L}_i} \Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z}) \Pr(\bar{f}'(\boldsymbol{z}) = y) \mathrm{d}\boldsymbol{z}$$

Therefore,

$$\begin{aligned} \Pr(\bar{f}(\phi(\boldsymbol{x})) = y) &= \Pr(\bar{f}'(\phi(\boldsymbol{x})) = y), \text{ and} \\ \Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y) &= \Pr(\bar{f}'(\phi(\bar{\boldsymbol{x}})) = y). \end{aligned}$$

Hence, it suffices to consider the following program

$$\begin{aligned} (\mathrm{I}) \triangleq \min_{g:[n]\to[0,1]} & \sum_{i=1}^{n} \int_{\mathcal{L}_i} \Pr(\phi(\bar{\boldsymbol{x}}) = \boldsymbol{z}) g(i) \mathrm{d}\boldsymbol{z}, \\ s.t. \ & \sum_{i=1}^{n} \int_{\mathcal{L}_i} \Pr(\phi(\boldsymbol{x}) = \boldsymbol{z}) g(i) \mathrm{d}\boldsymbol{z} = p, \end{aligned}$$

where the optimum is equivalent to the program

$$\min_{\bar{f}\in\mathcal{F}:\Pr(\bar{f}(\phi(\boldsymbol{x}))=y)=p} \Pr(\bar{f}(\phi(\bar{\boldsymbol{x}})) = y),$$

and the each g corresponds to a solution $\bar{f}$. For example, the f^* in the statement corresponds to the g^* defined as:

$$g^*(i) = \begin{cases} 1, & \text{if } i < H^*, \\ \frac{p - \sum_{i=1}^{H^*-1} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i)}{\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_{H^*})}, & \text{if } i = H^*, \\ 0, & \text{if } i > H^*. \end{cases} \tag{5.17}$$

We may simplify the program as

$$\begin{aligned} (\mathrm{I}) = \min_{g:[n]\to[0,1]} & \sum_{i=1}^{n} \Pr(\phi(\bar{\boldsymbol{x}}) \in \mathcal{L}_i) g(i), \\ s.t. & \sum_{i=1}^{n} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) g(i) = p. \end{aligned}$$

Clearly, if $\eta_i = 0$, all the optimal g will assign $g(i) = 0$; our solution g^* satisfies this property since

$$H^* \triangleq \min_{H \in \{1,\ldots,n\}: \sum_{i=1}^{H} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) \geq p} H \tag{5.18}$$

implies $\eta_{H^*} > 0$ (otherwise, it implies that $\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_{H^*}) = 0$ and leads to a contradiction). Hence, we can ignore the regions with $\eta_i = 0$, assume $\eta_n > 0$, and simplify program (I) again as

$$(\mathrm{I}) = \min_{g:[n]\to[0,1]} \sum_{i=1}^{n} \frac{1}{\eta_i} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) g(\eta_i), \tag{5.19}$$

$$s.t. \sum_{i=1}^{n} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) g(\eta_i) = p. \tag{5.20}$$

It is evident that g^* satisfies the constraint (5.20), and we will prove that any $g \neq g^*$ that satisfies constraint (5.20) cannot be better.

$\forall g : [n] \to [0,1]$, we define $\Delta(i) \triangleq (g^*(i) - g(i))P(\phi(\boldsymbol{x}) \in \mathcal{L}_i)$. Then we have

$$\sum_{i=1}^{n} \frac{1}{\eta_i} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) g(i) = \sum_{i=1}^{n} \frac{1}{\eta_i} \Big[\Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) g^*(i) - \Delta(i) \Big]$$
$$= \sum_{i=1}^{H^*} \frac{1}{\eta_i} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) g^*(i) - \sum_{i=1}^{n} \frac{1}{\eta_i} \Delta(i). \tag{5.21}$$

Note that $\Delta(i) \geq 0$ for $i < H^*$, $\Delta(i) \leq 0$ for $i > H^*$, and $\sum_{i=1}^{n} \Delta(\eta_i) = 0$ due to the constraint (5.20). Therefore, we have

$$\sum_{i=1}^{n} \frac{1}{\eta_i} \Delta(\eta_i) \leq \sum_{i=1}^{n} \frac{1}{\eta_{H^*}} \Delta(\eta_i) = 0. \tag{5.22}$$

Finally, combining (5.21) and (5.22),

$$\sum_{i=1}^{n} \frac{1}{\eta_i} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) g(i) \geq \sum_{i=1}^{H^*} \frac{1}{\eta_i} \Pr(\phi(\boldsymbol{x}) \in \mathcal{L}_i) g^*(i). \tag{5.23}$$

□

5.A.3 Proof of Lemma 5.4

Proof. If $\varphi(\cdot)$ is defined as Eq. (5.6), $\forall \boldsymbol{x}, \bar{\boldsymbol{x}} \in \mathcal{X}$ such that $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_0 = r$, below we show that $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}$ is independent of $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$. Indeed, since $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_0$ is the number of non-zero elements of $\boldsymbol{x} - \bar{\boldsymbol{x}}$, we know there are exactly r dimensions such that $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ do not match. Notice that $\varphi(\cdot)$ applies to each dimension of $\boldsymbol{x}$ independently, so we can safely ignore any correlations between two dimensions. Therefore, by the symmetry of the distribution, we can rearrange the *order* of coordinates, and assume $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ differ for the first r dimensions, and match for the rest $d - r$ dimensions.

Notice that the randomization $\varphi(\cdot)$ has the nice property that the perturbing probabilities are oblivious to the *actual values* of the input. Therefore, by the definition of $\boldsymbol{x}_C, \bar{\boldsymbol{x}}_C$, we know that they are the canonical form of all pairs of $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ such that $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_0 = r$; hence, $\rho_{\boldsymbol{x},\bar{\boldsymbol{x}}}(p)$ is constant and equals $\rho_r(p)$ for every $p \in [0,1]$. □

5.A.4 Proof of Lemma 5.5

Proof. In this proof, we adopt the notation of the canonical form $\boldsymbol{x}_C$ and $\bar{\boldsymbol{x}}_C$ from Appendix 5.A.3.

Recall that $\boldsymbol{x}_C$ is a zero vector, and $\bar{\boldsymbol{x}}_C$ has the first r entries equal to 1 and the last $d-r$ entries equal to 0. We use the likelihood tuple (u,v) to refer the scenario when $\boldsymbol{x}_C$ "flips" u coordinates (the likelihood is $\alpha^{d-u}\beta^u$), and $\bar{\boldsymbol{x}}_C$ "flips" v coordinates (the likelihood is $\alpha^{d-v}\beta^v$). Note that $u \le v$ by assumption. For $r \in [d]$ and $(u,v) \in \{0,1,\dots,d\}^2$, the number of possible outcome $\boldsymbol{z} \in \mathcal{X}$ with the likelihood tuple (u,v) can be computed in the following way:

$$|\mathcal{L}(u,v;r)| = \sum_{i=0}^{\min(u,d-r)} \sum_{j=0}^{u-i} \frac{(K-1)^j \mathbb{I}((u-i-j)+(v-i-j)+j=r)r!}{(u-i-j)!(v-i-j)!j!} \frac{K^i(d-r)!}{(d-r-i)!i!},$$

where the first summation and the term

$$\frac{K^i(d-r)!}{(d-r-i)!i!}$$

correspond to the case where i entries out of the last $(d-r)$ coordinates in $\boldsymbol{x}_C$ and $\bar{\boldsymbol{x}}_C$ are both modified. Notice that $\boldsymbol{x}_C$ and $\bar{\boldsymbol{x}}_C$ are equal in the last $d-r$ dimensions, so if $\boldsymbol{x}_C$ has i entries modified among them, in order to ensure that $\bar{\boldsymbol{x}}_C$ equals $\boldsymbol{x}_C$ after modification, $\bar{\boldsymbol{x}}_C$ should have exactly the same i entries modified as well (in order to become the same $\boldsymbol{z}$ in Eq. (5.7)).

The second summation and the term

$$\frac{(K-1)^j \mathbb{I}((u-i-j)+(v-i-j)+j=r)r!}{(u-i-j)!(v-i-j)!j!}$$

corresponds to the case where j entries out of the first r coordinates of $\boldsymbol{x}_C$ and $\bar{\boldsymbol{x}}_C$ are modified to any values other than $\{0,1\}$, $u-i-j$ entries in $\boldsymbol{x}_C$ are modified to 1, and $v-i-j$ entries in $\bar{\boldsymbol{x}}_C$ are modified to 0. By the same analysis, we know that both $\boldsymbol{x}_C$ and $\bar{\boldsymbol{x}}_C$ should have exactly the same j entries modified to any value other than $\{0,1\}$. The indicator function $\mathbb{I}((u-i-j)+(v-i-j)+j=r)$ simply verifies that whether the value of j is valid. Note that these two summations have covered all possible cases of modifications on $\boldsymbol{x}_C$ and $\bar{\boldsymbol{x}}_C$ in $\mathcal{L}(u, v; r)$.

After fixing the value of i and j, each summand is simply calculating the number of symmetric cases. $(K-1)^j$ means there are j entries modified to $(K-1)$ possible values. $\frac{r!}{(u-i-j)!(v-i-j)!j!}$ is the number of possible configurations for the first r coordinates. K^i means there are i entries momdified to K possible values. $\frac{(d-r)!}{(d-r-1)!i!}$ is the number of possible configurations for the last $d-r$ coordinates.

Now it remains to simplify the expression. Let $j^* \triangleq u + v - 2i - r$, we have

$$
\begin{aligned}
|\mathcal{L}(u, v; r)| &= \sum_{i=0}^{\min(u,d-r)} \frac{\mathbb{I}(j^* \geq 0)\mathbb{I}(j^* \leq u-i)(K-1)^{j^*} r!}{(u-i-j^*)!(v-i-j^*)!j^*!} \frac{K^i (d-r)!}{(d-r-i)!i!}, \\
&= \sum_{i=\max\{0,v-r\}}^{\min(u,d-r)} \frac{\mathbb{I}(j^* \geq 0)(K-1)^{j^*} r!}{(u-i-j^*)!(v-i-j^*)!j^*!} \frac{K^i (d-r)!}{(d-r-i)!i!}, \\
&= \sum_{i=\max\{0,v-r\}}^{\min(u,d-r,\lfloor \frac{u+v-r}{2} \rfloor)} \frac{(K-1)^{j^*} r!}{(u-i-j^*)!(v-i-j^*)!j^*!} \frac{K^i (d-r)!}{(d-r-i)!i!}. \qquad (5.24)
\end{aligned}
$$

Moreover, we know that $|\mathcal{L}(u, v; r)| = |\mathcal{L}(v, u; r)|$ holds by the symmetry between $\boldsymbol{x}_C$ and $\bar{\boldsymbol{x}}_C$. □

5.B Algorithms for Decision Trees

5.B.1 Dynamic Programming for Restricted Decision Trees

Given an input $\boldsymbol{x} \in \mathcal{X}$, we run dynamic programming (Algorithm 2) for computing the certificate, based on the same idea mentioned in Section 5.3.6.

We use $\texttt{adv}[i, r]$ to denote the worst prediction at node i if at most r features can be perturbed. Algorithm 2 uses the following updating rule for $\texttt{adv}[i, r]$.

$$
\begin{aligned}
&\texttt{adv}[i, r] = \min\{ \\
&\min_{\bar{r} \in \{0,1,\dots,r\}} \{\alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=1\}} \beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=0\}} \texttt{adv}[\texttt{right}[i], \bar{r}] + \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=0\}} \beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=1\}} \texttt{adv}[\texttt{left}[i], r - \bar{r}]\}, \\
&\min_{\bar{r} \in \{0,1,\dots,r-1\}} \{\alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=0\}} \beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=1\}} \texttt{adv}[\texttt{right}[i], \bar{r}] + \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=1\}} \beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=0\}} \texttt{adv}[\texttt{left}[i], r - 1 - \bar{r}]\}\}
\end{aligned}
$$

Algorithm 2 DP($\boldsymbol{x}$, i, R)

```
if $i$ is leaf then
    for $r = 1, \cdots, R$ do
        adv$[i, r]$ =Leaf-Output($i$)
    end for
    Return
end if
$rw = \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=1\}} \beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=0\}}$
$lw = \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=0\}} \beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}[i]}=1\}}$
DP($\boldsymbol{x}$, right[i], $R$)
DP($\boldsymbol{x}$, left[i], $R$)
for $r = 1, \cdots, R$ do
    adv$[i, r] = 1$
    for $\bar{r} = 0, \cdots r$ do
        adv$[i, r] = \min\{$adv$[i, r], rw *$ adv[right$[i], \bar{r}] + lw *$ adv[left$[i], r - \bar{r}]\}$
    end for
    for $\bar{r} = 0, \cdots r - 1$ do
        adv$[i, r] = \min\{$adv$[i, r], lw *$ adv[right$[i], \bar{r}] + rw *$ adv[left$[i], r - 1 - \bar{r}]\}$
    end for
end for
```

There are two cases in this updating rule. In the first case, the feature used at node i is not perturbed, so it remains to see if we perturb $\bar{r}$ features in the right subtree and $r - \bar{r}$ features in the left subtree, what is the minimum adversarial prediction if $\bar{r} \in \{0, \cdots, r\}$. In the second case, the feature used at node i is perturbed, and we check if we perturb $r-1$ features in the two subtrees, what is the minimum adversarial prediction. Combining the two cases together, we get the solution for `adv`$[i, r]$.

5.B.2 Training Algorithm for Decision Trees

We consider the randomization scheme introduced in Eq. (5.6): for every coordinate in a given input $\boldsymbol{x}$, we may perturb its value with probability β. After perturbation, $\boldsymbol{x}$ may arrive at any leaf node, rather than following one specific path as in the standard decision tree. Therefore, when training, we maintain the probability of arriving at the current tree node for every input $\boldsymbol{x}$, denoted as `probs`. The probability is multiplied by α or β after each layer, depending on the input and the feature used for the current

Algorithm 3 Train(X,Y, `maxdep`)

```
 1: $Q = [(\texttt{root}, 0, [1, 1, \cdots, 1])]$
 2: while Q not Empty do
 3:     $i, \texttt{dep}, \texttt{probs} = Q.\texttt{pop}()$
 4:     if $\texttt{dep}{=}\texttt{maxdep}$ then
 5:         Assign-Leaf-Node($i, \texttt{probs}$)
 6:         Continue
 7:     end if
 8:     f-list = Get-available-features()
 9:     for idx in f-list do
10:         $\texttt{list} = [(y, \alpha^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}}=1\}}\beta^{\mathbb{I}\{\boldsymbol{x}_{\texttt{idx}}=0\}}\texttt{probs}[\boldsymbol{x}])$ for $(\boldsymbol{x}, y)$ in $X]$
11:         $\texttt{idx}^*$ = Update-best-feature-score $(\texttt{idx}, \texttt{list}, \texttt{idx}^*)$
12:     end for
13:     $\texttt{idx}[i]{=}\ \texttt{idx}^*$
14:     left-probs=probs
15:     right-probs=probs
16:     for $\boldsymbol{x}$ in $X$ do
17:         if $\boldsymbol{x}_{\texttt{idx}} = 1$ then
18:             $\texttt{left-probs}[\boldsymbol{x}] = \texttt{left-probs}[\boldsymbol{x}] * \beta$
19:             $\texttt{right-probs}[\boldsymbol{x}] = \texttt{right-probs}[\boldsymbol{x}] * \alpha$
20:         else
21:             $\texttt{right-probs}[\boldsymbol{x}] = \texttt{right-probs}[\boldsymbol{x}] * \beta$
22:             $\texttt{left-probs}[\boldsymbol{x}] = \texttt{left-probs}[\boldsymbol{x}] * \alpha$
23:         end if
24:     end for
25:     $Q.\texttt{push}(\texttt{left}[i], \texttt{dep} + 1, \texttt{left-probs})$
26:     $Q.\texttt{push}(\texttt{right}[i], \texttt{dep} + 1, \texttt{right-probs})$
27: end while
```

tree node. See Algorithm 3 for details.

The overall framework of Algorithm 3 is standard: we train the tree nodes greedily in breadth-first ordering, and pick the best splitting feature every time. However, when picking the best splitting feature, the standard decision tree uses Gini impurity based on all the remaining training data that follow the path from the root to the current tree node. In our algorithm, this includes all the training data, but with different arriving probabilities. Therefore, we apply the weighted Gini impurity metric instead. Specifically, for a split, its weighted Gini impurity is (after `probs` is updated with `idx`):

$$1 - \left(\frac{\sum_{\boldsymbol{x}\in\mathcal{X},y=1} \texttt{probs}[\boldsymbol{x}]}{\sum_{\boldsymbol{x}\in\mathcal{X}} \texttt{probs}[\boldsymbol{x}]}\right)^2 - \left(\frac{\sum_{\boldsymbol{x}\in\mathcal{X},y=0} \texttt{probs}[\boldsymbol{x}]}{\sum_{\boldsymbol{x}\in\mathcal{X}} \texttt{probs}[\boldsymbol{x}]}\right)^2$$

When the arriving probability for each $\boldsymbol{x}$ is restricted to be either 0 or 1, this definition becomes the standard Gini impurity.

5.C Experimental Details

5.C.1 Supplementary Materials for the MNIST Experiment

For the MNIST experiment, we use a simple 4-layer convolutional network, where the first two layers are convolutional layers, and the last two layers are feedforward layers. For the two convolutional layers, we use kernel size 5, and output channels 20 and 50, respectively. For the two feedforward layers, we use 500 hidden nodes. We tune the hyperparameter $\alpha \in \{0.72, 0.76, 0.80, 0.84, 0.88, 0.92, 0.96\}$ for $\mu(R)$ in the validation set using the certificates from the Gaussian distribution [29]. The resulting α is 0.8. We also train a CNN with an isotropic Gaussian with the σ that corresponds to $\alpha = 0.8$ (see §5.3.5).

The learning procedure for all the models are the same (except the distribution). The batch size is 400. We train each model for 30 epochs with the SGD optimizer with Nesterov momentum (momentum = 0.9). The learning rate is initially set to be 0.05 and annealed by a factor of 10 for every 10 epochs of training. The models are implemented in `PyTorch` [113], and run on a GPU with 12G memory.

5.C.2 Supplementary Materials for the ImageNet Experiment

We use the `PyTorch` [113] implementation provided by Cohen et al. [29] as the backbone and implement our algorithm based on their pipeline. Thus, the training details are consistent to the one reported in their paper except that we use a different distribution. Here, we summarize some important details. ResNet-50 is used as the base classifier for our ImageNet experiment, whose architecture is provided

Table 5.C.1: The guaranteed accuracy for different α of ResNet50 models smoothed by the discrete distribution on ImageNet.

α value	ACC@r							
	$r=0$	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$	$r=7$
0.5	0.666	0.412	0.388	0.000	0.000	0.000	0.000	0.000
0.4	0.650	0.538	0.356	0.338	0.000	0.000	0.000	0.000
0.3	0.592	0.516	0.314	0.300	0.274	0.234	0.000	0.000
0.2	0.524	0.448	0.394	0.270	0.218	0.212	0.190	0.176
0.1	0.350	0.314	0.282	0.248	0.212	0.182	0.150	0.100

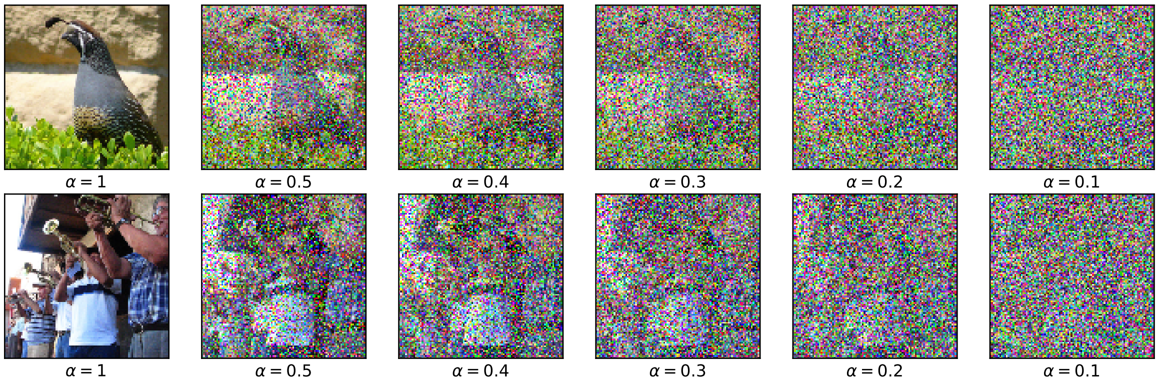


Figure 5.C.1: ImageNet images corrupted by varying levels of the discrete noise.

in `torchvision`. After the randomization is done, we normalize each image by subtracting the dataset mean (0.485, 0.456, 0.406) and dividing by the standard deviation (0.229, 0.224, 0.225). Parameters are optimized by SGD with momentum set as 0.9. The learning rate is initially set to be 0.1 and annealed by a factor of 10 for every 30 epochs of training. The total number of training epochs is 90. The batch size is 300, parallelized across 2 GPUs. We tune $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ for the discrete distribution, and measure the performance in ACC@r, compared to the classifier under an additive isotropic Gaussian noise [29].[5] The samples of the randomized image for each α are visualized in Figure 5.C.1. We follow the prior work [29] to evaluate every 100th image in the validation set. The detailed accuracy numbers of our approach under different α and r are available in Table 5.C.1.

[5]We run their released model from `https://github.com/locuslab/smoothing`.

5.C.3 Supplementary Materials for the Chemical Property Prediction Experiment

The dataset contains $1,513$ molecules (data points). We split the data into the training, validation, and testing sets with the ratio 0.8, 0,1, and 0,1, respectively. Following common practice in chemical property prediction [158], the splitting is done based on the Bemis-Murcko scaffold [12]; the molecules within a split are inside different scaffolds from the other splits. We refer the details of scaffold splitting to [158]. We observe similar experiment results when we use a random split.

For both the decision tree and the randomly smoothed decision tree, we tune the depth limit in $\{6, 7, 8, 9, 10\}$. For the randomly smoothed decision tree, we also tune $\alpha \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$. The tuned α is 0.8, and the tuned depth limits are 10 for both models.

5.C.4 Computing Adversarial AUC

Assume that there are $n + m$ data points, n of them are positive instances, denoted as $A \triangleq \{x_1, \cdots, x_n\}$, and m of them are negative instances, denoted $B \triangleq \{x_{n+1}, \cdots, x_{n+m}\}$. Denote the whole dataset as $X \triangleq A \cup B$. For data point $x \in X$, we may adversarially perturb x up to the perturbation radius r, denoted as x^r. Note that, since $\mathcal{Y}$ is binary, maximizing the probability for predicting one class can be equivalently done by minimizing the probability for predicting the other class. Hence, we may use Algorithm 2 to find the adversaries for both the positive and negative examples. Below we use the prediction probability for the class 1 as the score. Denote the score of x and x^r as $s(x)$ and $s(x^r)$. For $x \in A$, we know that $s(x) \geq s(x^r)$, and for $x \in B$, $s(x) \leq s(x^r)$.

If we are only allowed to perturb $k < n + m$ data points, to minimize AUC, we aim to solve the following program:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m}\Big[a_i b_j \hat{\mathbb{I}}(s(x_i^r), s(x_{j+n}^r)) + a_i(1-b_j)\hat{\mathbb{I}}(s(x_i^r), s(x_{j+n})) \\
& + (1-a_i)b_j \hat{\mathbb{I}}(s(x_i), s(x_{j+n}^r)) + (1-a_i)(1-b_j)\hat{\mathbb{I}}(s(x_i), s(x_{j+n}))\Big] \\
\text{subject to} \quad & \sum_{i\in[n]} a_i + \sum_{j\in[m]} b_j \leq k, \\
& a_i \in \{0,1\}, \quad i = 1, ..., n \\
& b_j \in \{0,1\}, \quad j = 1, ..., m
\end{aligned}
$$

We may use standard mixed-integer programming solvers like Gurobi to solve the program. Here we use a_i to denote whether data point $x_i \in A$ is perturbed, and b_j to denote whether data point $x_{j+n} \in B$ is perturbed. The function $\hat{\mathbb{I}}(x, x')$ is an indicator function defined as

$$
\hat{\mathbb{I}}(x, x') \triangleq \begin{cases} 1, & \text{if } x > x', \\ 0.5, & \text{if } x = x', \\ 0, & \text{if } x < x'. \end{cases} \tag{5.25}
$$

Chapter 6

Discussion

This thesis presents a series of methods for building models that are transparent yet expressive for complex tasks. For instance, we improved the capacity of existing models by enforcing transparency only locally (Chapter 2 and 3). We also generalized simple interpretable models by extending coordinate-wise cuts in decision trees to linear classifications (Chapter 4). Finally, focusing on robustness, we made black-box classifiers provably robust over discrete spaces via a simple smoothing technique (Chapter 5).

The methods in Chapter 2, 3, 4, and 5 realize locally transparent mappings, locally linear mappings, locally constant mappings, and locally constant classifiers, respectively. These functional perspectives provide an alternative view of this thesis as a hierarchical investigation of local model classes. Nevertheless, useful properties in machine learning models clearly extend beyond local properties. Below we discuss some limitations and potential extensions of the methods developed in this thesis.

The game-theoretic approach in Chapter 2 requires an interpretable definition of local neighborhoods. Sometimes such neighborhoods are easy to construct based on existing domain knowledge, but not always. For example, in image classifications, we may need the neighborhoods to reflect visual similarities rather than, e.g., generic ℓ_p distances. In this case, we could first seek some embeddings that capture visual similarities (akin to concept activation vectors [71]), and then construct the neighborhoods based on the embedding space.

Local models developed in both Chapter 2 and 3 are not specifically controlled across different local regions, making it possible for local models to differ even for neighboring regions. In order to ensure that boundary cases are not treated very differently if neighborhoods are constructed in a slightly modified way, stability of neighboring local models is necessary. Solving this problem will draw on robustness and fairness, further contributing to transparency.

As briefly discussed in Chapter 4, the extension of locally constant networks to a boosting framework empirically outperforms gradient boosted decision trees. Performance is, however, only comparable to random forests. Developing a randomized ensemble method for locally constant networks akin to random forest could lead to potential improvements in practical performance. The theoretical analyses can also be extended. For instance, we utilized dummy decision nodes to simplify the statements, so the precise relationship between an oblique decision tree and the equivalent locally constant networks was not fully characterized.

The smoothing mechanism for constructing robust classifiers introduced in Chapter 5 can be extended to more structured settings. For example, instead of simple additive noise, smoothing mechanism may involve transformation of images. Studying how structured smoothing may impact the guarantees is an important future direction. On the other hand, we have shown in the decision tree example that the certificates can be tightened by taking the functional form of the classifier into account. Similar extensions to other classifiers are possible but potentially technically challenging.

While this thesis is motivated by interpretability, we explicitly separated transparency from (human) interpretability. Mathematical statements that are at the heart of transparency do not necessarily imply that they are understandable by end users. Better connecting these two lines of research would be really helpful for practical AI systems.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Maruan Al-Shedivat, Avinava Dubey, and Eric P Xing. Contextual explanation networks. *arXiv preprint arXiv:1705.10301*, 2017.

[3] David Alvarez-Melis and Tommi S Jaakkola. A causal framework for explaining the predictions of black-box sequence-to-sequence models. *Empirical Methods in Natural Language Processing*, 2017.

[4] David Alvarez-Melis and Tommi S. Jaakkola. On the robustness of interpretability methods. *Workshop on Human Interpretability in Machine Learning at ICML 2018; arXiv preprint arXiv:1806.08049*, 2018.

[5] David Alvarez-Melis and Tommi S. Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems*, 2018.

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, 2017.

[7] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, pages 1120–1128, 2016.

[8] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. " What is relevant in a text document?": An interpretable machine learning approach. *PloS one*, 2017.

[9] Randall Balestriero and Richard Baraniuk. Mad max: Affine spline insights into deep learning. *arXiv preprint arXiv:1805.06576v5*, 2018.

[10] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25, 2006.

[11] Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The cramer distance as a solution to biased wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017.

[12] Guy W Bemis and Mark A Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of medicinal chemistry*, 39(15):2887–2893, 1996.

[13] Kristin P Bennett. Global tree optimization: A non-greedy decision tree algorithm. *Computing Science and Statistics*, pages 156–156, 1994.

[14] Kristin P Bennett and Ayhan Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information processing systems*, 1999.

[15] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[16] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[17] Aleksandar Bojchevski, Johannes Klicpera, and Stephan Günnemann. Efficient robustness certificates for discrete data: Sparsity-aware randomized smoothing for graphs, images and more. In *International Conference on Machine Learning*, 2020.

[18] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[19] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[20] Richard P Brent. Fast training algorithms for multilayer neural nets. *IEEE Transactions on Neural Networks*, 2(3):346–354, 1991.

[21] Xiaoyu Cao and Neil Zhenqiang Gong. Mitigating evasion attacks to deep neural networks via region-based classification. In *Proceedings of the 33rd Annual Computer Security Applications Conference.* ACM, 2017.

[22] Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Provably minimally-distorted adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

[23] Miguel A Carreira-Perpinán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In *Advances in Neural Information Processing Systems*, pages 1211–1221, 2018.

[24] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017.

[25] Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big data*, 5(2):153–163, 2017.

[26] Krzysztof J Cios and Ning Liu. A machine learning method for generation of a neural network architecture: A continuous id3 algorithm. *IEEE Transactions on Neural Networks*, 3(2):280–291, 1992.

[27] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. *arXiv preprint arXiv:1704.08847*, 2017.

[28] Charles J Clopper and Egon S Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 1934.

[29] Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, 2019.

[30] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 1995.

[31] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of relu networks via maximization of linear regions. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

[32] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pages 2702–2711, 2016.

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE international conference on computer vision*. Ieee, 2009.

[34] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[35] Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. In *Advances in Neural Information Processing Systems*, pages 3603–3613, 2019.

[36] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.

[37] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pages 2224–2232, 2015.

[38] Krishnamurthy Dvijotham, Sven Gowal, Robert Stanforth, Relja Arandjelovic, Brendan O'Donoghue, Jonathan Uesato, and Pushmeet Kohli. Training verified learners with learned verifiers. *arXiv preprint arXiv:1805.10265*, 2018.

[39] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *the 34th Annual Conference on Uncertainty in Artificial Intelligence*, 2018.

[40] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[41] Gamaleldin Elsayed, Dilip Krishnan, Hossein Mobahi, Kevin Regan, and Samy Bengio. Large margin deep networks for classification. In *Advances in Neural Information Processing Systems*, 2018.

[42] Chris Finlay, Aram-Alexandre Pooladian, and Adam M Oberman. The logbarrier adversarial attack: making effective use of decision boundary information. *arXiv preprint arXiv:1903.10396*, 2019.

[43] Matteo Fischetti and Jason Jo. Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. *arXiv preprint arXiv:1712.06174*, 2017.

[44] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23:296–309, 2018.

[45] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[46] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[47] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 2016.

[48] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. *AAAI Conference on Artificial Intelligence*, 2019.

[49] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[50] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[51] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.

[52] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[53] Ed Griffen, Andrew G Leach, Graeme R Robb, and Daniel J Warner. Matched molecular pairs as a medicinal chemistry tool: miniperspective. *Journal of medicinal chemistry*, 54(22):7739–7750, 2011.

[54] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[55] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, 2017.

[56] Moritz Hardt, Eric Price, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *Advances in Neural Information Processing Systems*, 2016.

[57] Hao He, Hao Wang, Guang-He Lee, and Yonglong Tian. Probgan: Towards probabilistic gan with theoretical guarantees. In *International Conference on Learning Representations*, 2019.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[60] Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. *International Conference on Machine Learning*, 2018.

[61] Gaurush Hiranandani, Harikrishna Narasimhan, and Oluwasanmi Koyejo. Fair performance metric elicitation. In *Advances in Neural Information Processing Systems*, 2020.

[62] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[63] Chen-Yu Hsu, Rumen Hristov, Guang-He Lee, Mingmin Zhao, and Dina Katabi. Enabling identification and behavioral sensing in homes using radio reflections. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[64] Chen-Yu Hsu, Abbas Zeitoun, Guang-He Lee, Dina Katabi, and Tommi Jaakkola. Self-supervised learning of appliance usage. In *International Conference on Learning Representations*, 2020.

[65] Hsiang Hsu, Shahab Asoodeh, and Flavio Calmon. Obfuscation via information density estimation. In *International Conference on Artificial Intelligence and Statistics*, pages 906–917. PMLR, 2020.

[66] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[67] Po-Sen Huang, Robert Stanforth, Johannes Welbl, Chris Dyer, Dani Yogatama, Sven Gowal, Krishnamurthy Dvijotham, and Pushmeet Kohli. Achieving verified robustness to symbol substitutions via interval bound propagation. *arXiv preprint arXiv:1909.01492*, 2019.

[68] Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. Certified robustness to adversarial word substitutions. *arXiv preprint arXiv:1909.00986*, 2019.

[69] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, 2018.

[70] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[71] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pages 2668–2677. PMLR, 2018.

[72] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[73] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

[74] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent tradeoffs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*, 2016.

[75] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, 2017.

[76] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, 2015.

[77] Jelena Kovačević, Amina Chebira, et al. An introduction to frames. *Foundations and Trends® in Signal Processing*, 2008.

[78] Isaac Lage, Andrew Slavin Ross, Been Kim, Samuel J Gershman, and Finale Doshi-Velez. Human-in-the-loop interpretability prior. In *Advances in Neural Information Processing Systems*, 2018.

[79] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *International Conference on Knowledge Discovery and Data Mining*, 2016.

[80] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[81] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting Structured Data*, 1(0), 2006.

[82] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. *IEEE Symposium on Security and Privacy (SP)*, 2019.

[83] Guang-He Lee, David Alvarez-Melis, and Tommi S. Jaakkola. Towards robust, locally linear deep networks. In *International Conference on Learning Representations*, 2019.

[84] Guang-He Lee and Tommi S. Jaakkola. Oblique decision trees from derivatives of relu networks. In *International Conference on Learning Representations*, 2020.

[85] Guang-He Lee, Wengong Jin, David Alvarez-Melis, and Tommi S. Jaakkola. Functional transparency for structured data: a game-theoretic approach. In *International Conference on Machine Learning*, 2019.

[86] Guang-He Lee, Yang Yuan, Shiyu Chang, and Tommi Jaakkola. Tight certificates of adversarial robustness for randomly smoothed classifiers. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[87] J. Lee, H. Qiu, G. Yu, J. Lin, and University of Cincinnati. Rexnord Technical Services (2007). IMS. Bearing data set. *NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/project/prognostic-data-repository), NASA Ames Research Center, Moffett Field, CA*, 7(8), 2016.

[88] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing Neural Predictions. In *Empirical Methods in Natural Language Processing*, 2016.

[89] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. In *International Conference on Machine Learning*, 2017.

[90] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Second-order adversarial attack and certifiable robustness. *arXiv preprint arXiv:1809.03113*, 2018.

[91] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[92] Tania Lombrozo. The structure and function of explanations. *Trends in cognitive sciences*, 10(10):464–470, 2006.

[93] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

[94] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 30, page 3, 2013.

[95] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[96] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[97] Bjoern H Menze, B Michael Kelm, Daniel N Splitthoff, Ullrich Koethe, and Fred A Hamprecht. On oblique random forests. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 453–469. Springer, 2011.

[98] John Miller and Moritz Hardt. When recurrent models don't need to be recurrent. *arXiv preprint arXiv:1805.10369*, 2018.

[99] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, 2018.

[100] Guido Montúfar. Notes on the number of linear regions of deep neural networks. 2017.

[101] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, 2014.

[102] Youssef Mroueh, Chun-Liang Li, Tom Sercu, Anant Raj, and Yu Cheng. Sobolev gan. *International Conference on Learning Representations*, 2018.

[103] Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

[104] Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[105] Sreerama K Murthy, Simon Kasif, Steven Salzberg, and Richard Beigel. Oc1: A randomized algorithm for building oblique decision trees. In *AAAI Conference on Artificial Intelligence*, volume 93, pages 322–327. Citeseer, 1993.

[106] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pages 807–814, 2010.

[107] Arkadi Nemirovski. Prox-method with rate of convergence o (1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

[108] Jerzy Neyman and Egon Sharpe Pearson. Ix. on the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 1933.

[109] Mohammad Norouzi, Maxwell Collins, Matthew A Johnson, David J Fleet, and Pushmeet Kohli. Efficient non-greedy optimization of decision trees. In *Advances in Neural Information Processing Systems*, pages 1729–1737, 2015.

[110] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016.

[111] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.

[112] The European Parliament and The European Council. General data protection regulation. In *Off. J. Eur. Union*, 2014.October 1995 (2016).

[113] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[114] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, pages 2825–2830, 2011.

[115] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[116] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *International Conference on Machine Learning*, pages 2847–2854, 2017.

[117] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

[118] Aditi Raghunathan, Jacob Steinhardt, and Percy S Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pages 10877–10887, 2018.

[119] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[120] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[121] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

[122] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

[123] Elan Rosenfeld, Ezra Winston, Pradeep Ravikumar, and Zico Kolter. Certified robustness to label-flipping attacks via randomized smoothing. In *International Conference on Machine Learning*, 2020.

[124] Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker. Towards verification of artificial neural networks. In *MBMV*, pages 30–40, 2015.

[125] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *https://arxiv.org/abs/1610.02391 v3*, 7(8), 2016.

[126] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. *International Conference on Machine Learning*, 2018.

[127] Ishwar Krishnan Sethi. Entropy nets: from decision trees to neural networks. *IEEE*, 78(10):1605–1613, 1990.

[128] Ohad Shamir. The sample complexity of learning linear predictors with the squared loss. *Journal of Machine Learning Research*, pages 3475–3486, 2015.

[129] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[130] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, 2018.

[131] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. In *International Conference on Learning Representations*, 2018.

[132] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[133] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, pages 11895–11907, 2019.

[134] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[135] Suraj Srinivas and Francois Fleuret. Knowledge transfer with Jacobian matching. In *International Conference on Machine Learning*, pages 4723–4731, 2018.

[136] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[137] Teague Sterling and John J Irwin. Zinc 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

[138] Govindan Subramanian, Bharath Ramsundar, Vijay Pande, and Rajiah Aldrin Denny. Computational modeling of β-secretase 1 (bace-1) inhibitors using ligand based approaches. *Journal of chemical information and modeling*, 2016.

[139] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328. PMLR, 2017.

[140] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[141] Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *International Conference on Machine Learning*, 2019.

[142] Yonglong Tian, Guang-He Lee, Hao He, Chen-Yu Hsu, and Dina Katabi. Rf-based fall monitoring using convolutional neural networks. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2018.

[143] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2017.

[144] KD Tocher. Extension of the neyman-pearson theory of tests to discontinuous variates. *Biometrika*, 1950.

[145] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

[146] Vladimir N. Vapnik. Estimation of dependences based on empirical data. 1982. *NY: Springer-Verlag*, 1995.

[147] Vladimir N. Vapnik and A. Sterin. On structural risk minimization or overall risk in a problem of pattern recognition. *Automation and Remote Control*, 1977.

[148] Jennifer Wortman Vaughan and Hanna Wallach. A human-centered agenda for intelligible machine learning. *Machines We Trust: Getting Along with Artificial Intelligence*, 2020.

[149] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[150] Tong Wang. Gaining free or low-cost interpretability with interpretable partial substitute. In *International Conference on Machine Learning*. PMLR, 2019.

[151] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *International Conference on Machine Learning*, 2018.

[152] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 1994.

[153] DC Wickramarachchi, BL Robertson, Marco Reale, CJ Price, and J Brown. Hhcart: An oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016.

[154] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *the 35th International Conference on Machine Learning*, 2018.

[155] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5283–5292, 2018.

[156] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems*, 2018.

[157] Mike Wu, Michael C. Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *AAAI Conference on Artificial Intelligence*, 2018.

[158] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

[159] Ziang Yan, Yiwen Guo, and Changshui Zhang. Deep defense: Training dnns with improved adversarial robustness. In *Advances in Neural Information Processing Systems*, 2018.

[160] Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

[161] Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. In *Advances in Neural Information Processing Systems*, 2018.

[162] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[163] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, 2018.

[164] Mingmin Zhao, Shichao Yue, Dina Katabi, Tommi S Jaakkola, and Matt T Bianchi. Learning sleep stages from radio signals: a conditional adversarial architecture. In *International Conference on Machine Learning*, 2017.