Machine Learning Algorithms and Applications in Health Care

by

Matthew Sobiesk

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2022

Author .....
Sloan School of Management
September 6, 2021

Certified by .....
Dimitris Bertsimas
Boeing Professor of Operations Research
Thesis Supervisor

Accepted by .....
Georgia Perakis
William F. Pounds Professor of Management Science
Co-director, Operations Research Center

Machine Learning Algorithms and Applications in Health Care

by

Matthew Sobiesk

Submitted to the Sloan School of Management
on September 6, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

There have been many recent advances in machine learning, resulting in models which have had major impact in a variety of disciplines. Some of the best performing models are black boxes, which are not directly interpretable by humans. However, in some applications such as health care it is vital to use interpretable models to understand why the model is making its predictions, to ensure that using them to inform decision making will not unexpectedly harm the people it should instead be helping. This leads to the question of whether a trade off between predictive accuracy and interpretability exists, and how we can improve interpretable models' performances to reduce such trade offs if they do.

In the first chapter, we show that optimal decision trees are equivalent in terms of modeling power to neural networks. Specifically, given a neural network (feedforward, convolutional, or recurrent), we construct a decision tree with hyperplane splits that has identical in-sample performance. Building on previous research showing that given a decision tree, we can construct a feedforward neural network with the same in-sample performance, we prove the two methods are equivalent. We further compare decision trees and neural networks empirically on data from [31] and find that they have comparable performance.

In the second chapter, we propose a new machine learning method called Optimal Predictive Clustering (OPC). The method uses optimization with strong warm starts to simultaneously cluster data points and learn cluster-specific logistic regression models. It is designed to combine strong predictive performance, scalability, and interpretability. We then empirically compare OPC to a wide variety of other

methods such as Optimal Regression Trees with Linear Predictors (ORT-L) and XGBoost. We find that our method performs on par with cutting edge interpretable methods, and that it enhances an ensemble of methods to achieve the best out-of-sample performance across all models.

In the third chapter, we predict one year transplant outcomes for lung, liver, and kidney data to investigate whether predicted post-transplant outcomes should be included in the organ allocation system of organs other than lungs. We find that the models do not differentiate one-year graft survival or failure outcomes effectively enough to be useful components of the organ allocation process. We then theorize about possible reasons for this failure, including the actual transplant procedure having a large effect on the one-year graft outcome or the potential need for additional data, like genetic information.

Thesis Supervisor: Dimitris Bertsimas
Title: Boeing Professor of Operations Research

Acknowledgments

I would like to start by thanking my advisor, Dimitris Bertsimas, for his mentorship, encouragement, and support over the course of my studies. As an undergraduate at Cornell, a professor told me that I should seize any chance I could to work with Dimitris, and now I understand why. His passion, intellect, and drive to make the world a better place pushed me to be a stronger researcher and inspired my passion for health care analytics. I will always be grateful for the opportunity to have studied with him.

My thesis committee members, Rahul Mazumder and Nikos Trichakis, have also been excellent mentors and collaborators. Both of them are exceptional researchers, and it has been a joy to work with them. Their insightful comments and feedback have also been enormously helpful in improving this manuscript.

I am grateful to many more professors at the Operations Research center as well. I would like to thank Rob Freund for being part of my general exam committee and providing valuable feedback on the research I presented there, Georgia Perakis for being a great advisor in planning the ORC IAP Seminar and bringing warmth and joy to the ORC as a co-director, Patrick Jaillet for being a perpetually friendly and comforting presence as a co-director as well. I would also like to thank everyone I have worked alongside as a teaching assistant that I have not previously mentioned, including Amr Farahat, Carine Simon, Martin Copenhaver, Jónas Jónasson, Anne Quaadgras, Jordan Levine, and Michelle Li. It has been a pleasure.

I would also like to thank all of the people I have collaborated with in my research. Yuchen Wang was both very insighful and a joy to work with. Agni Orfanoudaki, Holly Wiberg, and Lea Kapelevich all taught me much during our SwissRe project, and were inspirations. Dr. Parsia Vagefi taught me much about transplantation prob-

lems, and Basmah Safdar, Gary Desir, and Richard Taylor all provided invaluable insights in my research into predicting COVID-19 outcomes. I'm also grateful to Ted Papalexopoulos and Jack Dunn for being available to discuss research quandaries, brainstorm problem solutions, and just give thoughts and feedback on research ideas in general.

I would like to thank everyone I was a teaching assistant with, including Hari Bandi, Jean Pauphilet,Isabelle Bensimon, Kara Keelley, Barry Brundy, Jourdain Lamperski, Michael Beeler, Yee Sian Ng, Michael Li, Vasileios Digalakis, and Zhen Lin. I'd also like to thank my fellow collaborators for the 2020 ORC IAP seminar Jessamyn Liu and Galit Lukin – they were a pleasure to work with.

I also want to thank my other friends at the ORC, including Arthur Delarue, Jackie Baek, Will Ma, Emma Gibson, Emily Meigs, Leann Thayaparan, Kayla Cummings, Rebecca Zhang, Sharon Xu, Peter Cohen, Xiaoyue Gong, Patricio Foncea,Andy Zheng, Nihal Koduri, Nick Renegar, Elisabeth Paulson, Julia Yan, Kevin Zhang, Jordan Lamperski, Velibor Mišić, Martin Copenhaver, Jack Dunn, Daisy Zhao, Colin Pawlowski, Nishanth Mundru, Brad Sturt, Hussein Hazimeh, and Daisy Zhao. All of you have motivated me to become a better researcher with your incredible minds and brightened my day just by being wonderful people.

I would also like to thank other friends from my time at MIT, including members of the MIT Gymnastics team, Qi Yang, Zhao Jinglong, Vishal Patil, Julie Tagaki, Rohan Chitnis, Fred Koehler, Elaine McVay, Derek Leung, Andrea Carney, and Stephanie Chen.

I would also like to thank my teachers and professors I collaborated with, in particular David Ruppert, Chris Arney, and Jamol Pender. I had my first research experiences with David and Chris, and I am so grateful for my time with them. I am also especially grateful to Jamol Pender for convincing me to apply to MIT in

the first place – without that push, none of this would have been possible.

Finally, I want to thank my family for their endless love and support. Throughout my years of graduate school I have laughed, cried, and strategized with them, and they have been with me every step of the way. I truly could not have done this without their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The development of powerful training methodologies like gradient boosting and stochastic gradient descent combined with increased computational power has resulted in a new age of high performing machine learning methods such as neural networks and XGBoost ([72],[55],[78],[24]). These methods have performed extremely well across a variety of disciplines ([40],[45],[65],[25],[5],[93],[68],[87]), but lack interpretability. However, in a variety of applications such as health care, interpretability is a crucial requirement for models, in order to build trust and ensure that they do not learn spurious connections from the data that could result in dangerous consequences such as incorrect treatments being prescribed to patients. This raises several important questions. Is there a tradeoff between having having models with high predictive performance versus models that are interpretable? And if there is such a tradeoff, how significant is it? This thesis is concerned with both theoretical and empirical comparisons of how black-box and interpretable methods perform relative to each

other, and the application of such methods in the health care domain. It is outlined as follows.

Chapters 2 and 3 both focus on the investigation of interpretable methods and comparisons of their performance to that of black box models. Chapter 2 begins with a theoretical comparison of decision trees and neural networks. Since decision trees are one of the most interpretable and widely used machine learning methods, understanding their performance relative to neural networks provides useful information on the interpretability-performance trade-off, and helps identify areas where interpretable methods can be used without a major loss in predictive ability. Through constructive proofs we show how to build decision trees with hyperplane splits that have equivalent performance to neural networks, followed by an empirical comparison of decision trees with neural networks across a variety of domains.

In Chapter 3, we propose a novel interpretable machine learning technique called Optimal Predictive Clustering (OPC). This method uses mixed integer optimization with strong warm starts to simultaneously cluster data points and learn cluster-specific logistic regression models, and is designed to achieve performance at the same level as cutting edge predictive methods like Optimal Regression Trees with Lasso Predictors (ORL-L) while being faster to train. We empirically compare both the predictive performance, training speed, and interpretability of the method to a wide variety of others, including both interpretable methods like ORT-L and model trees and black box methods like XGBoost. We further investigate the benefits of including OPC models in ensembles alongside the other methods, to see how they can improve upon the state-of-the-art for predictions overall.

In Chapter 4 we predict one year graft survival for transplant recipients. With the Organ Procurement and Transplantation Network planning on instituting "more equitable system of allocating deceased donor organs," there will likely be renewed

debate about whether predicted port-transplant outcomes should be included in the organ allocation system of organs other than lungs. We therefore analyze liver, kidney, and lung transplant recipient-donor pairs to understand how well these post-transplant outcomes can be predicted, and how useful such predictions would be as part of an organ allocation system.

Having performed theoretical comparisons of interpretable and black box models in Chapter 2, a methodological proposal for a new interpretable model in Chapter 3, and empirical comparisons in Chapters 2-4, we offer concluding remarks in Chapter 5.

1.2 Main Contributions

Our contributions in this thesis can be summarized as follows, listed by chapter.

Chapter 2: The equivalence of neural networks and optimal decision trees

In this chapter, we investigate the modeling power of neural networks in comparison with optimal classification and regression trees with hyperplanes (OCT-Hs and ORT-Hs). We show that a variety of neural networks (feedforward, convolutional and recurrent) can be transformed to decision trees with hyperplanes with the same accuracy in the training set, showing that OCT-Hs and ORT-Hs are at least as powerful as neural networks. Combined with a result from Sethi ([80]), we show that OCT-Hs and neural networks are equivalent in terms of modeling power. We further compared the performance of optimal classification trees and neural networks empirically. Our contributions include:

- A unique constructive proof that explains how given a neural network one can

build an equivalent decision tree with hyperplane splits, linking two of the most popular and widely utilized machine learning methods more deeply than their categorization as universal approximators and shedding new light on their strengths and weaknesses.

- Given that our construction of trees that emulate a given neural network necessitates the construction of deep trees, we further compared the performance of shallower OCT-Hs and FNNs on twelve well known data sets, and find that the two methods have very similar performances even without theoretical guarantees of equivalence. In 11 out of the 12 data sets the equivalent FNNs and the OCT-H have very similar accuracy. Moreover, in these data sets OCT and OCT-H also have very similar accuracy. In seven out of twelve data sets, the trees perform as well as much larger neural networks. This indicates that there is indeed merit to using OCT-Hs in practice.

- Given that OCT-Hs have an edge in interpretability compared to neural networks, without loss of modeling power, decision trees are therefore arguably the method of choice in applications where interpretability is key.

Chapter 3: Optimal Predictive Clustering

In this chapter we combine clustering algorithms and linear regression to propose Optimal Predictive Clustering (OPC). By solving a mixed-integer optimization problem utilizing strong warm starts, we are able to find near-optimal clusters at high speeds while learning effective cluster-specific regression models. We also cluster points using both $\mathbf{X}$ and $\mathbf{y}$ values, which allows the clusters we find to be better suited for prediction tasks. We compare OPC, Lasso regression, Classification and Regression Trees (CART), Optimal Regression Trees (ORT, ORT-L), XGBoost,

Clus, and Model Trees on 20 real-world datasets and show that OPC achieves strong performance in the following ways:

- **Performance**: We demonstrate that OPC increases out-of-sample R^2 by 0.019 (3.11%) on average compared to ORT-L on 20 datasets from UCI and LIACC Machine Learning Repository ([30],[96]). Furthermore, OPC increases out-of-sample R^2 by 0.134 (27.21% improvement) on average over Lasso regression and by 0.059 (10.37% improvement) on average over Clus on the same 20 datasets. An ensemble built from all the methods also increases out-of-sample R^2 by 0.007 (1.10% improvement) on average over Model Trees and by 0.007 (1.10% improvement) on average over XGBoost.

- **Scalability**: We show the average time to solve the first version of OPC is about 800 seconds on the same 20 datasets, which is $\sim$30$\times$ faster than ORT-L. Furthermore, the maximum time to solve large scale problems ($n \sim 70000$, $p \sim 30$) is under 2000 seconds.

- **Interpretability**: We show how to use the centroids the proposed method finds to create profiles of points in a given cluster and understand why it makes the predictions it does. We further compare OPC with ORT-L in two real world datasets, showing there is no significant loss in interpretability when using the proposed method.

Chapter 4: On the Limitations of Predicting Post Transplant Outcomes

In this chapter we train cutting edge machine learning methods on the UNOS liver, lung, and kidney transplantation data to predict whether a organ will still be functioning in one year or not. Since the Organ Procurement and Transplantation

Network (OPTN) is trying to define a new continuous distribution model, which is intended to be a "more equitable system of allocating deceased donor organs," there will likely be discussion of which factors should be included in the system. Predicted post-transplant survival is a likely candidate for inclusion, as it is already used as a factor in lung allocations. The investigation in this chapter therefore uses data from the OPTN to see how well these post-transplant outcomes can be predicted for several organs using the current data, to better understand their usefulness if included in the allocation process. We show the following:

- The best AUCs we achieved for the different organs are overall low, at 0.66 for the liver data, 0.62 for lung data, and 0.67 for kidney data.
- Since the highest AUC overall was 0.67, when it comes to allocation these models would very frequently misclassify or fail to stratify the patient who has a higher post-transplant survival compared to one who has a lower chance of post-graft across all organs. It therefore is questionable whether post-transplant survival predictions should be used to stratify patients or as a factor in organ allocation decisions for now.
- The machine learning methods used to predict post-transplant survival are state-of-the-art modeling techniques, which have been applied effectively in the transplant area (ex. in defining MELD or the OPOM score ([100],[11])). We therefore argue that the main issue is with the data, and discuss several potential ways of addressing this issue.

Chapter 2

The equivalence of neural networks and optimal decision trees

2.1 Introduction

Neural networks, a supervised learning technique, have become one of the most widely used machine learning techniques today. Historically, one can trace the beginnings of neural networks to 1943. That year, [62] proposed to model neurons with simple electronic circuits, mirroring the fact that neurons, like circuits, either activate or not. Following this paper, there were more developments in the field, such as the creation of a system that could learn how to classify input data known as the Perceptron ([75]), the development of backpropagation, a technique to train neural networks ([99], [76]), the proof that multilayer feedforward neural networks are universal approximators ([53], [47]), and many more. Increased computational power, advances in optimization (stochastic gradient methods), and the massive availability of data sets have also lead to the development of a methodology known as deep

learning, which involves training large neural networks with many hidden layers. For a survey in developments in neural networks and deep learning, see [72], [55], and [78].

Generally, a neural network's architecture is defined by

- L hidden layers, indexed $\ell = 1, \ldots, L$, and one output layer.
- Hidden layer ℓ consisting of N_ℓ nodes, indexed $i = 1, \ldots, N_\ell$.
- An output layer consisting of q nodes, indexed $i = 1, \ldots, q$.
- Some non-linear function $\phi(x)$ associated with the hidden layers.
- Some function $\phi_O(x)$ associated with the output layer.

An example of a feedforward neural network can be seen in Figure 2-1. It has 2 hidden layers, with $N_1 = 2$ nodes in the first hidden layer and $N_2 = 3$ nodes in the second. One of the major innovations that took place in neural networks was an increase in variety in the $\phi(\cdot)$ functions used. While earlier versions such as the Perceptron used an indicator function $\phi(x) = 1\{x \geq 0\}$, to try to capture the binary way neurons either fired or did not, some more recent choices such as $\phi(x) = \max(x, 0)$ have also been used to create effective neural networks with high predictive accuracy.

Deep learning has had some great successes across a variety of applications. For example, it revolutionized image recognition ([87], [40], [45]), with competitors in the ImageNet Large Scale Visual Recognition Challenge almost exclusively using deep learning in their programs ([77], [52]). It has also been very successful in machine translation, and other forms of natural language processing – Google Translate improved dramatically when it began to use deep learning in its translation algorithms

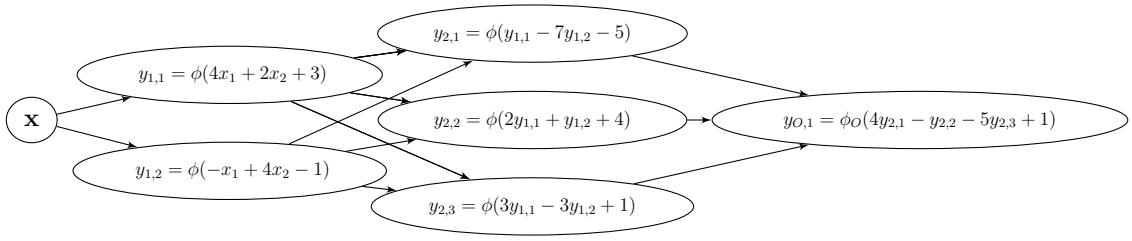

Figure 2-1: An example of a feedforward neural network.

([65], [25], [5], [93]). Additionally, speech recognition technology has markedly improved with the inclusion of deep learning ([37], [39]).

These examples, however, represent only a fraction of the applications neural networks are used in. With high profile successes, over 2 million results in a search on Google scholar on the topic "neural network articles" ([38]), and influential articles in the field garnering thousands of citations, it is clear that neural networks are an integral part of the field of artificial intelligence nowadays. However, neural networks face some challenges alongside their strengths and successes. They rely on heuristics in their training process, like dropout ([91], [103], [46]) and early stopping ([73], [104]). While neural networks often work well, it is unclear when they work well, why they work well, and if they do not work well how to improve them. Importantly, given that they have thousands to tens of thousands of parameters, they are not interpretable by humans.

In contrast with neural networks, classification and regression trees are highly interpretable ([6]). In the words of Leo Breiman, "On interpretability, trees rate an A+" ([18]). Trees partition the covariates separately, thus dividing the input data points into disjoint sets that are easily interpretable by humans. An example of this can be seen in Figure 2-2. Based on two covariates, body temperature (x_1) and blood glucose (HbA1c) (x_2), we want to classify whether a person is healthy. The

tree in Figure 2-2 classifies a person as healthy (Class 1) if ($x_1 < 97$) or ($x_1 > 97$ and $x_2 < 6.9$) and a person as not healthy (Class 0) if $x_1 > 97$ and $x_2 > 6.9$. The fact that the decision at each node is based on a single variable makes it very clear why a given path is taken, which is why decision trees are easy to use and understand, even when they have larger depths.

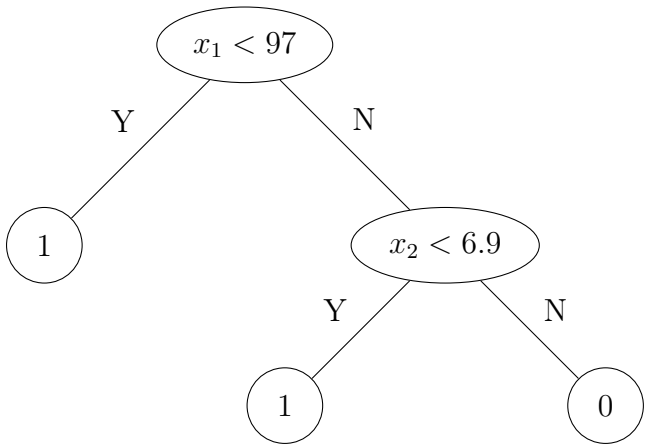

Figure 2-2: A sample decision tree.

Originally proposed by [18] CART is a greedy method for building trees that results in suboptimal trees. However, there has recently been significant progress in finding trees that are near optimal ([7, 8]). Combining mixed integer optimization and local search methods, the authors find optimal classification trees (OCTs) and optimal regression trees (ORTs) that significantly improve upon CART, while remaining computationally tractable. Furthermore, their approach allows one to consider hyperplane splits, leading to optimal classification trees with hyperplanes (OCT-Hs) and optimal regression trees with hyperplanes (ORT-Hs), which generalize support vector machines. These trees give comparable results in a variety of real world datasets with boosted trees and improve upon random forests, two widely used black box methods ([8]). OCT-Hs and ORT-Hs are less interpretable than

trees whose splits rely on only one variable (OCTs and ORTs), but are still more interpretable than neural networks. An example of this can be seen in Figure 2-3. Based on the same two covariates as before, body temperature (x_1) and blood glucose (HbA1c) (x_2), we again want to classify whether a person is healthy. The tree in Figure 2-3 classifies a person as healthy (Class 1) if ($0.4x_1 + 1.5x_2 < 46.7.7$) and ($0.9x_1 - 6x_2 < 53.1$) and a person as not healthy (Class 0) otherwise. While the hyperplane splits make the tree less interpretable, but it is still fairly clear why a point is sorted to a given leaf node.

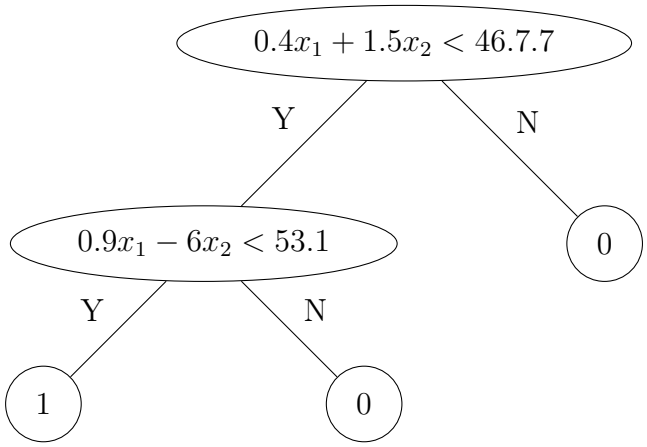

Figure 2-3: A sample decision tree with hyperplane splits.

In this paper, we investigate the modeling power of neural networks in comparison with OCT-Hs and ORT-Hs. We prove that a variety of neural networks (feedforward, convolutional and recurrent) can be transformed to classification and regression trees with hyperplanes with the same accuracy in the training set, showing that OCT-Hs and ORT-Hs are at least as powerful as neural networks. This work complements that of Sethi, who showed that decision trees can be transformed into neural networks with no loss in accuracy ([80]). We therefore know that OCT-Hs and neural networks are exactly equivalent in terms of the functions they can model. Given that our

constructions necessitate the construction of decision trees of significant depth, we explored the practical implication of these findings. We report a comparison of OCT-Hs and neural networks on twelve well known data sets and show that the two methods exhibit remarkably close accuracy.

Contributions

Our contributions include:

1. We show that a given classification neural network is equivalent in terms of modeling power to an OCT-H. In Table 2.1 we show the parameters of our construction and the section our construction appears in.

2. We show that a given regression neural network can be transformed to an ORT-H. In Table 2.2 we show the parameters of our construction and the section our construction appears in.

3. We show in Section 2.7 that neural networks trained in the machine learning framework TensorFlow and OCT-Hs trained in the Julia programming language have comparable out-of-sample performance in classifying data in twelve different data sets, and in one of them OCT-H has an edge in performance.

We feel that these findings are significant for the following reasons:

1. They link two of the most popular and widely utilized machine learning methods more deeply than their categorization as universal approximators, shedding new light on their strengths and weaknesses.

2. Given that OCT-Hs have an edge in interpretability compared to neural networks, without loss of modeling power, decision trees might be the method

Given Model	Given Parameters	At least as powerful model	New Model Parameters	Section
Classification Feedforward Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = 1\{x \geq 0\}$	OCT-H	• Depth N_1	2.4.1
Classification Feedforward Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = \max(x, 0)$	OCT-H	• Depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$	2.4.3
Classification Convolutional Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = 1\{x \geq 0\}$	OCT-H	• Depth N_1	2.5.1
Classification Convolutional Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = \max(x, 0)$	OCT-H	• Depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$	2.5.3
Classification Recurrent Neural Network	• Takes input sequence of length T^* • 1 hidden layer • N_1 nodes in that hidden layer • $\phi(x) = 1\{x \geq 0\}$	OCT-H	• Depth $T^* \times N_1$	2.6.1
Classification Recurrent Neural Network	• Takes input sequence of length T^* • 1 hidden layer • N_1 nodes in each hidden layer • $\phi(x) = \max(x, 0)$	OCT-H	• Depth $q-1+T^* \times N_1$	2.6.3

Table 2.1: Summary of the relationship of classification NNs and OCT-Hs.

Given Model	Given Parameters	At least as powerful model	New Model Parameters	Section
Regression Feedforward Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = 1\{x \geq 0\}$	ORT-H	• Depth N_1	2.4.2
Regression Feedforward Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = \max(x, 0)$	ORT-H	• Depth $\sum_{\ell=1}^{L} N_\ell$	2.4.4
Regression Convolutional Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = 1\{x \geq 0\}$	ORT-H	• Depth N_1	2.5.2
Regression Convolutional Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = \max(x, 0)$	ORT-H	• Depth $\sum_{\ell=1}^{L} N_\ell$	2.5.4
Regression Recurrent Neural Network	• Takes input sequence of length T^* • 1 hidden layer • N_1 nodes in that hidden layer • $\phi(x) = 1\{x \geq 0\}$	ORT-H	• Depth $T^* \times N_1$	2.6.2
Regression Recurrent Neural Network	• Takes input sequence of length T^* • 1 hidden layer • N_1 nodes in each hidden layer • $\phi(x) = \max(x, 0)$	ORT-H	• Depth $T^* \times N_1$	2.6.4

Table 2.2: Summary of the relationship of regression NNs and ORT-Hs.

of choice in applications where interpretability matters. This does not mean that we feel OCT-Hs will replace deep learning – just that in cases where interpretability is paramount, OCT-Hs have an advantage in that regard.

3. Given the success of stochastic gradient methods in neural networks, it might be worthwhile to investigate their application in the design of optimal trees. Conversely, given the success of mixed integer optimization and local search methods in optimal trees, it might be worthwhile to investigate their application in neural networks.

The structure of the rest of the paper is as follows. In Section 2.2, we discuss related literature comparing decision trees and neural networks. In Section 2.3, we review the mathematical structure of the neural networks we consider, as well as decision trees. In Sections 2.4, 2.5 and 2.6, we prove that we can transform feedforward, convolutional and recurrent neural networks into decision trees, respectively. In Section 2.7, we compare the classification accuracies of comparable decision trees and neural networks on twelve data sets. Lastly, in Section 2.8, we include our concluding remarks.

2.2 Related Work

The relationship of neural networks and decision trees as models and their similarities in performance have received a great deal of attention. Both models are universal approximators, allowing the two methods to be used to approximate each other ([58]). There have also been several papers comparing their performance in a purely empirical manner ([23], [1]).

One significant result in comparing the two techniques was the proof that a

decision tree can be transformed into an equivalent neural network ([80], [81]). Based on this transformation, there has been research investigating using decision trees as warm starts for neural networks ([50], [48]), and constructing neural network architectures based on decision trees and forests ([74], [66], [57]).

While there has been some work on transforming a neural network into an equivalent decision tree, the converse of Sethi's work, the networks involved have been small scale and the process is generally by inspection ([85]). There has been additional research into how to extract decision trees ([83]) and decision rules ([94], [97], [86], [84]) from neural networks, but the resultant models are not equivalent to the original networks, although they generally have comparable performance out-of-sample.

Lastly, there has also been some experimentation in designing decision trees and forests with non-deterministic splits, and comparing those models with neural networks ([82], [51], [35], [14]). A non-deterministic split is typically viewed as calculating the probability a point goes a particular direction at a split, and is usually represented as the output of a sigmoid function. While these models have the advantage of being continuous models, meaning that one can use stochastic gradient descent methods to train them, they lack the interpretability of decision trees with hard splits.

Our paper differs from this past work in the following ways:

1. Our proof that a decision tree can express the same function as a neural network is constructive, exact, and works for neural networks of any size or depth.

2. The splits in the tree we build are deterministic, not probabilistic.

3. We build a single decision tree, and not a less interpretable forest.

4. We cover a greater variety of network types (feedforward, convolutional, and recurrent), as opposed to just focusing on the feedforward case.

5. We discuss the ReLU activation function in addition to the perceptron activation function, which was not mentioned in much of the earlier work.

2.3 Mathematical Formulation of Neural Networks and Optimal Decision Trees

In this section, we describe the mathematical structure of the different types of neural networks and optimal classification and regression trees we consider in this paper.

2.3.1 Feedforward Neural Networks

A classification feedforward neural network (FNN) is trained on input and output pairs $(\mathbf{x}_j, o_j)$, $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $o_j \in \{1, \ldots, q\}$, $j = 1, \ldots, N$. The output o_j represents the class the point $\mathbf{x}_j$ belongs to. The characteristics of FNNs are:

- There are L hidden layers. Hidden layer $\ell = 1, \ldots, L$ consists of N_ℓ nodes. Node $n_{\ell,i}$, $i = 1, \ldots, N_\ell$, in hidden layer ℓ is characterized by a vector $\mathbf{W}_{\ell,i} \in \mathbb{R}^{N_{\ell-1}}$ and scalar $b_{\ell,i} \in \mathbb{R}$. The node computes

$$y_{\ell,i} = \phi(\mathbf{W}_{\ell,i}^T \mathbf{y}_{\ell-1} + b_{\ell,i}),$$

  where $\phi(x)$ is a nonlinear function, and $\mathbf{y}_{\ell-1}$ is the vector of outputs of the hidden layer $\ell - 1$. We define $\mathbf{y}_0 \triangleq \mathbf{x}$, the input of the FNN.

- There is one output layer that consists of q nodes. Node i, $i = 1, \ldots, q$ in the

output layer is characterized by $\mathbf{W}_{O,i} \in \mathbb{R}^{N_L}$ and scalar $b_{O,i} \in \mathbb{R}$. Unlike in the hidden layers, the output layer's activation function $\phi_O(x)$ is a vector valued function, where

$$y_{O,i} = \phi_O(\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O)_i, \ i = 1, \ldots, q. \tag{2.1}$$

For ease of notation in the diagrams, we will define

$$\phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_L + b_{O,i}) = \phi_O(\mathbf{W}_O^T \mathbf{y}_L + b_O)_i, \ i = 1, \ldots, q.$$

Note that $\phi_O(x)$ need not be the same as $\phi(x)$ applied element-wise to the vector $\mathbf{y}_L$.

- The final prediction of the network is found by calculating $k = \text{arg-lex-max}_{i=1,\ldots,q}(y_{O,i})$. The lexicographic maximum means that in case of a tie, the smallest index k is our choice. We then use k as our predicted class value.

An example of a classification feedforward neural network is depicted in Figure 2-4. Here, we have 2 hidden layers, with 3 nodes in the first hidden layer, 4 nodes in the second, and 2 nodes in the output layer. Also note that each node in this network has its own unique weights $\mathbf{W}_{\ell,i}$ and $b_{\ell,i}$ (which we have to solve for in the training process), and that the nodes in one layer have directed edges leading to all the nodes in the next layer (a trait known as being "fully connected").

Two potential choices for the activation function $\phi(\cdot)$ are the perceptron activation function, where

$$\phi(x) = 1\{x \geq 0\}, \tag{2.2}$$

and the rectified linear unit or ReLU, where

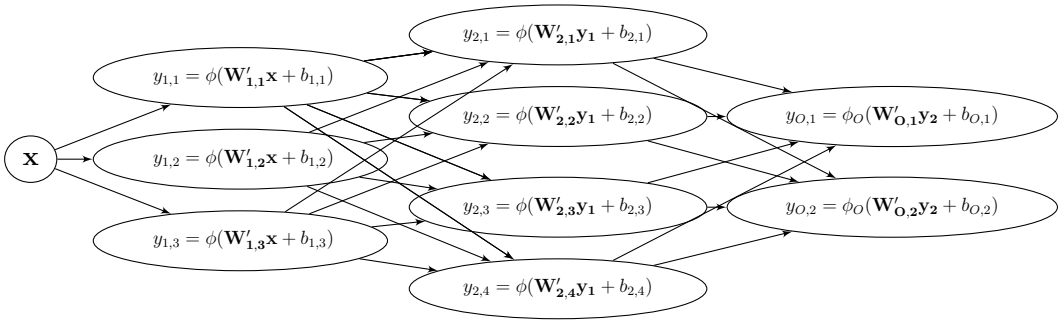

Figure 2-4: An example of a classification feedforward neural network.

$$\phi(x) = \max(x, 0). \tag{2.3}$$

If (2.5) is chosen as the activation function, then $(\phi_O(\mathbf{x}))_i = 1\{x_i \geq 0\}$. If (2.6) is chosen as the activation function, then $\phi_O(\mathbf{x})$ is defined as

$$(\phi_O(\mathbf{x}))_i = \begin{cases} 1, & \text{where } i = \text{argmax}_{i=1,\ldots,N_0}(x_i), \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

A regression FNN is defined in much the same way as the above, with a two minor differences. First, the training data pairs are $(\mathbf{x}_j, \mathbf{o}_j)$, $j = 1, \ldots, N$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $\mathbf{o}_j \in \mathbb{R}^q$. Second, $\phi_O(\mathbf{x})$ is the identity function, so the network simply outputs $\mathbf{W}_O^T\mathbf{y}_L + \mathbf{b}_O$.

When using neural networks, the standard goal is to solve for weight matrices and bias vectors $\mathbf{W}_\ell$ and $\mathbf{b}_\ell$, $\ell = 1, \ldots, L, O$. This is usually done by minimizing some loss function using stochastic gradient descent ([16], [56]), and is typically augmented with techniques like early stopping ([73], [104]) and dropout ([91], [103], [46]). Due to the heuristic nature of these methods, we are not guaranteed an optimal solution for $\mathbf{W}_\ell$ and $\mathbf{b}_\ell$, $\ell = 1, \ldots, L, O$.

Two potential choices for the activation function $\phi(\cdot)$ are the perceptron activation function, where

$$\phi(x) = 1\{x \geq 0\}, \tag{2.5}$$

and the rectified linear unit or ReLU, where

$$\phi(x) = \max(x, 0). \tag{2.6}$$

If (2.5) is chosen as the activation function, then $(\phi_O(\mathbf{x}))_i = 1\{x_i \geq 0\}$. If (2.6) is chosen as the activation function, then $\phi_O(\mathbf{x})$ is defined as

$$(\phi_O(\mathbf{x}))_i = \begin{cases} 1, & \text{where } i = \text{argmax}_{i=1,\ldots,N_0}(x_i), \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

A regression FNN is defined in much the same way as the above, with a two minor differences. First, the training data pairs are $(\mathbf{x}_j, \mathbf{o}_j)$, $j = 1, \ldots, N$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $\mathbf{o}_j \in \mathbb{R}^q$. Second, $\phi_O(\mathbf{x})$ is the identity function, so the network simply outputs $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$.

When using neural networks, the standard goal is to solve for weight matrices and bias vectors $\mathbf{W}_\ell$ and $\mathbf{b}_\ell$, $\ell = 1, \ldots, L, O$. This is usually done by minimizing some loss function using stochastic gradient descent ([16], [56]), and is typically augmented with techniques like early stopping ([73], [104]) and dropout ([91], [103], [46]). Due to the heuristic nature of these methods, we are not guaranteed an optimal solution for $\mathbf{W}_\ell$ and $\mathbf{b}_\ell$, $\ell = 1, \ldots, L, O$.

2.3.2 Convolutional Neural Networks

A convolutional neural network (CNN) ([78], [37]) is designed to be especially effective at classifying images, so its architecture is meant to exploit the traits of image data. CNNs use three types of hidden layers: convolutional, pooling and regular layers. It is is trained on input and output pairs $(\mathbf{x}_j, o_j)$, $j = 1, \ldots, N$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $o_j \in \{1, \ldots, q\}$, $j = 1, \ldots, N$. The output o_j represents the class the point $\mathbf{x}_j$ belongs to. Their characteristics are as follows.

- There are L hidden layers in the network. The first L_c are pairs of convolutional layers and pooling layers, while the remaining $L - L_c$ are regular hidden layers like those in Section 2.3.1.

- The convolutional and the regular hidden layers each have N_ℓ nodes, $\ell = 1, \ldots, L_c, L_c + 1, \ldots, L$. Node $n_{\ell,i}$, $i = 1, \ldots, N_\ell$, $\ell = 1, \ldots, L$ is characterized by a vector $\mathbf{W}_{\ell,i} \in \mathbb{R}^{N_{\ell-1}}$ and scalar $b_{\ell,i} \in \mathbb{R}$. The node computes

$$y_{\ell,i} = \begin{cases} \phi(\mathbf{W}_{\ell,i}^T \mathbf{P}_{\ell-1} + b_{\ell,i}), & \text{if } \ell = 1, \ldots, L_c, \\ \phi(\mathbf{W}_{\ell,i}^T \mathbf{y}_{\ell-1} + b_{\ell,i}), & \text{if } \ell = L_c + 1, \ldots, L, \end{cases}$$

  where $\phi(x)$ is a nonlinear function, $\mathbf{P}_{\ell-1}$ is the vector of outputs of the previous pooling layer, $\ell = 1, \ldots, L_c$ and $\mathbf{y}_{\ell-1}$ is the vector of outputs of the previous regular hidden layer $\ell = L_c + 1, \ldots, L$.

- We define $\mathbf{P}_0 \triangleq \mathbf{x}$

- The pooling layers all have R_ℓ nodes, $\ell = 1, \ldots, L_c$, with the property that

$k_\ell = \frac{N_\ell}{R_\ell}$ is an integer. Node $p_{\ell,i}$, $i = 1, \ldots, R_\ell$, in a pooling layer computes

$$P_{\ell,i} = \psi(y_{\ell,m},\ m \in S_{\ell,i}),$$

where $\psi(\cdot)$ is some function, $\mathbf{y}_\ell$ is the vector of outputs of the previous convolutional layer, and $S_{\ell,i}$ is the collection of indices that affect the computation of $P_{\ell,i}$. The sets $S_{\ell,i}$ have the properties

1. $\cup_{i=1}^{R_\ell} S_{\ell,i} = \{1, \ldots, N_\ell\}$
2. $S_{\ell,i} \cap S_{\ell,j} = \varnothing$
3. $|S_{\ell,i}| = k_\ell, i = 1, \ldots, R_\ell$

The most commonly used pooling layer type is the max pooling layer that calculates:

$$P_{\ell,i} = \max_{m \in S_{\ell,i}} (y_{\ell,m}),\ i = 1, \ldots, R_\ell. \tag{2.8}$$

- The output layer consists of q nodes. A node in the output layer is characterized by $\mathbf{W}_{O,i} \in \mathbb{R}^{N_L}$ and scalar $b_{O,i} \in \mathbb{R}$. The node computes $y_{O,i} = \phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_L + b_{O,i})$ as defined in Eq. (2.1), where $\phi_O(x)$ is some function, which need not be $\phi(x)$ applied element-wise to an input vector.

- The final prediction of the network is found by calculating $k = \text{arg-lex-max}_{i=1,\ldots,q}(y_{O,i})$. The lexicographic maximum means that in case of a tie, the smallest index k is our choice. We then use k as our predicted class value.

A regression CNN is defined in much the same way as the above, with a two minor differences. First, the training data pairs are $(\mathbf{x}_j, \mathbf{o}_j)$, $j = 1, \ldots, N$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$

and $\mathbf{o}_j \in \mathbb{R}^q$. Second, $\phi_O(\mathbf{x})$ is the identity function, so the network simply outputs $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$.

2.3.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are specialized to be able to handle sequential input data ([60]). In general, it is expected that the data have the same dimensions at each step in the sequence, but that the sequence can be of arbitrary length. For this paper, though, we will assume with some loss of generality both that the data have the same dimensions at each step in the sequence and that every potential input sequence has length T^*. The way that a simple RNN works is that the network has a single hidden layer that processes the data in a sequential manner over a series of time steps $t = 1, \ldots, T^*$. It "remembers" past inputs by using the hidden layer output at time step t as an input into the hidden layer at time step $t+1$. As a result of this, the network structure of RNNs contains self loops, unlike FNNs and CNNs.

At the final time step T^*, and given output function $\phi_O(\cdot)$ and output weights $\mathbf{W}_O \in \mathbb{R}^{n_1 \times q}$ and $\mathbf{b}_O \in \mathbb{R}^q$, we then calculate $\mathbf{y}_O$, where $y_{O,i} = \phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_{T^*,1} + b_{O,i})$ for $i = 1, \ldots, q$. This classification prediction can be viewed in some cases as the network predicting the next value in the given sequence.

A recurrent neural network is trained on the input sequence and output pair $((\mathbf{x}_{t,j})_{t=1,\ldots,T^*}, o_j)$, $j = 1, \ldots, N$, where $\mathbf{x}_t \in \mathbb{R}^{N_0}$, $t = 1, \ldots, T^*$, and $o_j \in \{1, \ldots, q\}$ is the class the sequence $(\mathbf{x}_{t,j})_{t=1,\ldots,T^*}$ is in. Their characteristics are as follows.

- There is one hidden layer and one output layer. The hidden layer contains N_1 nodes, $i = 1, \ldots, N_1$. Node $n_{1,i}$ is characterized by vectors $\mathbf{W}_{g,i} \in \mathbb{R}^{N_0}$ and

$\mathbf{W}_{h,i} \in \mathbb{R}^{N_1}$, and scalar $b_{1,i}$. It computes

$$y_{t,1,i} = \phi(\mathbf{W}_{g,i}^T \mathbf{x}_t + \mathbf{W}_{h,i} \mathbf{y}_{t-1,1} + b_{1,i}),$$

where $\phi(x)$ is some non-linear function, and $\mathbf{y}_{t-1,1}$ is the vector of outputs of the hidden layer in the previous time step.

- We define $\mathbf{y}_{0,1}$ as the zero vector $\mathbf{0} \in \mathbb{R}^{N_1}$.
- The output layer consists of q nodes, $i = 1, \ldots, q$. A node in the output layer is characterized by $\mathbf{W}_{O,i} \in \mathbb{R}^{N_1}$ and scalar $b_{O,i} \in \mathbb{R}$. It computes $y_{O,i} = \phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_{T^*,1} + b_{O,i})$ as defined in Eq. (2.1), where $\phi_O(x)$ is some function that need not be the same as $\phi(x)$ applied element-wise to the vector $\mathbf{y}_L$.
- The final prediction of the network is found by calculating $k = \text{arg-lex-max}_{i=1,\ldots,q}(y_{O,i})$. The lexicographic maximum means that in case of a tie, the smallest index k is our choice. We then use k as our predicted class value.

A regression RNN is defined in much the same way as the above, with a two minor differences. First, the training data inputs are $((\mathbf{x}_{t,j})_{t=1,\ldots,T^*}, \mathbf{o}_j)$, $j = 1, \ldots, N$, where $\mathbf{x}_t \in \mathbb{R}^{N_0}$, $t = 1, \ldots, T^*$, and $\mathbf{o}_j \in \mathbb{R}^q$. Second, $\phi_O(\mathbf{x})$ is the identity function, so the network simply outputs $\mathbf{W}_O^T \mathbf{y}_{T^*,1} + \mathbf{b}_O$.

2.3.4 Optimal Classification and Regression Trees

Given training input data in $\mathbb{R}^{N_0}$, a classification tree partitions the space into disjoint subspaces. Following this it assigns a classification value to each subspace. Once this is complete, given an input data point the tree sorts it into a subspace, and the class

value associated with this subspace is the tree's prediction for the point's class value. The decision tree is trained on input and output pairs $(\mathbf{x}_j, o_j)$, $j = 1, \ldots, N$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $o_j \in \{1, \ldots, q\}$ is the class $\mathbf{x}_j$ is in. Its characteristics are as follows.

- It has depth N_1, where the depth is the maximum number of split nodes in a tree one visits before reaching a leaf node that contains an output value.
- The maximal tree of depth N_1 has $T = 2^{N_1+1} - 1$ nodes.
- Nodes 1 through $\lfloor T/2 \rfloor$ of this maximal tree are split nodes, otherwise known as branch nodes, while nodes $\lfloor T/2 \rfloor + 1$ through T are leaf nodes.
- Each branch node i, $i = 1, \ldots, \lfloor T/2 \rfloor$ is assigned split parameters $\mathbf{w}_i, b_i$, where $\mathbf{w}_i \in \mathbb{R}^{N_0}$ and $b_i \in \mathbb{R}$.
- Given input $\mathbf{x}$, at a given node i we calculate $\mathbf{w}_i^T \mathbf{x} + b_i$.
- If $\mathbf{w}_i^T \mathbf{x} + b_i < 0$, we take the left branch of the split to a new tree node; otherwise, we take the right branch.
- Once we have passed through at most N_1 different nodes, we arrive at a leaf node.
- Each leaf node is assigned a classification value $k \in \{1, \ldots, q\}$ that it uses as the predicted class for all points sorted to it.
- Thus, if $\mathbf{x}$ is assigned to leaf node r with classification value k_r, $r \in \{\lfloor T/2 \rfloor + 1, \ldots, T\}$ and $k_r \in \{1, \ldots, q\}$, the network outputs k_r as the classification value for $\mathbf{x}$.

In the past, split parameters $\mathbf{w}_i, b_i$ for the branch nodes and the classification values k_r for the leaf nodes were found using greedy methods like CART, which result in suboptimal solutions and are only able to handle splits based on a single parameter of $\mathbf{x}$. This would mean that exactly one entry of $\mathbf{w}_i$ could be nonzero for all $i = 1, \ldots, \lfloor T/2 \rfloor$. As shown in [8], if one uses mixed integer optimization and local search methods to solve for split parameters and leaf classification values given a maximum tree depth N_1, one is capable of finding a near optimal tree in a computationally tractable manner. Furthermore, these optimal classification trees are able to use hyperplane based splits that can take into account all parameters of $\mathbf{x}$ at a given split, instead of just a single one. These traits allow optimal decision trees with hyperplane splits to be more flexible models while increasing their accuracy in classifying training data, at the cost of making them a bit less interpretable. Optimal classification trees with splits that are parallel to the axis are denoted as OCT and with hyperplane splits as OCT-H.

Decision trees in general can be used for regression as well. Here the training data are $(\mathbf{x}_j, \mathbf{o}_j)$, $j = 1, \ldots, N$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $\mathbf{o}_j \in \mathbb{R}^q$. The splits are the same as before, but there are two different possibilities for what kind of output to associate with the leaf nodes. One version involves each leaf having a single point estimate output value y_O^r associated with it (typically the mean of all the outputs of data points sorted to that leaf node), for $r = \lfloor T/2 \rfloor + 1, \ldots, T$. The other is when there is a different linear function defined by weights $\mathbf{W}_O^r, \mathbf{b}_O^r$ at each node r, where $\mathbf{W}_O^r \in \mathbb{R}^{N_0 \times q}$ and $\mathbf{b}_O^r \in \mathbb{R}^q$. This linear function is applied to all the points sorted to the rth leaf in order to to make the prediction $(\mathbf{W}_O^r)^T \mathbf{x} + \mathbf{b}_O^r$ as the point's output value for $r = \lfloor T/2 \rfloor + 1, \ldots, T$. Current optimal tree implementations are able to solve for optimal regression trees when $q = 1$. Optimal regression trees with splits that are parallel to the axis are denoted with ORT and with hyperplane splits with

ORT-H.

Figure 2-5 contains an example of a classification tree of depth 2. The split in first node is based on whether $\mathbf{w}_1^T\mathbf{x} < b_1$ or not, and the splits in nodes second and third nodes are respectively based on whether $\mathbf{w}_2^T\mathbf{x} < b_2$ or not or whether $\mathbf{w}_3^T\mathbf{x} < b_3$ or not.

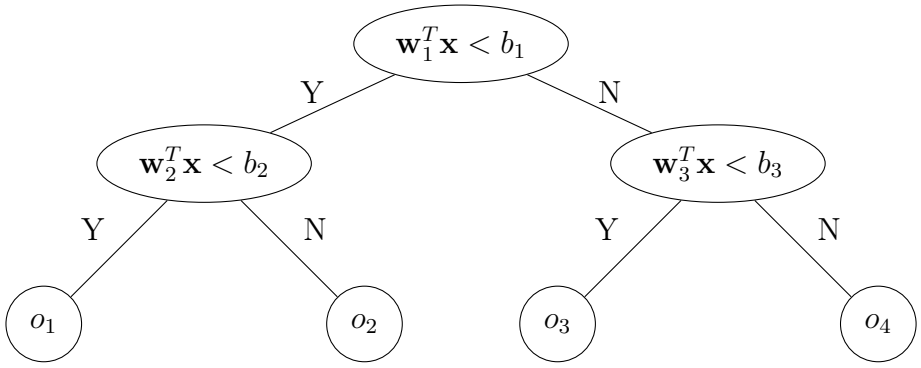

Figure 2-5: An OCT-H of depth 2. Data in the four leaf nodes are classified as o_1, o_2, o_3, and o_4.

2.4 Feedforward Neural Networks and Optimal Trees

In this section, we construct an OCT-H (ORT-H) that can classify training data at least as well as a classification (regression) feedforward neural network (FNN) with perceptron or Rectified Linear Unit activation functions.

2.4.1 Perceptron Classification FNNs and OCT-Hs

The key result of this section is as follows.

Theorem 1. *An OCT-H with maximum depth N_1 can classify the data in a training set at least as well as a given classification FNN with the perceptron activation function (2.5) and N_1 nodes in the first hidden layer.*

Proof. Our proof is constructive. We are given a FNN $\mathcal{N}_1$ with the following characteristics:

- The perceptron activation function as defined in (2.5).
- Output function $\phi_O(\mathbf{x}) : [0,1]^q \to [0,1]^q$.
- L hidden layers and one output layer, indexed $\ell = 1, \ldots, L, O$.
- N_ℓ nodes in each layer, indexed $i = 1, \ldots, N_\ell$.
- q nodes in the output layer, indexed $i = 1, \ldots, q$.
- Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$.

We construct a classification tree $\mathcal{T}_1$ with hyperplane splits of maximum depth N_1 that makes the same predictions as $\mathcal{N}_1$. This construction relies on the fact that a FNN with the perceptron activation function and N_1 nodes in the first hidden layer has at most 2^{N_1} distinct outputs from the first hidden layer, which means the entire network has at most 2^{N_1} distinct output values. We can therefore assign the 2^{N_1} outputs of the network to each of the 2^{N_1} leaf nodes of the tree. It follows that an OCT-H of maximum depth N_1 has at least the same classification accuracy as $\mathcal{N}_1$.

To construct $\mathcal{T}_1$, given the inequality from the first node in the first hidden layer of $\mathcal{N}_1$ defined by the weight vector $\mathbf{W}_{1,1}$ and bias scalar $b_{1,1}$, we define the first split as

$$\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1} < 0, \tag{2.9}$$

which results in the simple split seen in Figure 2-6.

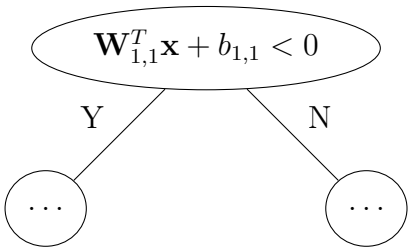

Figure 2-6: The first split of decision tree $\mathcal{T}_1$.

Independent of whether inequality (2.9) is satisfied or not, the second split is given by

$$\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2} < 0.$$

Figure 2-7 provides a visualization of the new branches we added to the tree in Figure 2-6.

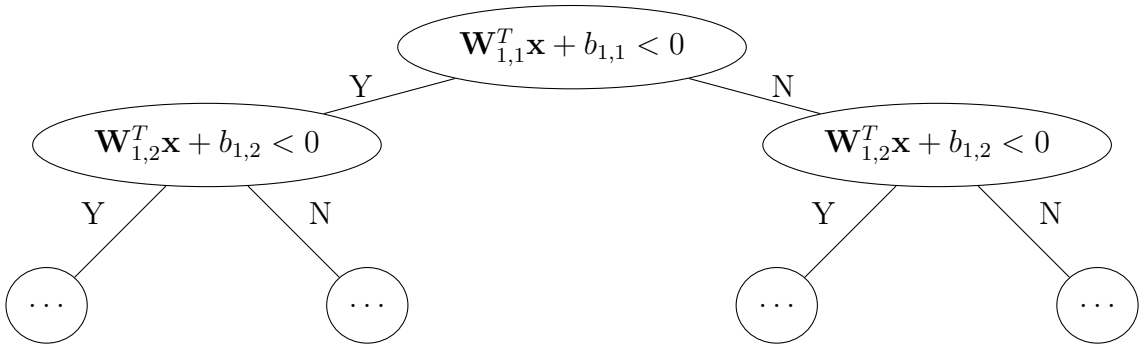

Figure 2-7: The first two depths of tree $\mathcal{T}_1$.

We continue this process for all N_1 nodes in the first hidden layer, building a

decision tree of depth N_1, with every split at depth N_1 being given by

$$\mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1} < 0.$$

Having defined the splits of the tree $\mathcal{T}_1$, we now outline how to find the class value associated with each of the tree's leaves. Suppose that input $\mathbf{x}$ is assigned to the rth leaf node of tree $\mathcal{T}_1$, $r = 1, \ldots, 2^{N_1}$. Define $\mathcal{I}_{1,r}$ as

$$\mathcal{I}_{1,r} = \{i \mid \mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} \geq 0\},$$

which is the set of all depths i of tree $\mathcal{T}_1$ where the inequality $\mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} \geq 0$ is satisfied by an input $\mathbf{x}$ sorted to the rth leaf node. Likewise define $\mathcal{I}_{2,r}$ as

$$\mathcal{I}_{2,r} = \{i \mid \mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} < 0\}.$$

We know that in the neural network the first hidden layer outputs are

$$1\{\mathbf{W}'_{1,i} \mathbf{x} + b_{1,i} \geq 0\} = 1 \text{ for } i \in \mathcal{I}_{1,r} \text{ and } 1\{\mathbf{W}_{2,i} \mathbf{x} + b_{2,i} \geq 0\} = 0 \text{ for } i \in \mathcal{I}_{2,r}.$$

To complete the construction of $\mathcal{T}_1$, we need to assign a classification value to every leaf of $\mathcal{T}_1$. Given an input $\mathbf{x}$ of $\mathcal{N}_1$, there are 2^{N_1} possible binary vectors that the first hidden layer of $\mathcal{N}_1$ could output. These 2^{N_1} vectors, by our construction of $\mathcal{T}_1$, exactly correspond to the 2^{N_1} leaves of $\mathcal{T}_1$. Given $\mathbf{y}_1^r$, the output of the first hidden layer associated with leaf node r, the final prediction of $\mathcal{N}_1$ will be $k(\mathbf{y}_1^r)$, which is calculated deterministically given the $\mathbf{y}_1^r$ vector and the $\mathbf{W}_{\ell,i}$, $b_{\ell,i}$ values by using the process outlined in Section 2.3.1. In every node r of the tree we assign the classification value $k(\mathbf{y}_1^r)$, the classification value associated with the first hidden

layer output.

We next show that the output of $\mathcal{T}_1$ is the same as the output of $\mathcal{N}_1$ for input data point $\mathbf{x}$. To see this, if $\mathbf{x}$ is input into $\mathcal{N}_1$, the first hidden layer outputs $\mathbf{y}_1(\mathbf{x})$, resulting in the final network output $k(\mathbf{y}_1)$. However, in the decision tree, $\mathbf{x}$ is assigned to the leaf node where $\mathbf{y}_1^r = \mathbf{y}_1(\mathbf{x})$, and is once again assigned output value $k(\mathbf{y}_1^r) = k(\mathbf{y}_1)$ by construction. Thus, for a given data point $\mathbf{x}$, the network and the tree predict the same classification value.

Since an OCT-H does at least as well as $\mathcal{T}_1$ in classifying the training data, it must do at least as well as $\mathcal{N}_1$ too. Thus, by construction, we have that an optimal decision tree with maximum depth N_1 can classify data in a training set at least as well as the given FNN with perceptron activation function, $L \geq 1$ hidden layers, and N_1 nodes in the first hidden layer, completing the proof of the theorem. □

We remark that

1. The construction of tree $\mathcal{T}_1$ is independent of L, the number of hidden layers.

2. While the output function $\phi_O(\mathbf{x})$ affects the values for classification, the construction of $\mathcal{T}_1$ is not affected by $\phi_O(\mathbf{x})$; only the output values of the leaves of $\mathcal{T}_1$ are affected by $\phi_O(\mathbf{x})$.

3. With this result, we have that any perceptron neural network can be modeled as a decision tree with hyperplane splits. Combined with the result from [80] that any decision tree can be modeled as a perceptron neural network, the two techniques must have the exact same modeling ability, indicating that they are equivalent.

We next present an example of how to perform the above procedure. The neural

network we are working with was trained on data $(\mathbf{X}, \mathbf{o})$ and is shown in Figure 2-8, and the resultant decision tree is shown in Figure 2-9.

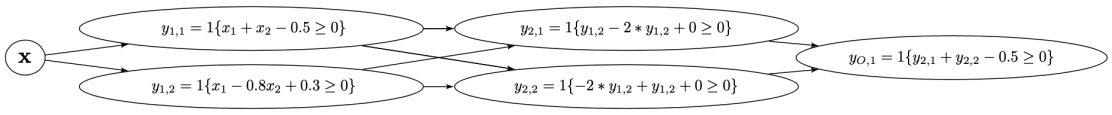

Figure 2-8: An FNN with the perceptron activation function performing an XOR operation.

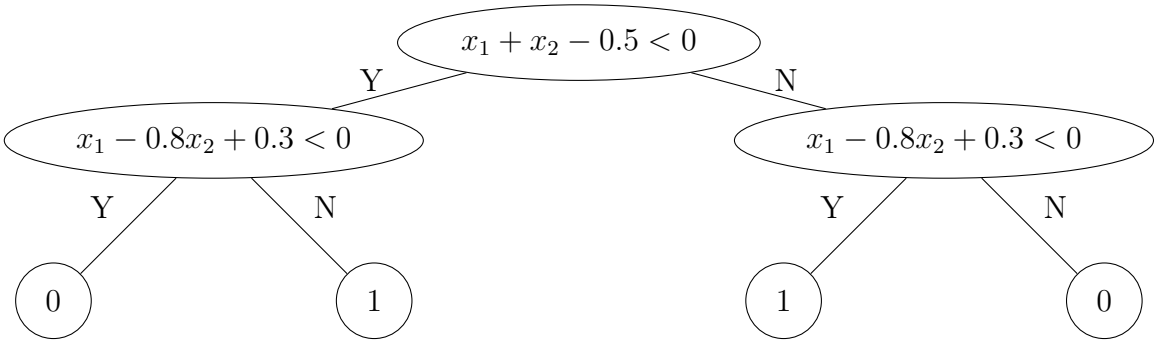

Figure 2-9: The reformulation of the Figure 2-8 neural network into a decision tree.

The network in Figure 2-8 is an XOR network; it outputs one if $x_1 + x_2 - 0.5 \geq 0$ and $x_1 - 0.8x_2 + 0.3 < 0$, or if $x_1 + x_2 - 0.5 < 0$ and $x_1 - 0.8x_2 + 0.3 \geq 0$, and zero otherwise. We can reformulate it into the tree in Figure 2-9 by defining the split at the first node of the decision tree as

$$x_1 + x_2 - 0.5 < 0.$$

Likewise, after that we define the split at both nodes at depth two of the decision tree as

$$x_1 - 0.8x_2 + 0.3 < 0.$$

We then decide which output values to assign to which leaf nodes by tracing what a given data point's path down the tree would be. For example, take some point like $\mathbf{x} = (0.3, 1)$, which takes the right path at the first node in the decision tree, and then the left path at the second node. Then we have that $y_{1,1} = 1$ and $y_{1,2} = 0$. Inputing these values into the second layer of the neural network gives $y_{2,1} = 1$ and $y_{2,2} = 0$. Finally, inputting those values into the final node of the network results in $y_{O,1} = 1$, so we assign that value to the leaf node $\mathbf{x}$ arrives at. By using this method, we assign output values of one to the second and third leaf nodes, and zeros elsewhere. Thus, the tree we construct classifies the given data in the same way as the neural network. By solving for an optimal tree with depth 2 over the data $(\mathbf{X}, \mathbf{o})$, we must do at least as well on the data as the Figure 2-9 tree by the optimality of the final OCT-H. Thus, the optimal tree does at least as well as the Figure 2-8 neural network as well.

We feel that the construction gives insights into the workings of a perceptron neural network. One can imagine that such a network has the first layer nodes serving as the splits for a given decision tree, with the remaining hidden layers and the output layer just serving as a particularly complex way of solving for the leaf node values.

2.4.2 Perceptron Regression FNNs and ORT-Hs

The only difference between a regression FNN with the perceptron activation function and a classification FNN with the perceptron activation function in the construction of the network is that $\phi_O(\mathbf{y}_L) \in \mathbb{R}^q$. Because the first hidden layer of such a regression FNN can output at most 2^{N_1} unique vectors, there are only 2^{N_1} possible output vectors of the network. In this case, one can modify the proof in Section 2.4.1 by

assigning these 2^{N_1} unique vectors to the leaf nodes of the decision tree in the place of classification values. With this adjustment, extending Theorem 1 to regression FNNs with perceptron activation functions is straightforward.

2.4.3 Rectified Linear Unit Classification FNNs and OCT-Hs

In this section, we show that given a classification FNN with ReLU activation functions, we can construct a classification tree with hyperplane splits that can classify the given training data at least as well as the given FNN. Unlike in Section 2.4.1, we are dealing with a continuous as opposed to a discrete activation function, which requires a different process to construct an equivalent decision tree.

The theorem is as follows.

Theorem 2. *An OCT-H with maximum depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$ can classify data in a training set at least as well as a given classification FNN with the rectified linear unit activation function (2.6), L hidden layers, N_ℓ nodes in layer $\ell = 1, \ldots, L$, and q nodes in the output layer.*

Proof. We are given a FNN $\mathcal{N}_2$ with the following characteristics:

- The rectified linear unit activation function, as defined in (2.6).
- L hidden layers and one output layer, indexed $\ell = 1, \ldots, L, O$.
- N_ℓ nodes in each layer, indexed $i = 1, \ldots, N_\ell$.
- q nodes in the output layer, indexed $i = 1, \ldots, q$.
- Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$.

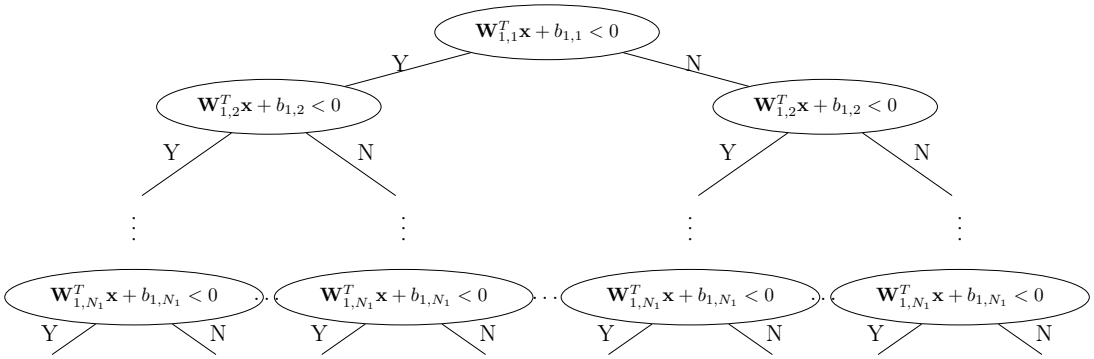

Figure 2-10: The decision tree $\mathcal{T}_2$ we are building up to depth N_1.

We construct a classification tree $\mathcal{T}_2$ with hyperplane splits of maximum depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$ that makes the same predictions as $\mathcal{N}_2$. It follows that an OCT-H of maximum depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$ has at least the same classification accuracy as $\mathcal{N}_2$.

We build the tree up to depth N_1 exactly the same way as in the proof in Section 2.4.1. This part of the tree is shown in Figure 2-10. This is the subtree that simulates the output of $\mathcal{N}_2$ after the first hidden layer.

After depth N_1, there are 2^{N_1} branches, as shown in Figure 2-10. Note that there are 2^{N_1} possible output vectors $\mathbf{y}_1$ of the first layer of $\mathcal{N}_2$,

$$(0, \ldots, 0)^T, (\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1}, \ldots, 0)^T, \ldots, (\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1}, \ldots, \mathbf{W}_{1,N_1}^T\mathbf{x} + b_{1,N_1})^T,$$

as each node of the first layer $\mathcal{N}_2$ computes

$$\mathbf{y}_{1,i} = \max\{\mathbf{W}_{1,i}^T\mathbf{x} + b_{1,i}, 0\},\ i = 1, \ldots, N_1.$$

These 2^{N_1} possible values of $\mathbf{y}_1$ correspond to the 2^{N_1} branches in the tree $\mathcal{T}_2$ shown in Figure 2-10. In each of these branches we have implicitly calculated the

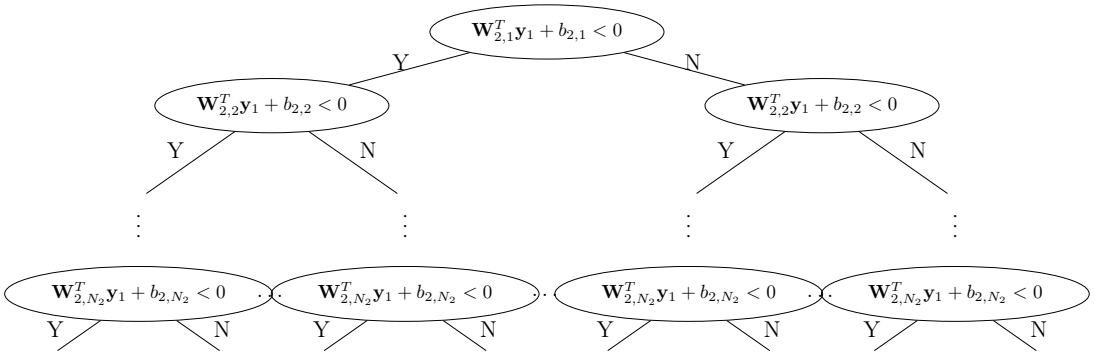

Figure 2-11: Subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ of depth N_2 is concatenated to the corresponding branch of the subtree depicted in Figure 2-10, resulting in a subtree of depth N_1+N_2.

corresponding $\mathbf{y}_1$ values. For example, the first branch corresponds to $(0,\ldots,0)^T$, the second corresponds to $(0,\ldots,0,\mathbf{W}_{1,N_1}^T\mathbf{x}+b_{1,N_1})^T$, etc.

We then model the second layer of $\mathcal{N}_2$ by constructing after each branch a new subtree of depth N_2 as in Figure 2-10, but with the corresponding value of $\mathbf{y}_1$ playing the role of $\mathbf{x}$. In the subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ in Figure 2-11 we substitute in the corresponding value of $\mathbf{y}_1$ as a linear function of $\mathbf{x}$. For example, if $\mathbf{y}_1 = (0,\ldots,0,\mathbf{W}_{1,N_1}^T\mathbf{x}+b_{1,N_1})$, then subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ is depicted in Figure 2-12.

Given that at each branch we know exactly $\mathbf{y}_1$, as an explicit function of $\mathbf{x}$, $\mathcal{T}_{2,2}(\mathbf{y}_1)$ is a decision tree where all inequalities are explicitly written as linear functions of $\mathbf{x}$. Continuing in this way we model the output vector $\mathbf{y}_\ell$ of the ℓth hidden layer of $\mathcal{N}_2$ as a classification tree by propagating the values of $\mathbf{y}_1$, $\mathbf{y}_2$,..., $\mathbf{y}_L$ as explicit linear functions of $\mathbf{x}$.

We model the output layer similarly by observing that in this layer we calculate

$$\text{argmax}_{i=1,\ldots,q}(\mathbf{W}_{O,i}^T\mathbf{y}_L + b_{O,i}) \tag{2.10}$$

in order to find the output class. The construction of the subtree $\mathcal{T}_{2,O}(\mathbf{y}_L)$ is depicted

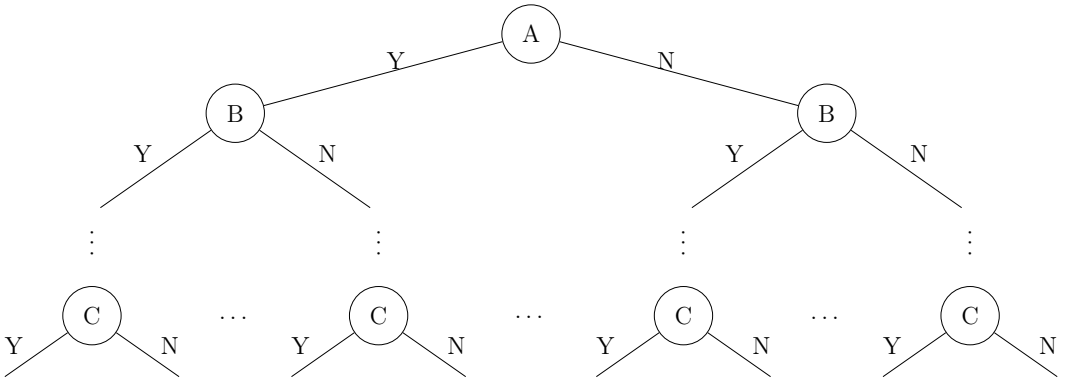

Figure 2-12: The resulting subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ for $\mathbf{y}_1 = (0, \ldots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})$. A, B, C are as follows:

- A is $\mathbf{W}_{2,1}^T (0, \ldots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})^T + b_{2,1} < 0$.
- B is $\mathbf{W}_{2,2}^T (0, \ldots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})^T + b_{2,2} < 0$.
- C is $\mathbf{W}_{2,N_2}^T (0, \ldots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})^T + b_{2,N_2} < 0$.

in Figure 2-13.

We simulate the calculation of (2.10) with $\mathcal{T}_{2,O}(\mathbf{y}_L)$ in the following manner. Node A checks whether $\mathbf{W}_{O,1}^T \mathbf{y}_L + b_{O,1}$ or $\mathbf{W}_{O,2}^T \mathbf{y}_L + b_{O,2}$ is larger. Node B_1 checks whether $\mathbf{W}_{O,2}^T \mathbf{y}_L + b_{O,2}$ or $\mathbf{W}_{O,3}^T \mathbf{y}_L + b_{O,3}$ is larger conditioned that $\mathbf{W}_{O,2}^T \mathbf{y}_L + b_{O,2} > \mathbf{W}_{O,1}^T \mathbf{y}_L + b_{O,1}$. Node B_2 checks whether $\mathbf{W}_{O,1}^T \mathbf{y}_L + b_{O,1}$ or $\mathbf{W}_{O,3}^T \mathbf{y}_L + b_{O,3}$ is larger conditioned that $\mathbf{W}_{O,1}^T \mathbf{y}_L + b_{O,1} > \mathbf{W}_{O,2}^T \mathbf{y}_L + b_{O,2}$, and so on. Each branch of the tree thus explicitly calculates which output node outputs the highest value (using a lexicographic decision rule in case of ties). We can then assign the class associated with that node as the output for the appropriate leaf nodes of $\mathcal{T}_{2,O}(\mathbf{y}_L)$. Since at each branch we know $\mathbf{y}_L$ as an explicit function of $\mathbf{x}$, we also have that $\mathcal{T}_{2,O}(\mathbf{y}_L)$ is a decision tree where all inequalities are explicitly written linear functions of $\mathbf{x}$ as well.

By the construction of $\mathcal{T}_2$, the output value of $\mathcal{T}_2$ is the same as $\mathcal{N}_2$ given an input

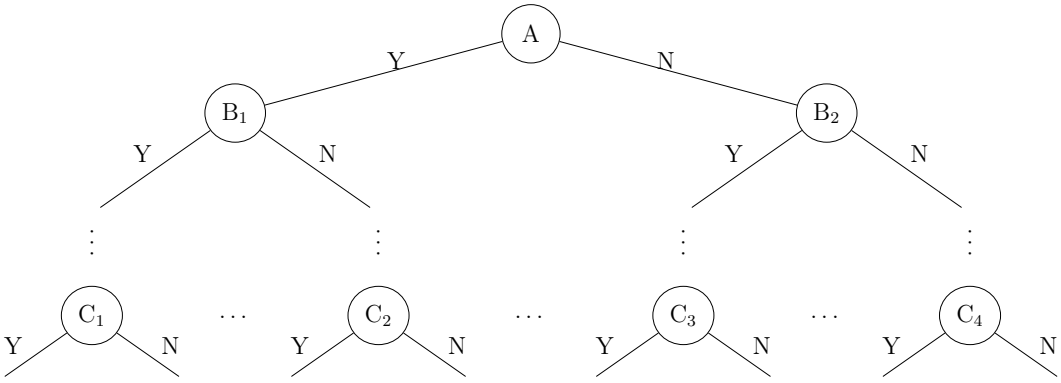

Figure 2-13: The subtree $\mathcal{T}_{2,O}(\mathbf{y}_L)$. The labels $A, B_1, B_2, C_1, C_2, C_3, C_4$ are as follows:

- A is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,2})^T \mathbf{y}_L + b_{O,1} - b_{O,2} < 0$.
- B_1 is $(\mathbf{W}_{O,2} - \mathbf{W}_{O,3})^T \mathbf{y}_L + b_{O,2} - b_{O,3} < 0$.
- B_2 is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,3})^T \mathbf{y}_L + b_{O,1} - b_{O,3} < 0$.
- C_1 is $(\mathbf{W}_{O,q-1} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,q-1} - b_{O,q} < 0$.
- C_2 is $(\mathbf{W}_{O,5} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,5} - b_{O,q} < 0$.
- C_3 is $(\mathbf{W}_{O,3} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,3} - b_{O,q} < 0$.
- C_4 is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,1} - b_{O,q} < 0$.

vector $\mathbf{x}$. Since $\mathcal{T}_2$ is a classification tree with hyperplanes, it follows that an OCT-H has at least as good accuracy as $\mathcal{T}_2$, completing the proof of the theorem. □

We next present an example of the construction. The FNN with the ReLu activation function is shown in Figure 2-14, using the output function defined in Eq.(2.7), and the resulting classification tree with hyperplane splits is shown in Figure 2-15.

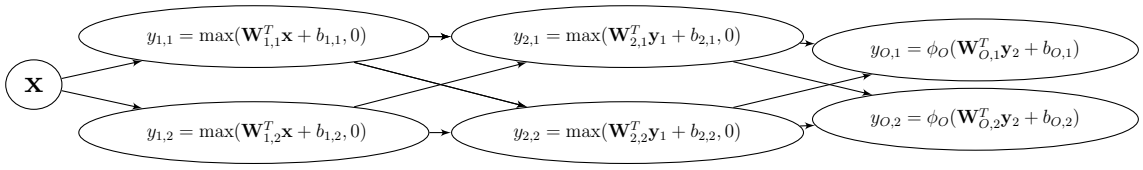

Figure 2-14: A ReLU FNN.

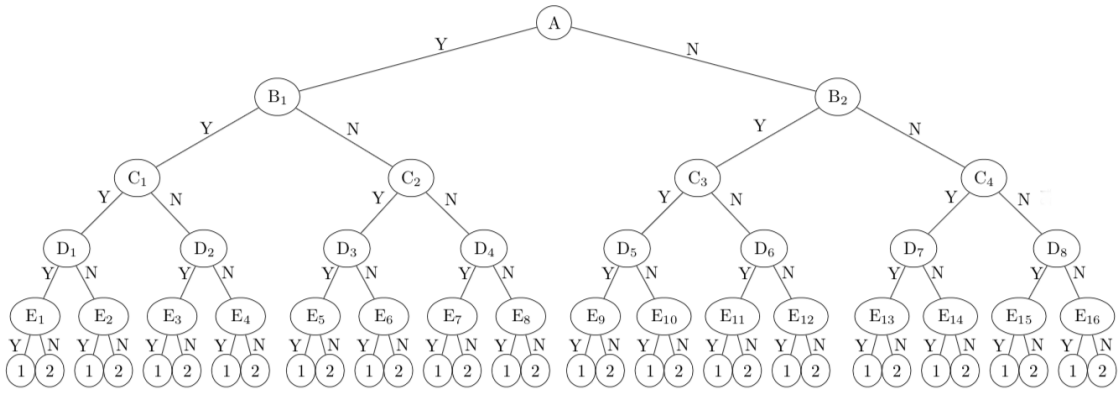

Figure 2-15: The reformulation of the Figure 2-14 FNN into a classification tree. The labels $A, B_1, \ldots, E_{16}$ are as follows:

- A is $\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1} < 0$.
- B_1 is $\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2} < 0$.
- B_2 is $\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2} < 0$.
- C_1 is $b_{2,1} < 0$.
- C_2 is $(W_{2,1})_2 \times (\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2}) + b_{2,1} < 0$.
- C_3 is $(W_{2,1})_1 \times (\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1}) + b_{2,1} < 0$.
- C_4 is $(W_{2,1})_1 \times (\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1}) + (W_{2,1})_2 \times (\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2}) + b_{2,1} < 0$.
- D_1 is $b_{2,2} < 0$.
- D_2 is $b_{2,2} < 0$.
- D_3 is $(W_{2,2})_2 \times (\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2}) + b_{2,2} < 0$.
- D_4 is $(W_{2,2})_2 \times (\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2}) + b_{2,2} < 0$.
- D_5 is $(W_{2,2})_1 \times (\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1}) + b_{2,2} < 0$.
- D_6 is $(W_{2,2})_1 \times (\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1}) + b_{2,2} < 0$.
- D_7 is $(W_{2,2})_1 \times (\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1}) + (W_{2,2})_2 \times (\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2}) + b_{2,2} < 0$.
- D_8 is $(W_{2,2})_1 \times (\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1}) + (W_{2,2})_2 \times (\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2}) + b_{2,2} < 0$.
- E_1 is $b_{O,1} - b_{O,2} < 0$.
- E_2 is $((W_{O,1})_2 - (W_{O,2})_2) \times (b_{2,2}) + b_{O,1} - b_{O,2} < 0$.
- E_3 is $((W_{O,1})_1 - (W_{O,2})_1) \times (b_{2,1}) + b_{O,1} - b_{O,2} < 0$.

- E_4 is $((W_{O,1})_1 - (W_{O,1})_1) \times (b_{2,1}) + ((W_{O,1})_2 - (W_{O,2})_2) \times (b_{2,2}) + b_{O,1} - b_{O,2} < 0$.
- E_5 is $b_{O,1} - b_{O,2} < 0$.
- E_6 is $((W_{O,1})_2 - (W_{O,1})_2) \times ((W_{2,2})_2 \times (\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}) + b_{2,2}) + b_{O,1} - b_{O,2} < 0$.
- E_7 is $((W_{O,1})_1 - (W_{O,2})_1) \times ((W_{2,1})_2 \times (\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}) + b_{2,1}) + b_{O,1} - b_{O,2} < 0$.
- E_8 is $((W_{O,1})_1 - (W_{O,2})_1) \times ((W_{2,1})_2 \times (\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}) + b_{2,1})$
  $+ ((W_{O,1})_2 - (W_{O,2})_2) \times ((W_{2,2})_2 \times (\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}) + b_{2,2}) + b_{O,1} - b_{O,2} < 0$.
- E_9 is $b_{O,1} - b_{O,2} < 0$.
- E_{10} is $((W_{O,1})_1 - (W_{O,2})_1) \times ((W_{2,2})_1 \times (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}) + b_{2,2}) + b_{O,1} - b_{O,2} < 0$.
- E_{11} is $((W_{O,1})_1 - (W_{O,2})_1) \times ((W_{2,1})_1 \times (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}) + b_{2,1}) + b_{O,1} - b_{O,2} < 0$.
- E_{12} is $((W_{O,1})_1 - (W_{O,2})_1) \times ((W_{2,1})_1 \times (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}) + b_{2,1})$
  $+ ((W_{O,1})_1 - (W_{O,2})_1) \times ((W_{2,2})_1 \times (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}) + b_{2,2}) + b_{O,1} - b_{O,2} < 0$.
- E_{13} is $b_{O,1} - b_{O,2} < 0$.
- E_{14} is $((W_{O,1})_2 - (W_{O,2})_2) \times ((W_{2,2})_1 \times (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1})$
  $+ (W_{2,2})_2 \times (\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}) + b_{2,2}) + b_{O,1} - b_{O,2} < 0$.
- E_{15} is $((W_{O,1})_1 - (W_{O,2})_1) \times ((W_{2,1})_1 \times (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1})$
  $+ (W_{2,1})_2 \times (\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}) + b_{2,1}) + b_{O,1} - b_{O,2} < 0$.
- E_{16} is $((W_{O,1})_1 - (W_{O,2})_1) \times ((W_{2,1})_1 \times (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}) + (W_{2,1})_2 \times (\mathbf{W}_{1,2}^T \mathbf{x} +$
  $b_{1,2}) + b_{2,1}) + ((W_{O,1})_2 - (W_{O,2})_2) \times ((W_{2,2})_1 \times (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1})$
  $+ (W_{2,2})_2 \times (\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}) + b_{2,2}) + b_{O,1} - b_{O,2} < 0$.

We start by defining the first split of the tree as

$$\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1} < 0,$$

and then both splits at depth 2 of the tree as

$$\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2} < 0.$$

Through these splits, we model the first hidden layer of the neural network as a classification tree. However, at depth 3 we start modeling the second hidden layer with splits in the tree, so we must define $\mathbf{y}_1$ values as linear functions of $\mathbf{x}$ based on the path taken down the tree. For example, for an input $\mathbf{x}$ to get to node C_1, it must have taken the Y branch at A and the Y branch at B_1, which means that if it were input into the neural network it would have a first hidden layer output of $(0,0)^T$. Thus, the split at C_1 is defined as

$$\mathbf{W}_{2,1}^T\mathbf{y}_1 + b_{2,1} < 0 \iff \mathbf{W}_{2,1}^T(0,0)^T + b_{2,1} < 0 \iff b_{2,1} < 0.$$

Next, for an input $\mathbf{x}$ to get to node C_2, it must have taken the Y branch at A and the N branch at B_1, which means that if it were input into the neural network, it would have a first hidden layer output of $(0, \mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2})^T$. Thus, the split at C_2 is defined as follows:

$$\mathbf{W}_{2,1}^T\mathbf{y}_1 + b_{2,1} < 0 \iff \mathbf{W}_{2,1}^T(0, \mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2})^T + b_{2,1} < 0$$

$$\iff (W_{2,1})_2 \times (\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2}) + b_{2,1} < 0.$$

This process continues for all the remaining split nodes of the tree. After that, we

must find which classification value the network assigns to the input. Based on our construction of splits at depth 5, an input takes the Y path if the network output is $(0, 1)^T$, indicating a classification value k of 2. Likewise, an input takes the N path if the network output is $(1, 0)^T$, indicating a classification value k of 1. We therefore assign these values to the corresponding leaf nodes of the tree, completing the construction.

2.4.4 Rectified Linear Unit Regression FNNs and ORT-Hs

If the network is a regression neural network with the rectified linear unit activation function $\mathcal{N}_2$ instead of a classification neural network, meaning it outputs $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O \in \mathbb{R}^q$, we are also able to build a regression tree $\mathcal{T}_2$ to make the same predictions as the neural network. We build the same decision tree as in the proof of Theorem 2 in Section 2.4.3 by building subtrees up until subtree $\mathcal{T}_{2,L}(\mathbf{y}_{L-1})$, the subtree with splits based on weights and biases $\mathbf{W}_{L,i}, b_{L,i}$, $i = 1, \ldots, N_L$. This results in a tree of depth $\sum_{\ell=1}^{L} N_\ell$. Then, for each leaf node of this tree we assign the linear function $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$ as the output value, where by the construction of the tree $\mathbf{y}_L$ is a linear function of $\mathbf{x}$. Through this process, $\mathcal{T}_2$ calculates the same output as $\mathcal{N}_2$ given an input vector $\mathbf{x}$. Since $\mathcal{T}_2$ is a regression tree with hyperplane splits, it follows that an ORT-H has at least as good accuracy as $\mathcal{T}_2$, completing this extension of Theorem 2. An example of the final subtree is depicted in Figure 2-16.

2.5 Convolutional Neural Networks and Optimal Trees

In this section, we construct an OCT-H (ORT-H) that can classify training data at least as well as a classification (regression) convolutional neural network (CNN) with

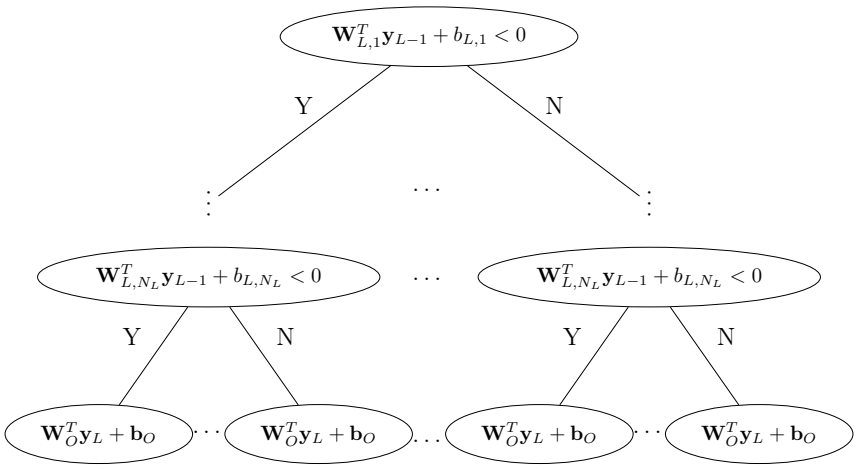

Figure 2-16: Subtree $\mathcal{T}_{2,L}(\mathbf{y}_{L-1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $\sum_{\ell=1}^{L} N_\ell$. Note that the leaves have linear functions $\mathbf{W}_O^T\mathbf{y}_L + \mathbf{b}_O$ as outputs.

perceptron or Rectified Linear Unit activation functions.

2.5.1 Perceptron Classification CNNs and OCT-Hs

For the case of a CNN with the perceptron activation function, as in Section 2.4.1, the output of the neural network follows deterministically given the output of the first hidden layer, regardless of convolutional or pooling layers. Thus, the fact that an OCT-H of depth N_1 can classify training data at least as well as a given classification CNN with the perceptron activation function, max pooling layers, and N_1 nodes in the first hidden layer follows directly from Theorem 1.

2.5.2 Perceptron Regression CNNs and ORT-Hs

One can also extend the result in Section 2.5.1 to the case of a regression CNN with the perceptron activation function, where $\phi_O(\mathbf{y}_L) \in \mathbb{R}^q$. To do this, note that because the first hidden layer can output at most 2^{N_1} unique vectors there are only

2^{N_1} possible output values of the network. In this case, one can modify the proof of Theorem 1 by assigning these 2^{N_1} unique values to the leaf nodes of the decision tree in the place of classification values. With this adjustment, extending the above proof to regression CNNs with perceptron activation functions is straightforward.

2.5.3 Rectified Linear Unit Classification CNNs and OCT-Hs

While Theorem 1 could be applied directly in Section 2.5.1, the addition of pooling layers implies that the OCT-H construction given in Theorem 2 does not apply to CNNs with ReLU activation functions. This is because while the proof in Section 2.4.3 takes into account functions of the form

$$\max(\mathbf{W}^T\mathbf{x} + b, 0)$$

in its construction, pooling layers involve functions of the form

$$\max(\max(\mathbf{W}_1^T\mathbf{x} + b_1, 0), \ldots, \max(\mathbf{W}_n^T\mathbf{x} + b_n, 0)),$$

which requires additional steps to be transformed into linear splits at branch nodes. The next theorem illustrates the OCT-H construction for this case.

Theorem 3. *An OCT-H with depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$ can classify training data at least as well as a given CNN with the rectified linear unit activation function (2.6), max pooling layers as defined in (2.8), and output function defined in Eq. (2.7).*

Proof. Our proof is constructive. We are given a CNN $\mathcal{N}_4$ as described in Section 2.3.2 with the following characteristics:

- The rectified linear unit activation function defined in (2.6).

- The output function as defined in (2.7).
- L hidden layers and one output layer, indexed $\ell = 1, \ldots, L$.
- Max pooling layers as defined in (2.8).
- N_ℓ nodes in each hidden layer, indexed $i = 1, \ldots, N_\ell$.
- Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$.
- L_c pooling layers.
- R_ℓ nodes in pooling layer, indexed $i = 1, \ldots, R_\ell$.
- Each node in a pooling layer takes as input a set $S_{\ell,i}$ of nodes from the previous convolutional layer with the properties defined in Section 2.3.2.
- q nodes in the output layer.

We now construct a decision tree $\mathcal{T}_4$ with hyperplane splits of maximum depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$ that makes the same predictions as $\mathcal{N}_4$. It follows that an OCT-H of maximum depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$ has at least the same classification accuracy as $\mathcal{N}_4$.

First, we construct a subtree $\mathcal{T}_{4,1}$ with splits based on the output of first node in the first pooling layer $P_{1,1}$ as defined in Eq. (2.8). Without loss of generality, assume that $S_{1,1} = \{1, \ldots, k_1\}$. Then the first split of $\mathcal{T}_{4,1}$ is

$$(\mathbf{W}_{1,1} - \mathbf{W}_{1,2})^T \mathbf{x} + (b_{1,1} - b_{1,2}) < 0. \tag{2.11}$$

It is depicted in Figure 2-17, and determines whether the maximum among the first two hyperplanes is $\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}$ or $\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}$.

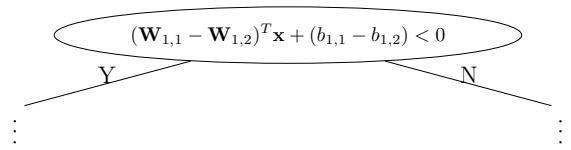

Figure 2-17: The first split of classification tree $\mathcal{T}_{4,1}$.

If Inequality (2.11) holds, then we next to check whether

$$(\mathbf{W}_{1,2} - \mathbf{W}_{1,3})^T\mathbf{x} + (b_{1,2} - b_{1,3}) < 0,$$

while if Inequality (2.11) does not hold, we need to check whether

$$(\mathbf{W}_{1,1} - \mathbf{W}_{1,3})^T\mathbf{x} + (b_{1,1} - b_{1,3}) < 0,$$

thus forming the hyperplane splits for $\mathcal{T}_{4,1}$ at depth 2 depicted in Figure 2-18.

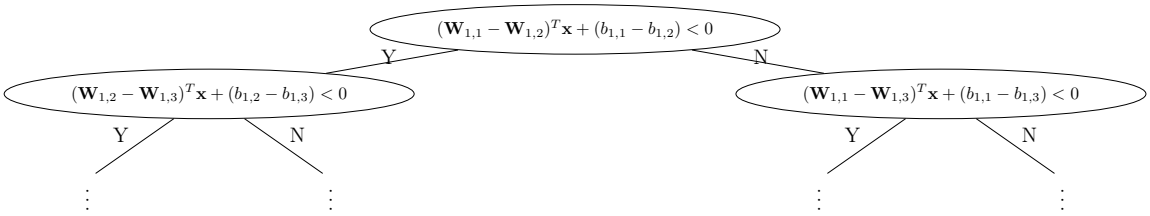

Figure 2-18: The decision tree $\mathcal{T}_{4,1}$ we are building up to depth 2. The identity of the maximum hyperplane so far is from left to right: 3, 2, 3 and 1.

At depth $k_1 - 1$ tree $\mathcal{T}_{4,1}$ will determine the identity of i^*

$$i^* = \arg \max_{i=1,\ldots,k_1} (\mathbf{W}_{1,i}^T\mathbf{x} + b_{1,i}).$$

After the $(k_1 - 1)$th hyperplane split, we append to $\mathcal{T}_{4,1}$ the final hyperplane split

$$\mathbf{W}_{1,i^*}^T\mathbf{x} + b_{1,i^*} < 0,$$

at the appropriate node. The tree $\mathcal{T}_{4,1}$ is depicted in Figure 2-19. It has depth k_1 and the construction is complete.

Following this, we repeat this process for all nodes in the first hidden and pooling layers, building a tree of depth N_1 that can be used to calculate the output of the pooling layer $\mathbf{P}_1$. In each of the branches at depth N_1 of this new subtree we implicitly calculate the corresponding $\mathbf{P}_1$ values. For example, the first branch corresponds to $(0, \ldots, 0)^T$, the second corresponds to $(0, \ldots, 0, \max_{i=(R_1-1)k_1+1,\ldots,R_1k_1}(\mathbf{W}_{1,i}^T\mathbf{x}+b_{1,i}))^T$, etc.

We then model the second hidden layer of $\mathcal{N}_4$ by constructing after each branch a new subtree of depth k_2 as in Figure 2-20, but using the corresponding value of $\mathbf{P}_1$ instead of $\mathbf{x}$ in the splits. We call this subtree $\mathcal{T}_{4,2}(\mathbf{P}_1)$, as it is the first of the subtrees created based on the neural network hidden layer two weights and $\mathbf{P}_1$.

We use the same process for adapting the remaining convolutional layer – pooling layer pairs. After we have finished modeling those pairs of layers, the process of modeling the remaining hidden layers and the output layer nodes as a decision tree is exactly the same as it was in the proof of Theorem 2. This once again results in the construction of a tree with splits based deterministically on $\mathbf{x}$.

By the construction of $\mathcal{T}_4$, $\mathcal{T}_4$ calculates the same output as $\mathcal{N}_4$ given an input vector $\mathbf{x}$. Since $\mathcal{T}_4$ is a decision tree with hyperplane splits, it follows that an optimal tree has at least as good accuracy as $\mathcal{T}_4$, proving the theorem. □

2.5.4 Rectified Linear Unit Regression CNNs and ORT-Hs

If the given CNN is a regression neural network with the rectified linear unit activation function $\mathcal{N}_4$ instead of a classification neural network, meaning it outputs $\mathbf{W}_O^T\mathbf{y}_L + \mathbf{b}_O \in \mathbb{R}^q$, we are also able to build a regression tree $\mathcal{T}_4$ to make the same

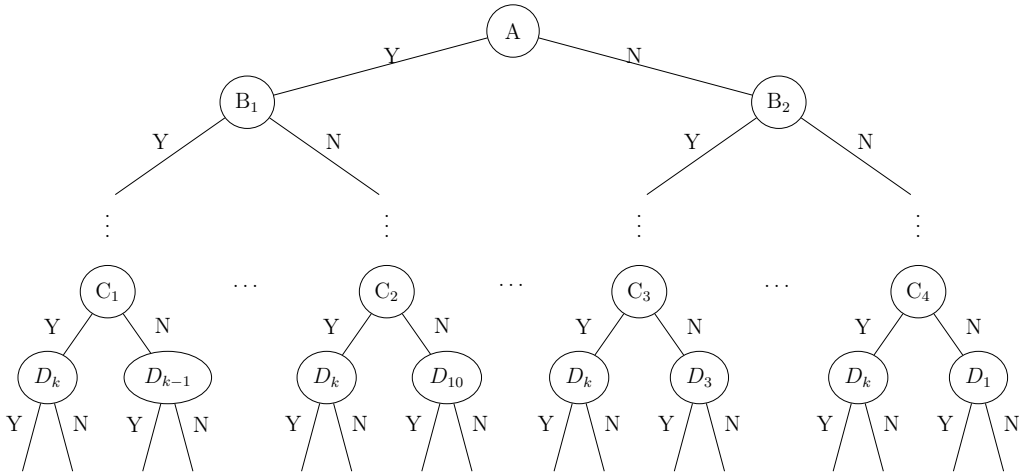

Figure 2-19: The resulting tree $\mathcal{T}_{4,1}$. The labels A, $B_1, \ldots, C_4$ and D_{i^*} are as follows:

- A is $(\mathbf{W}_{1,1} - \mathbf{W}_{1,2})^T\mathbf{x} + (b_{1,1} - b_{1,2}) < 0$
- B_1 is $(\mathbf{W}_{1,2} - \mathbf{W}_{1,3})^T\mathbf{x} + (b_{1,2} - b_{1,3}) < 0$
- B_2 is $(\mathbf{W}_{1,1} - \mathbf{W}_{1,3})^T\mathbf{x} + (b_{1,1} - b_{1,3}) < 0$
- C_1 is $(\mathbf{W}_{1,k_1-1} - \mathbf{W}_{1,k_1})^T\mathbf{x} + (b_{1,k_1-1} - b_{1,k_1}) < 0$
- C_2 is $(\mathbf{W}_{1,10} - \mathbf{W}_{1,k_1})^T\mathbf{x} + (b_{1,10} - b_{1,k_1}) < 0$
- C_3 is $(\mathbf{W}_{1,3} - \mathbf{W}_{1,k_1})^T\mathbf{x} + (b_{1,3} - b_{1,k_1}) < 0$
- C_4 is $(\mathbf{W}_{1,1} - \mathbf{W}_{1,k_1})^T\mathbf{x} + (b_{1,1} - b_{1,k_1}) < 0$
- D_{i^*} is $\mathbf{W}_{1,i^*}^T\mathbf{x} + b_{1,i^*} < 0$.

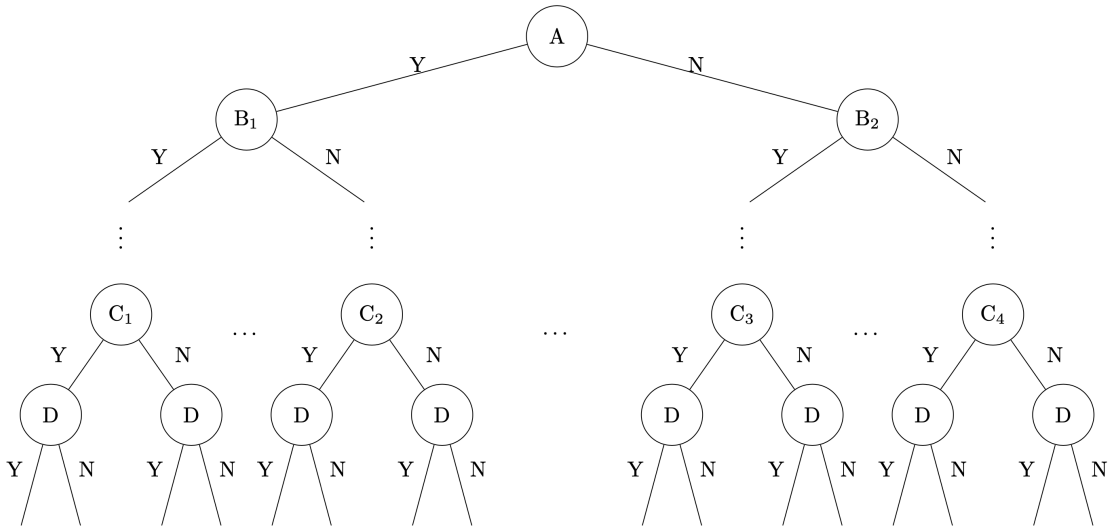

Figure 2-20: The subtree $\mathcal{T}_{4,2}(\mathbf{P}_1)$ up to depth k_2. The labels A, B_1, ..., D are as follows:

- A is $(\mathbf{W}_{2,1} - \mathbf{W}_{2,2})^T \mathbf{P}_1 + (b_{2,1} - b_{2,2}) < 0$
- B_1 is $(\mathbf{W}_{2,2} - \mathbf{W}_{2,3})^T \mathbf{P}_1 + (b_{2,2} - b_{2,3}) < 0$
- B_2 is $(\mathbf{W}_{2,1} - \mathbf{W}_{2,3})^T \mathbf{P}_1 + (b_{2,1} - b_{2,3}) < 0$
- C_1 is $(\mathbf{W}_{2,k_2-1} - \mathbf{W}_{2,k_2})^T \mathbf{P}_1 + (b_{2,k_2-1} - b_{2,k_2}) < 0$
- C_2 is $(\mathbf{W}_{2,10} - \mathbf{W}_{2,k_2})^T \mathbf{P}_1 + (b_{2,10} - b_{2,k_2}) < 0$
- C_3 is $(\mathbf{W}_{2,3} - \mathbf{W}_{2,k_2})^T \mathbf{P}_1 + (b_{2,3} - b_{2,k_2}) < 0$
- C_4 is $(\mathbf{W}_{2,1} - \mathbf{W}_{2,k_2})^T \mathbf{P}_1 + (b_{2,1} - b_{2,k_2}) < 0$
- D is $\mathbf{W}_{2,i^*}^T \mathbf{P}_1 + b_{2,i^*} < 0$

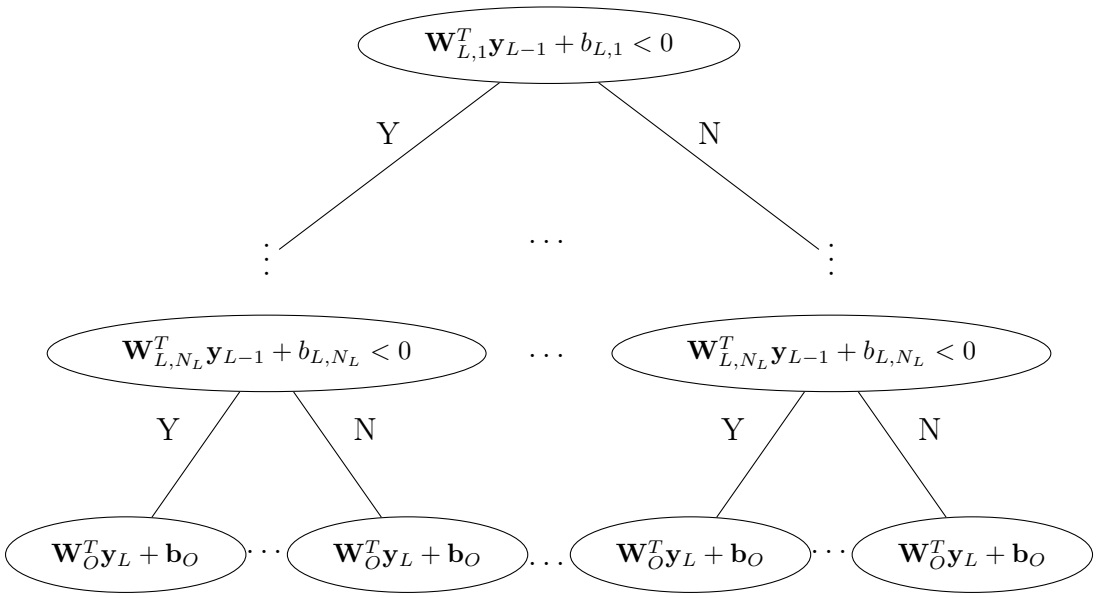

Figure 2-21: Subtree $\mathcal{T}_{4,L}(\mathbf{y}_{L-1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $\sum_{\ell=1}^{L} N_\ell$. Note that the leaves have linear functions $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$ as outputs.

predictions as the neural network. We build the same decision tree as in the proof of Theorem 3 in Section 2.5.3 by building subtrees up until subtree $\mathcal{T}_{4,L}(\mathbf{y}_{L-1})$, the subtree with splits based on weights and biases $\mathbf{W}_{L,i}, b_{L,i}$, $i = 1, \ldots, N_L$. This results in a tree of depth $\sum_{\ell=1}^{L} N_\ell$. Then, for each leaf node of this tree we assign the linear function $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$ as the output value, where by the construction of the tree $\mathbf{y}_L$ is a linear function of $\mathbf{x}$. Through this process, $\mathcal{T}_4$ calculates the same output as $\mathcal{N}_4$ given an input vector $\mathbf{x}$. Since $\mathcal{T}_4$ is a regression tree with hyperplane splits, it follows that an ORT-H has at least as good accuracy as $\mathcal{T}_4$, completing this extension of Theorem 4. An example of the final subtree is depicted in Figure 2-21.

2.6 Recurrent Neural Networks and Optimal Trees

In this section, we construct an OCT-H (ORT-H) that can be used to classify training data at least as well as a classification (regression) recurrent neural network (RNN) with perceptron or Rectified Linear Unit activation functions.

2.6.1 Perceptron Classification RNNs and OCT-Hs

The key result in this section is as follows.

Theorem 4. *An OCT-H with maximum depth $T^* \times N_1$ can classify sequential data with T^* terms per sequence in a training set at least as well as a given classification RNN with the perceptron activation function, one hidden layer containing N_1 nodes, and q nodes in the output layer.*

Proof. Our proof is constructive. We are given a classification RNN $\mathcal{N}_5$ as described in Section 2.3.3 with the following specifications:

- The perceptron activation function as defined in (2.5).
- Output function $\phi_O(\mathbf{y}_O) : [0,1]^q \rightarrow [0,1]^q$.
- One hidden layer and one output layer.
- N_1 nodes in the hidden layer, indexed $i = 1, \ldots, N_1$.
- q nodes in the output layer, indexed $i = 1, \ldots, q$.
- Node $n_{1,i}$ characterized by $\mathbf{W}_{g,i}, \mathbf{W}_{h,i}, b_{1,i}$.

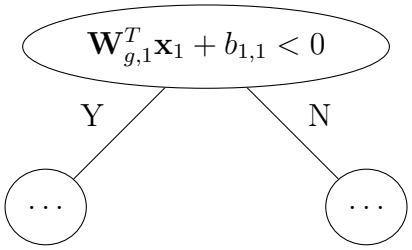

Figure 2-22: The first split of decision tree $\mathcal{T}_5$.

Using $\mathcal{N}_5$, we construct a decision tree $\mathcal{T}_5$ with hyperplanes and maximum depth $T^* \times N_1$ that can be used to make the same predictions as the network. It follows that an OCT-H of maximum depth $T^* \times N_1$ has at least the same classification accuracy as $\mathcal{N}_5$.

First, for ease of notation we define $\mathbf{x}^*$ as

$$\mathbf{x}^* = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T^*})^T, \tag{2.12}$$

which is the concatenation of all the vectors in the input sequence $(\mathbf{x}_t)_{t\in\{1,\ldots,T^*\}}$. Then we define the first split of $\mathcal{T}_5$ as

$$(\mathbf{W}_{g,1}, \mathbf{0}, \ldots, \mathbf{0})\mathbf{x}^* + b_{1,1} = \mathbf{W}_{g,1}^T\mathbf{x}_1 + b_{1,1} < 0, \tag{2.13}$$

where $(\mathbf{W}_{g,1}, \mathbf{0}, \ldots, \mathbf{0})$ is the concatenation of $\mathbf{W}_{g,1}$ horizontally with $T^* - 1$ zero vectors $\mathbf{0}$ with the same dimensions as $\mathbf{W}_{g,1}$. This results in the simple split seen in Figure 2-22.

Independent of whether inequality (2.13) is satisfied or not, the second split is given by

$$(\mathbf{W}_{g,2}, \mathbf{0}, \ldots, \mathbf{0})\mathbf{x}^* + b_{1,2} = \mathbf{W}_{1,2}^T\mathbf{x}_1 + b_{1,2} < 0,$$

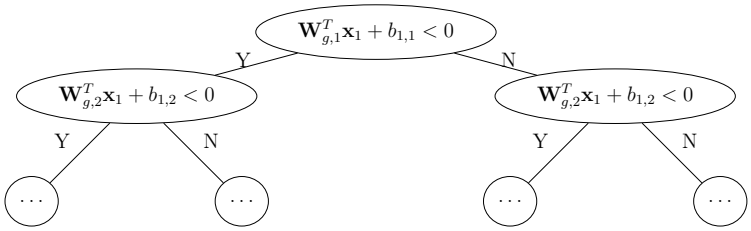

Figure 2-23: The tree $\mathcal{T}_5$ up to depth 2.

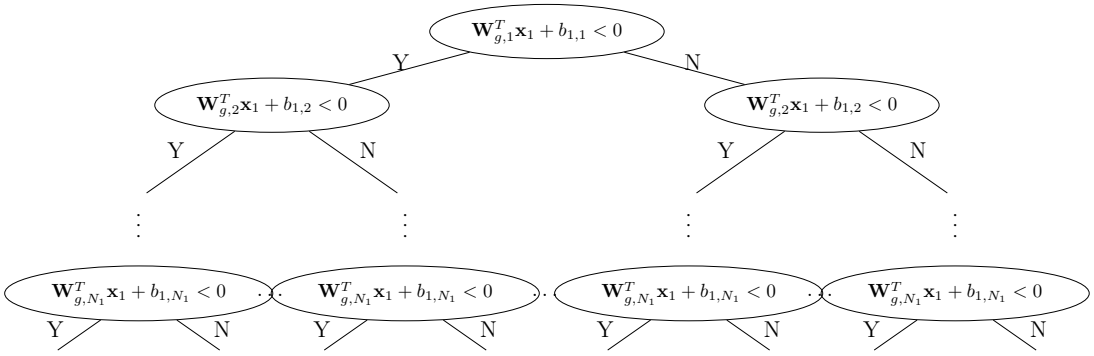

Figure 2-24: The decision tree $\mathcal{T}_5$ we are building up to depth N_1.

where $(\mathbf{W}_{g,2}, \mathbf{0}, \ldots, \mathbf{0})$ is the concatenation of $\mathbf{W}_{g,2}$ horizontally with $T^* - 1$ zero vectors $\mathbf{0}$ with the same dimensions as $\mathbf{W}_{g,2}$. Figure 2-23 provides a visualization of the new branches we added to the tree in Figure 2-22.

We continue this process for all N_1 nodes in the first hidden layer, building a decision tree of depth N_1, with every split at depth N_1 being given by

$$\mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1} < 0.$$

The resultant subtree is shown in Figure 2-24. This is the subtree that simulates the output of the hidden layer $\mathcal{N}_5$ after the first time step.

After depth N_1, there are 2^{N_1} branches, as shown in Figure 2-24. Note that there are 2^{N_1} possible output vectors $\mathbf{y}_{1,1}$ of the first hidden layer of $\mathcal{N}_5$,

$$(0,\ldots,0)^T,(1,\ldots,0)^T,\ldots,(1,\ldots,1)^T.$$

These 2^{N_1} possible values of $\mathbf{y}_{1,1}$ correspond to the 2^{N_1} branches in the tree $\mathcal{T}_5$ shown in Figure 2-24. In each of these branches we have implicitly calculated the corresponding $\mathbf{y}_{1,1}$ values. For example, the first branch corresponds to $(0,\ldots,0)^T$, the second corresponds to $(0,\ldots,0,1)^T$, etc.

We then model $\mathcal{N}_5$ at the second time step by constructing after each branch a new subtree of depth N_1 as in Figure 2-25, but with the corresponding value of $\mathbf{y}_{1,1}$ being used as a constant value in addition to $\mathbf{x}_2$. Thus, the first split of this new subtree is

$$(\mathbf{0},\mathbf{W}_{g,1},\mathbf{0},\ldots,\mathbf{0})\mathbf{x}^* + \mathbf{W}_{h,1}^T\mathbf{y}_{1,1} + b_{1,1} = \mathbf{W}_{g,1}^T\mathbf{x}_2 + \mathbf{W}_{h,1}^T\mathbf{y}_{1,1} + b_{1,1} < 0.$$

This process continues for the remaining nodes of the first hidden layer shown in Figure 2-25, resulting in the subtree in Figure 2-25. In this subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ we substitute in the corresponding binary vector $\mathbf{y}_{1,1}$. For example, if $\mathbf{y}_{1,1} = (0,\ldots,0,1)^T$, then subtree $\mathcal{T}_{5,2}((0,\ldots,0,1)^T)$ is depicted in Figure 2-26.

Given that at each branch we know exactly $\mathbf{y}_{1,1}$, $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ is a decision tree where all inequalities are explicitly written as linear functions of $\mathbf{x}^*$. Continuing in this way we model the output vector $\mathbf{y}_{t,1}$ of the tth time step of $\mathcal{N}_5$ as a classification tree by defining $\mathbf{y}_{t,1}$ based on the path taken down the previous subtree. At the tth subtree, the weights of the $\mathbf{x}_t$ vector contained in the vector $\mathbf{x}^*$ are $\mathbf{W}_{g,1}$, and the rest are zero.

To complete the construction of $\mathcal{T}_5$, we need to assign a classification value for

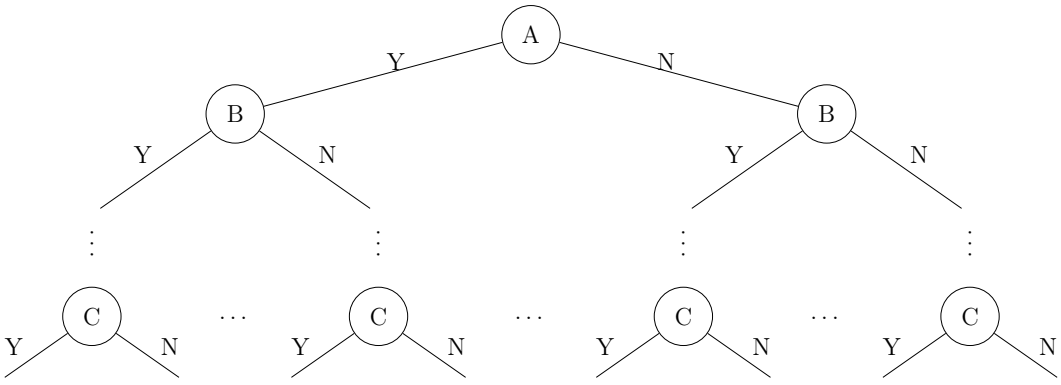

Figure 2-25: Subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ of depth N_1 is concatenated to the corresponding branch of the subtree depicted in Figure 2-24, resulting in a subtree of depth $2N_1$. The splits are as follows:

- A is $\mathbf{W}_{g,1}^T \mathbf{x}_2 + \mathbf{W}_{h,1}^T \mathbf{y}_{1,1} + b_{1,1} < 0$.
- B is $\mathbf{W}_{g,2}^T \mathbf{x}_2 + \mathbf{W}_{h,2}^T \mathbf{y}_{1,1} + b_{1,2} < 0$.
- C is $\mathbf{W}_{g,N_1}^T \mathbf{x}_2 + \mathbf{W}_{h,N_1}^T \mathbf{y}_{1,1} + b_{1,N_1} < 0$.

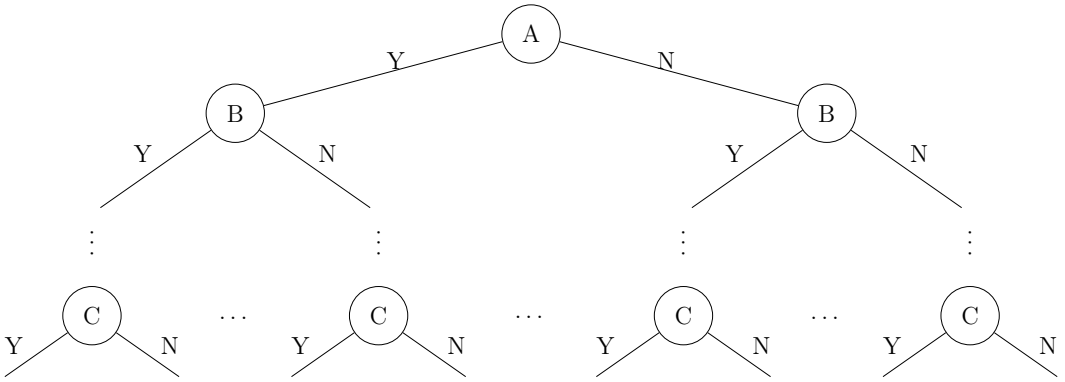

Figure 2-26: The resulting subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ for $\mathbf{y}_{1,1} = (0, \ldots, 0, 1)^T$. The labels of A, B, C are as follows:

- A is $\mathbf{W}_{g,1}^T \mathbf{x}_2 + \mathbf{W}_{h,1}^T (0, \ldots, 0, 1)^T + b_{1,1} < 0$.
- B is $\mathbf{W}_{g,2}^T \mathbf{x}_2 + \mathbf{W}_{h,2}^T (0, \ldots, 0, 1)^T + b_{1,2} < 0$.
- C is $\mathbf{W}_{g,N_1}^T \mathbf{x}_2 + \mathbf{W}_{h,N_1}^T (0, \ldots, 0, 1)^T + b_{1,N_1} < 0$.

every leaf of $\mathcal{T}_5$. At time step T^*, there are 2^{N_1} possible binary vectors that the first hidden layer of $\mathcal{N}_5$ could output. These 2^{N_1} vectors, by our construction of $\mathcal{T}_5$, exactly correspond to the 2^{N_1} leaves of the final subtree of $\mathcal{T}_5$. Given $\mathbf{y}_{T^*,1}^r$, the output of the first hidden layer associated with leaf node r, the final prediction of $\mathcal{N}_5$ will be $k(\mathbf{y}_{T^*,1}^r)$, which is calculated deterministically given the $\mathbf{y}_{T^*,1}^r$ vector and the $\mathbf{W}_{O,i}$, $b_{O,i}$ values by using the process outlined in Section 2.3.3. In every node r of the tree we assign the classification value $k(\mathbf{y}_{T^*,1}^r)$.

We next show that the output of $\mathcal{T}_5$ is the same as the output of $\mathcal{N}_5$ for input data sequence $\mathbf{x}_t$, $t = 1, \ldots, T^*$. To see this, if $\mathbf{x}_t$ is input into $\mathcal{N}_5$, the hidden layer outputs $\mathbf{y}_{1,1}(\mathbf{x}_1)$ at the first time step, $\mathbf{y}_{2,1}(\mathbf{x}_2, \mathbf{y}_{1,1})$ at the second time step, and so on, until at last the sequence is assigned classification value $k(\mathbf{y}_{T^*,1})$. However, by the construction of the tree, the point $\mathbf{x}^*$ is sorted down the paths corresponding to

$\mathbf{y}_{1,1}(\mathbf{x}_1)$, $\mathbf{y}_{2,1}(\mathbf{x}_2, \mathbf{y}_{1,1})$, and so on, until it is sorted to the leaf node where $\mathbf{y}^r_{T^*,1} = \mathbf{y}_{T^*,1}(\mathbf{x}^*)$, and is once again assigned $k(\mathbf{y}^r_{T^*,1}) = k(\mathbf{y}_{T^*,1})$ by construction. Thus, for a given data sequence $\mathbf{x}_t$, $t = 1, \ldots, T^*$, the network and the tree predict the same classification value.

Since an OCT-H does at least as well as $\mathcal{T}_5$ in classifying the training data, it must do at least as well as $\mathcal{N}_5$ too. Thus, by construction, we have that an optimal decision tree with maximum depth $T^* \times N_1$ can classify data in a training set at least as well as the given FNN with the perceptron activation function and one hidden layer with N_1 nodes in it, completing the proof of the theorem. □

We next present an example of how to perform the above procedure. The neural network we are working with was trained on data $((\mathbf{X}_t)_{t=1,\ldots,T^*}, \mathbf{o})$, where $T^* = 2$, and is shown in Figure 2-27. The resultant decision tree is shown in Figure 2-28.

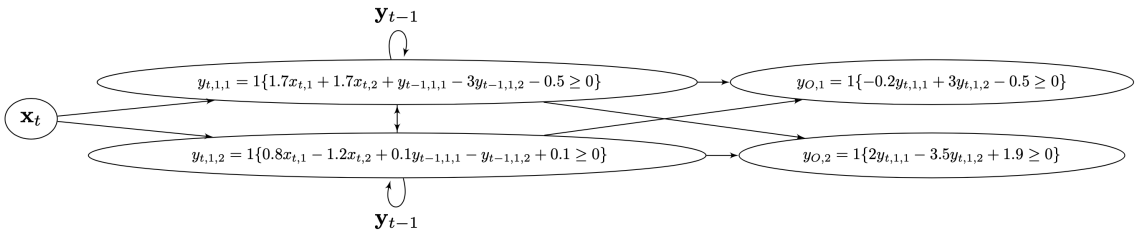

Figure 2-27: A RNN with the perceptron activation function.

The reformulation process is as follows. We assume the input to the tree is of the form

$$\mathbf{x}^* = (x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2})^T$$

We define the first split of the decision tree as

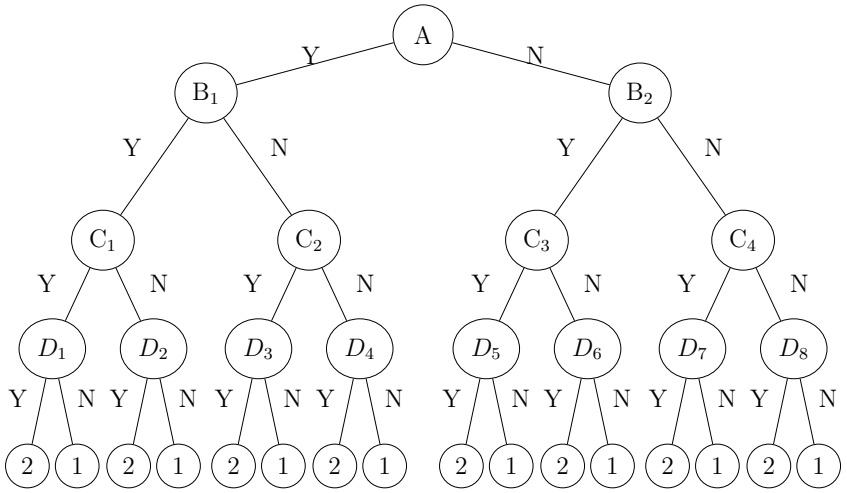

Figure 2-28: The resulting tree $\mathcal{T}_5$. The labels A, $B_1, \ldots, D_8$ are as follows:

- A is $1.7x_{1,1} + 1.7x_{1,2} + 0x_{2,1} + 0x_{2,2} - 0.5 < 0$
- B_1 is $0.8x_{1,1} - 1.2x_{1,2} + 0x_{2,1} + 0x_{2,2} + 0.1 < 0$
- B_2 is $0.8x_{1,1} - 1.2x_{1,2} + 0x_{2,1} + 0x_{2,2} + 0.1 < 0$
- C_1 is $0x_{1,1} + 0x_{1,2} + 1.7x_{2,1} + 1.7x_{2,2} + 1 \cdot 0 - 3 \cdot 0 - 0.5 < 0$
- C_2 is $0x_{1,1} + 0x_{1,2} + 1.7x_{2,1} + 1.7x_{2,2} + 1 \cdot 0 - 3 \cdot 1 - 0.5 < 0$
- C_3 is $0x_{1,1} + 0x_{1,2} + 1.7x_{2,1} + 1.7x_{2,2} + 1 \cdot 1 - 3 \cdot 0 - 0.5 < 0$
- C_4 is $0x_{1,1} + 0x_{1,2} + 1.7x_{2,1} + 1.7x_{2,2} + 1 \cdot 1 - 3 \cdot 1 - 0.5 < 0$
- D_1 is $0x_{1,1} - 0x_{1,2} + 0.8x_{2,1} + 1.2x_{2,2} + 0.1 \cdot 0 - 1 \cdot 0 + 0.1 < 0$
- D_2 is $0x_{1,1} - 0x_{1,2} + 0.8x_{2,1} + 1.2x_{2,2} + 0.1 \cdot 0 - 1 \cdot 0 + 0.1 < 0$
- D_3 is $0x_{1,1} - 0x_{1,2} + 0.8x_{2,1} + 1.2x_{2,2} + 0.1 \cdot 0 - 1 \cdot 1 + 0.1 < 0$
- D_4 is $0x_{1,1} - 0x_{1,2} + 0.8x_{2,1} + 1.2x_{2,2} + 0.1 \cdot 0 - 1 \cdot 1 + 0.1 < 0$
- D_5 is $0x_{1,1} - 0x_{1,2} + 0.8x_{2,1} + 1.2x_{2,2} + 0.1 \cdot 1 - 1 \cdot 0 + 0.1 < 0$
- D_5 is $0x_{1,1} - 0x_{1,2} + 0.8x_{2,1} + 1.2x_{2,2} + 0.1 \cdot 1 - 1 \cdot 0 + 0.1 < 0$
- D_7 is $0x_{1,1} - 0x_{1,2} + 0.8x_{2,1} + 1.2x_{2,2} + 0.1 \cdot 1 - 1 \cdot 1 + 0.1 < 0$
- D_8 is $0x_{1,1} - 0x_{1,2} + 0.8x_{2,1} + 1.2x_{2,2} + 0.1 \cdot 1 - 1 \cdot 1 + 0.1 < 0$

$$11.7x_{1,1} + 1.7x_{1,2} + 0x_{2,1} + 0x_{2,2} - 0.5 < 0,$$

and then both splits at depth two as

$$0.8x_{1,1} - 1.2x_{1,2} + 0x_{2,1} + 0x_{2,2} + 0.1 < 0.$$

With these splits, we model the first time step in the RNN. However, at depth 3 we start modeling the second time step, so the $\mathbf{y}_{1,1}$ values must be taken into account. For example, for an input $\mathbf{x}^*$ to get to node C_1, it must have taken the Y branch at A and the Y branch at B_1, which means that if it were input into the neural network it would have $\mathbf{y}_{1,1} = (0, 0)^T$. Thus, the split at C_1 is defined as

$$0x_{1,1} + 0x_{1,2} + 1.7x_{2,1} + 1.7x_{2,2} + 1 \cdot 0 - 3 \cdot 0 - 0.5 < 0$$

Next, for an input $\mathbf{x}$ to get to node C_2, it must have taken the Y branch at A and the N branch at B_1, which means that if it were input into the neural network, it would have a first hidden layer output of $(0, 1)^T$. Thus, the split at C_2 is defined as follows:

$$0x_{1,1} + 0x_{1,2} + 1.7x_{2,1} + 1.7x_{2,2} + 1 \cdot 0 - 3 \cdot 1 - 0.5 < 0$$

This process continues for all the remaining split nodes of the tree. After that, we must find which classification value the network assigns to the input. Based on our construction of splits, after depth 4 we are able to find the value of $\mathbf{y}_{T^*,1}$ that the hidden layer of the neural network would output at the final time step. For example, an input $\mathbf{x}^*$ that takes the Y branch at both C_1 and D_1 must have $\mathbf{y}_{T^*,1} = (0, 0)^T$. We can then use this vector to find the appropriate output value to assign to the leaf

node. Continuing the previous example, in the neural network inputting $(0,0)^T$ into the output layer results in a network output of $(0,1)^T$, so we assign the class value 2 to the left-most leaf node. This process continues for all the leaf nodes of the tree, completing the construction.

2.6.2 Perceptron Regression RNNs and ORT-Hs

In the case where we have a regression RNN with the perceptron activation function function, meaning $\phi_O(\mathbf{y}_L) \in \mathbb{R}^q$, then because the first hidden layer can output at most 2^{N_1} unique vectors there are only 2^{N_1} possible output values of the network. In this case, one can modify the proof of Theorem 4 in Section 2.6.1 by assigning these 2^{N_1} unique values to the leaf nodes of the decision tree in the place of classification values in each of the final subtrees. With this adjustment, extending the above proof to regression RNNs with perceptron activation functions is straightforward.

2.6.3 Rectified Linear Unit Classification RNNs and OCT-Hs

In this section, we show that given a RNN with Rectified Linear Unit activation functions, we can construct an ORT-H that can be used to classify given training data at least as well as that network. The theorem is as follows.

Theorem 5. *An OCT-H with maximum depth $T^* \times N_1 + q - 1$ can classify sequential data with T^* terms per sequence in a training set at least as well as a given classification RNN with the ReLU activation function, one hidden layer containing N_1 nodes, and q nodes in the output layer.*

Proof. Our proof is constructive. We are given that we have a recurrent neural network $\mathcal{N}_6$ with the following specifications:

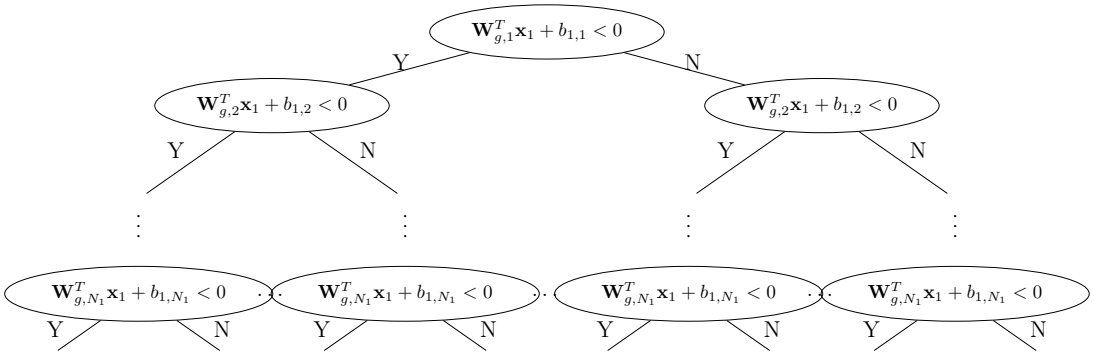

Figure 2-29: The decision tree $\mathcal{T}_6$ we are building up to depth N_1.

- The ReLU activation function.
- One hidden layer and one output layer.
- N_1 nodes in the hidden layer, indexed $i = 1, \ldots, N_1$.
- q nodes in the output layer, indexed $i = 1, \ldots, q$.
- Hidden layer node $n_{1,i}$ defined by $\mathbf{W}_{g,i}, \mathbf{W}_{h,i}, b_{1,i}$.

Using $\mathcal{N}_6$, we construct a decision tree $\mathcal{T}_6$ with hyperplanes and maximum depth $T^* \times N_1 + q - 1$ that makes the same predictions as $\mathcal{N}_6$. It follows that an optimal tree with hyperplanes of maximum depth $T^* \times N_1 + q - 1$ has at least the same classification accuracy as $\mathcal{N}_6$.

First, for ease of notation we define $\mathbf{x}^*$ as we did in Eq. (2.12). Then we build the tree up to depth N_1 exactly as we did in the proof in Section 2.6.1. This part of the tree is shown in Figure 2-29.

After depth N_1, there are 2^{N_1} branches, as shown in Figure 2-29. Note that there are 2^{N_1} possible output vectors $\mathbf{y}_{1,1}$ of the hidden layer of $\mathcal{N}_6$ at time step 1,

$$(0,\ldots,0)^T, (\mathbf{W}_{g,1}^T\mathbf{x}_1 + b_{1,1},\ldots,0)^T,\ldots,(\mathbf{W}_{g,1}^T\mathbf{x}_1 + b_{1,1},\ldots,\mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T,$$

as in each node of the first hidden layer $\mathcal{N}_6$ computes

$$(\mathbf{y}_{1,1})_i = \max\{\mathbf{W}_{g,i}^T\mathbf{x}_1 + b_{1,i}, 0\},\ i = 1,\ldots,N_1$$

at time step 1.

These 2^{N_1} possible values of $\mathbf{y}_{1,1}$ correspond to the 2^{N_1} branches in the tree $\mathcal{T}_6$ shown in Figure 2-29. In each of these branches we have implicitly calculated the corresponding $\mathbf{y}_{1,1}$ values. For example, the first branch corresponds to $(0,\ldots,0)^T$, the second corresponds to $(0,\ldots,0,\mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T$, etc.

We then model $\mathcal{N}_6$ at the second time step by constructing after each branch a new subtree of depth N_1 as in Figure 2-25, but with the corresponding value of $\mathbf{y}_{1,1}$ being used as a constant value in addition to $\mathbf{x}_2$. Thus, the first split of this new subtree is

$$(\mathbf{0},\mathbf{W}_{g,1},\ldots,\mathbf{0})^T\mathbf{x}^* + \mathbf{W}_{h,1}^T\mathbf{y}_{1,1} + b_{1,1} = \mathbf{W}_{g,1}^T\mathbf{x}_2 + \mathbf{W}_{h,1}^T\mathbf{y}_{1,1} + b_{1,1} < 0.$$

This process continues for the remaining nodes of the first hidden layer, resulting in the subtree shown in Figure 2-30.

In the subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ in Figure 2-30 we substitute in the corresponding binary vector $\mathbf{y}_{1,1}$. For example, if $\mathbf{y}_{1,1} = (0,\ldots,0,\mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T$, then subtree $\mathcal{T}_{6,2}((0,\ldots,0,\mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T)$ is depicted in Figure 2-31.

Given that at each branch we know exactly $\mathbf{y}_{1,1}$, $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ is a decision tree where

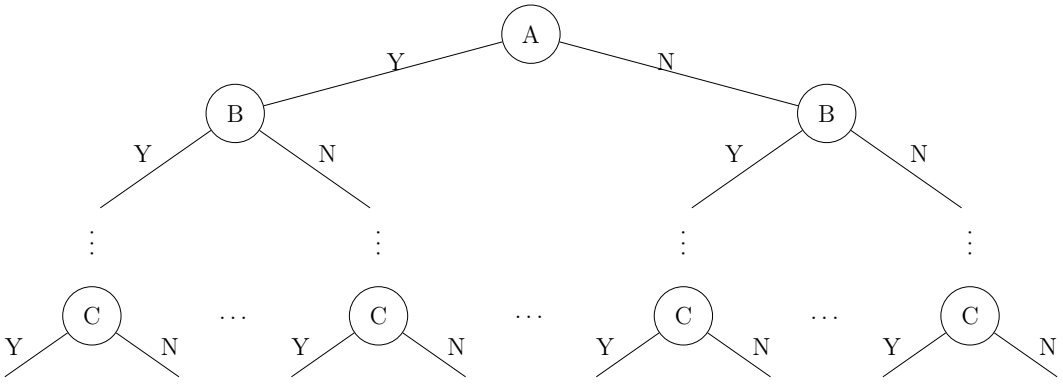

Figure 2-30: Subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ of depth N_1 is concatenated to the corresponding branch of the subtree depicted in Figure 2-29, resulting in a subtree of depth $2N_1$. The splits are as follows:

- A is $\mathbf{W}_{g,1}^T\mathbf{x}_2 + \mathbf{W}_{h,1}^T\mathbf{y}_{1,1} + b_{1,1} < 0$.
- B is $\mathbf{W}_{g,2}^T\mathbf{x}_2 + \mathbf{W}_{h,2}^T\mathbf{y}_{1,1} + b_{1,2} < 0$.
- C is $\mathbf{W}_{g,N_1}^T\mathbf{x}_2 + \mathbf{W}_{h,N_1}^T\mathbf{y}_{1,1} + b_{1,N_1} < 0$.

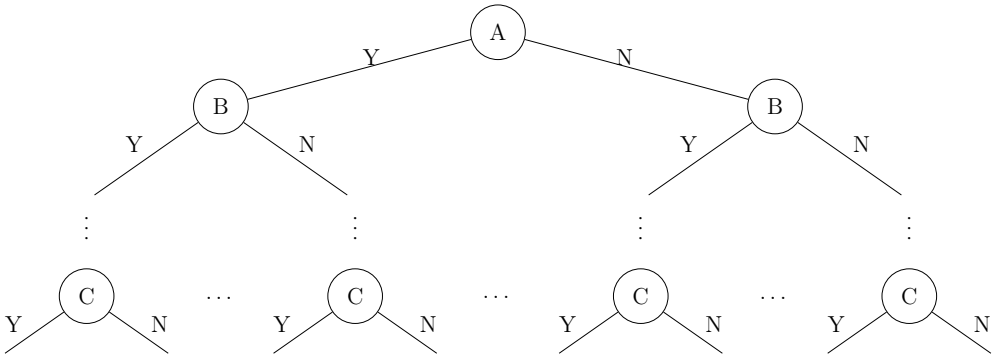

Figure 2-31: The resulting subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ for $\mathbf{y}_{1,1} = (0, \ldots, 0, \mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T$. The labels of A, B, C are as follows:

- A is $\mathbf{W}_{g,1}^T\mathbf{x}_2 + \mathbf{W}_{h,1}^T(0, \ldots, 0, \mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T + b_{1,1} < 0$.
- B is $\mathbf{W}_{g,2}^T\mathbf{x}_2 + \mathbf{W}_{h,2}^T(0, \ldots, 0, \mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T + b_{1,2} < 0$.
- C is $\mathbf{W}_{g,N_1}^T\mathbf{x}_2 + \mathbf{W}_{h,N_1}^T(0, \ldots, 0, \mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T + b_{1,N_1} < 0$.

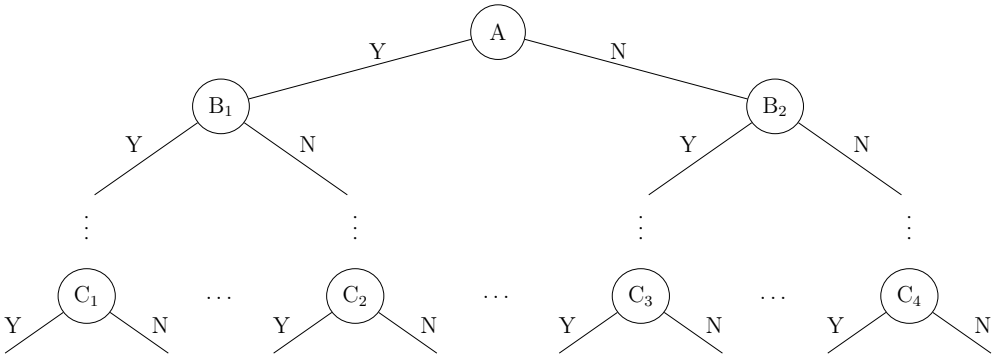

Figure 2-32: The subtree $\mathcal{T}_{6,O}(\mathbf{y}_1)$. The labels A, B_1, ..., C_4 are as follows:

- A is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,2})^T \mathbf{y}_{T^*,1} + b_{O,1} - b_{O,2}$
- B_1 is $(\mathbf{W}_{O,2} - \mathbf{W}_{O,3})^T \mathbf{y}_{T^*,1} + b_{O,2} - b_{O,3}$
- B_2 is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,3})^T \mathbf{y}_{T^*,1} + b_{O,1} - b_{O,3}$
- C_1 is $(\mathbf{W}_{O,q-1} - \mathbf{W}_{O,q})^T \mathbf{y}_{T^*,1} + b_{O,q-1} - b_{O,q}$
- C_2 is $(\mathbf{W}_{O,5} - \mathbf{W}_{O,q})^T \mathbf{y}_{T^*,1} + b_{O,5} - b_{O,q}$
- C_3 is $(\mathbf{W}_{O,3} - \mathbf{W}_{O,q})^T \mathbf{y}_{T^*,1} + b_{O,3} - b_{O,q}$
- C_4 is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,q})^T \mathbf{y}_{T^*,1} + b_{O,1} - b_{O,q}$

all inequalities are explicitly written as linear functions of $\mathbf{x}^*$.

Continuing in this way we model the output vector $\mathbf{y}_{t,1}$ of the tth time step of $\mathcal{N}_6$ as a classification tree by propagating the values of $\mathbf{y}_1$, $\mathbf{y}_2$,..., $\mathbf{y}_L$ as explicit linear functions of $\mathbf{x}^*$.

Following this, we model the output layer of the network the same way we did in Section 2.4.1, resulting in the subtree seen in Figure 2-32.

Given an output $\mathbf{y}_{T^*,1}$ of the hidden layer of $\mathcal{N}_6$, $\mathcal{N}_6$ calculates the vector $\mathbf{W}_O^T \mathbf{y}_{T^*,1} + \mathbf{b}_O$, and then $k = \arg\max_{i=1,\ldots,q}(\mathbf{W}_{O,i}^T \mathbf{y}_{T^*,1} + b_{O,i})$, outputting k as the classification prediction for input $\mathbf{x}^*$. We simulate this calculation with a subtree $\mathcal{T}_{6,O}(\mathbf{y}_{T^*,1})$ de-

picted in Figure 2-32, in the following manner. Node A checks whether $\mathbf{W}_{O,1}^T \mathbf{y}_{T^*,1} + b_{O,1}$ or $\mathbf{W}_{O,2}^T \mathbf{y}_{T^*,1} + b_{O,2}$ is larger. Node B_1 checks whether $\mathbf{W}_{O,2}^T \mathbf{y}_{T^*,1} + b_{O,2}$ or $\mathbf{W}_{O,3}^T \mathbf{y}_{T^*,1} + b_{O,3}$ is larger conditioned that $\mathbf{W}_{O,2}^T \mathbf{y}_{T^*,1} + b_{O,2} > \mathbf{W}_{O,1}^T \mathbf{y}_{T^*,1} + b_{O,1}$. Node B_2 checks whether $\mathbf{W}_{O,1}^T \mathbf{y}_{T^*,1} + b_{O,1}$ or $\mathbf{W}_{O,3}^T \mathbf{y}_{T^*,1} + b_{O,3}$ is larger conditioned that $\mathbf{W}_{O,1}^T \mathbf{y}_{T^*,1} + b_{O,1} > \mathbf{W}_{O,2}^T \mathbf{y}_{T^*,1} + b_{O,2}$, and so on. Each branch of the tree thus explicitly calculates which output node outputs the highest value (using a lexicographic decision rule in case of ties), and we can then assign the class associated with that node as the output to the appropriate leaf nodes of $\mathcal{T}_{6,O}(\mathbf{y}_{T^*,1})$. Since at each branch we know $\mathbf{y}_{T^*,1}$ as an explicit function of $\mathbf{x}^*$, we have that $\mathcal{T}_{6,O}(\mathbf{y}_{T^*,1})$ is a decision tree where all inequalities are explicitly written linear functions of $\mathbf{x}^*$ as well.

We next show that $\mathcal{T}_6$ can be used to make the same predictions as $\mathcal{N}_6$ for input data sequence $\mathbf{x}_t$, $t = 1, \ldots, T^*$. To see this, if $\mathbf{x}_t$ is input into $\mathcal{N}_6$, the hidden layer outputs $\mathbf{y}_{1,1}$ at the first time step, $\mathbf{y}_{2,1}$ at the second time step, and so on, until at last the sequence is assigned classification value $k(\mathbf{y}_{T^*,1})$. However, by the construction of the tree, the point $\mathbf{x}^*$ is sorted down the paths corresponding to $\mathbf{y}_{1,1}$, $\mathbf{y}_{2,1}$, and so on, until it is sorted to the leaf node corresponding to $k(\mathbf{y}_{T^*,1})$ by construction. Thus, for a given data sequence $\mathbf{x}_t$, the network and the tree predict the same classification value.

Since an optimal tree does at least as well as $\mathcal{T}_6$, that means that it must do at least as well as $\mathcal{N}_6$ too. Thus, by construction, we have that an optimal decision tree with maximum depth $T^* \times N_1 + q - 1$ can be used to classify data in a training set at least as well as a given neural network with the perceptron activation function, 1 hidden layer, and N_1 nodes in the first hidden layer, completing the proof of the theorem. □

2.6.4 Rectified Linear Unit Regression RNNs and ORT-Hs

Note that if the network is a regression neural network with the rectified linear unit activation function $\mathcal{N}_6$ instead of a classification neural network, meaning it outputs $\mathbf{W}_O^T \mathbf{y}_{T^*,1} + \mathbf{b}_O \in \mathbb{R}^q$, we are also able to build a regression tree $\mathcal{T}_6$ to make the same predictions as the neural network. We build the same decision tree as in the proof of Theorem 5 in Section 2.6.3 by building subtrees up until subtree $\mathcal{T}_{6,L}(\mathbf{y}_{T^*-1,1})$. This results in a tree of depth $T^* \times N_1$. Then, for each leaf node of this tree we assign the linear function $\mathbf{W}_O^T \mathbf{y}_{T^*,1} + \mathbf{b}_O$ as the output value, where by the construction of the tree $\mathbf{y}_{T^*,1}$ is a linear function of $\mathbf{x}^*$. Through this process, $\mathcal{T}_6$ calculates the same output as $\mathcal{N}_6$ given an input $\mathbf{x}^*$. Since $\mathcal{T}_6$ is a regression tree with hyperplane splits, it follows that an ORT-H has at least as good accuracy as $\mathcal{T}_6$, completing this extension of Theorem 6. An example of the final subtree is depicted in Figure 2-33.

2.7 Computational Results with Real World Data Sets

In this section, we examine the performance of FNNs and OCT-Hs on twelve data sets. We trained several different formulations of each model, and compared their out-of-sample accuracy on twelve data sets.

The twelve data sets we use are an anonymized version of the Framingham heart study data set developed using the Teaching Dataset provided by the National Heart, Lung, and Blood Institute, a modified version of the Poker Hand data set from the UCI Machine Learning Repository, and the Bank Marketing, Covertype, Dota 2 Games Results, Image Segmentation, Letter Recognition, Magic Gamma Telescope,

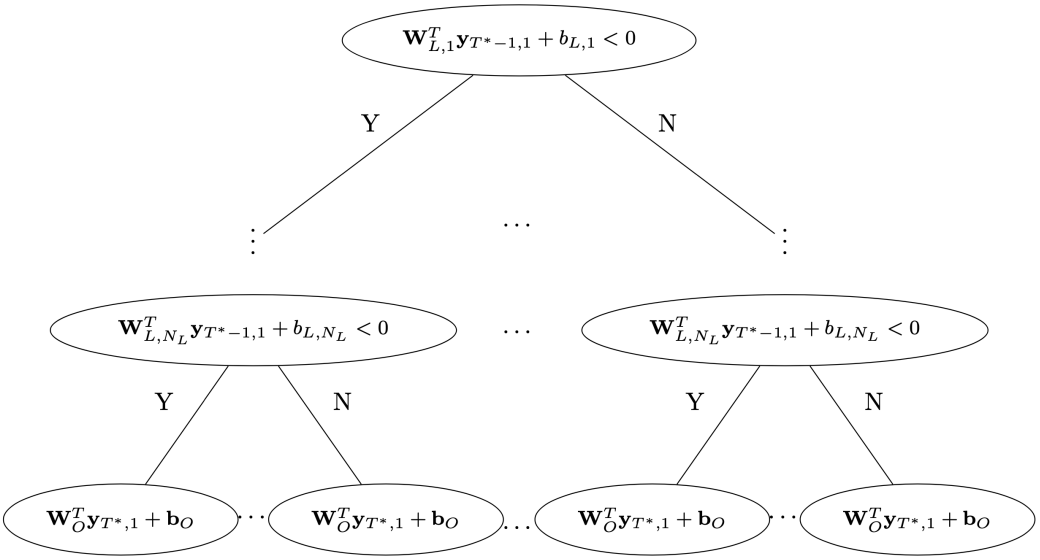

Figure 2-33: Subtree $\mathcal{T}_{6,L}(\mathbf{y}_{T^*-1,1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $T^* \times N_1$. Note that the leaves have linear functions $\mathbf{W}_O^T \mathbf{y}_{T^*,1} + \mathbf{b}_O$ as outputs.

Optical Recognition of Handwritten Digits, Skin Segmentation, Sonar – Mines vs. Rocks, and Thyroid Disease ANN data sets from the UCI Machine Learning Repository ([59]). In Table 2.3, we describe some details of these data sets.

To train the optimal trees, we used the Optimal Tree software ([8]). For each data set, we trained and cross validated optimal trees of varying depths with regular splits and optimal trees with hyperplane splits of the appropriate depths on the data sets, which was automatically done by the software. Once we had the best OCT and OCT-H based on the validation process, we calculated the models' accuracy in classifying the data in the test set – these out-of-sample accuracies are included as the "Best OCT" and "Best OCT-H" values in Table 2.5.

To train the neural networks, we used the TensorFlow code. For each neural network, we trained 50 neural networks with different random starts using grid search

Data Set	# Parameters	# Class Values	# Data Points
Bank Marketing	15	2	45,211
Covtype	54	7	581,012
Dota	116	2	102,944
Framingham heart study	15	2	3,658
Image Segmentation	18	7	210
Letter Recognition	16	26	20,000
Magic Gamma Telescope	10	2	19,020
Optical	64	10	3,821
Poker	10	10	625,942
Skin Segmentation	3	2	245,057
Sonar	60	2	208
Thyroid Disease ANN	21	3	3,772

Table 2.3: The data sets used and their parameters.

for the parameters. We then used the parameters with the best performance on a validation set to obtain the models' accuracy in classifying the data in the test set.

For activation functions, we use either the weighted sigmoid function (where a constant parameter within the sigmoid function is increased towards infinity as the training process goes on, making the function a closer and closer approximation of a threshold function) and the ReLU function. For the weighted sigmoid function we train two different variations of neural networks, known as NN1 and NN2 respectively.

1. Smaller neural networks, with N_1 equal to to 4, 6, or 8, and L (the number of hidden layers) equaling 1, 2, 3, 4, or 6. Since these neural networks have perceptron-like activation functions, based on Theorem 1 we would expect OCT-Hs to do approximately at least as well as these networks.

2. A version based on the work of [80], with two hidden layers, N_1 equal to the number of split nodes in a given tree, and N_2 equal to the number of leaf nodes in that tree. Given that construction, we expect these neural networks to do

Data Set	N_1
Bank Marketing	7
Covtype	2
Dota	7
Framingham heart study	7
Image Segmentation	2
Magic Gamma Telescope	7
Skin Segmentation	7
Sonar	7
Thyroid Disease ANN	6

Table 2.4: Data sets with equivalent ReLU NNs, and the number of nodes in the single hidden layer of those networks. N_1 was chosen so an OCT-H of depth 8 would be equivalent.

approximately at least as well as an OCT-H.

The best out-of-sample accuracies of these models are included in the columns "NN1" and "NN2" in Table 2.5.

For some of the data sets, it is not practical to train a tree that is deep enough to be equivalent to even a small ReLU neural network. For example, given a neural network with a single hidden layer containing five nodes, and trained on the Letter Recognition data set, by Theorem 2 we would need a tree of depth 29 to ensure equivalence. Thus, we have also split the ReLU neural networks into the equivalent case for some data sets, and an inequivalent case for all, respectively called NN3 and NN4. For the equivalent case, the networks are a single layer, with the number of nodes in that layer listed in Table 2.4.

For the inequivalent case, we use two hidden layer neural networks with either 5, 50, 500, or 1500 nodes in each layer. These inequivalent cases allow one to understand the potential performance gains a larger neural network could have on the data. The best out-of-sample accuracies of these models are included in the columns "NN3"

Data Set	Train	Valid	Test	Best OCT	Best OCTH	Best NN1	Best NN2	Best NN3	Best NN4
Bank	27,127	9,042	9,042	89.5%	89.8%	89.2%	89.2%	89.4 %	89.7%
Covtype	348,607	116,203	116,202	76.3 %	74.9 %	72.9 %	75.6%	72.3 %	90.1 %
Dota	55,590	37,060	10,294	55.8 %	57.7%	58.9%	58.5%	58.9%	59.5%
Framingham heart study	2,194	732	732	83.2%	82.9 %	82.4 %	82.1%	82.2%	82.4%
Image Segmentation	124	43	43	88.4%	86.0 %	83.7%	55.8%	81.4%	88.4%
Letter Recognition	12,000	4,000	4,000	67.2 %	81.1 %	55.5%	76.1 %	NA	97.0 %
Magic Gamma Telescope	11,414	3,803	3,803	86.3%	86.7%	85.2%	86.0%	87.3 %	87.7%
Optical	2,293	766	764	87.8%	90.7 %	90.1%	93.5%	NA	99.0%
Poker	40,686	40,687	544,569	93.2%	94.3%	95.2%	97.2%	NA	99.9%
Skin Segmentation	147,035	49,011	49,011	99.8%	99.9 %	99.8%	99.8%	99.9 %	99.9%
Sonar	125	42	41	61.0%	80.5%	80.5 %	73.2%	92.7 %	90.2%
Thyroid Disease ANN	2,264	754	754	99.7%	99.9 %	98.1%	96.6%	98.8%	98.5%

Table 2.5: Accuracy of FNNs, OCT-Hs and OCTs.

and "NN4" in Table 2.5.

The results form Table 2.5 lead to the following conclusions:

1. In 11 out of the 12 data sets (the exception is Sonar) the equivalent FNNs (NN1 and NN3) and the OCT-H have very similar accuracy. Moreover, in these data sets OCT and OCT-H also have very similar accuracy.

2. In seven out of twelve data sets, the trees perform as well as much larger networks (NN4).

3. In Letter Recognition OCT-H has a performance edge both with respect to all the neural networks and OCT.

4. In Sonar, the neural networks have a significant edge over both OCTs and OCT-Hs. Based on looking at the trees, it appears that since this is a data set with a large number of features relative to the number of points in the data set, the MIP/local search approach is having have trouble identifying the most important variables to use for splits.

We have proven in the paper that OCT-Hs and FNNs are equivalent in terms of power. However, the proofs require trees of large depths. These empirical results pro-

vide preliminary evidence that OCT-Hs (and OCTs) have comparable performance even with small depth. This indicates that there is indeed practical merit to use OCT-Hs in practice. The fact that OCTs gives similar results to FNNs is particularly noteworthy as OCTs are very interpretable.

2.8 Conclusion

In this paper, we showed that optimal decision trees are at least as powerful as neural networks in terms of modeling power and in the case of classification problems OCT-Hs and NNs have the same modeling power. While our constructions require deep trees that may be impractical to compute, we have also found that in twelve data sets that the modeling power of OCT-Hs and FNNs is indeed very similar even if the trees have small depth. While more empirical research is needed, we feel these findings are promising as OCT-Hs and especially OCTs are more interpretable than FNNs. They bring us closer to a significant objective of machine learning to build interpretable models with state of the art performance.

Chapter 3

Optimal Predictive Clustering

3.1 Introduction

One of the current major goals in machine learning is to create and implement methods that achieve state-of-the-art performance, are scalable, and are interpretable. However, none of the currently widely used state-of-the-art regression methods achieve success in all three metrics at the same time.

Lasso regression ([95]) is a widely used, fast, and fairly interpretable algorithm. One can solve for the model near-instantaneously, and by analyzing the magnitude and sign of the coefficients one can get an understanding of how a covariate contributed to the model's prediction. However, compared to methods such as XGBoost ([24]), it rarely achieves cutting edge performance.

CART ([18]) is a decision tree algorithm that uses a greedy training process to learn splits in the tree in order to make predictions. Like Lasso regression it scales very well, and it is even more interpretable, as the path a data point takes down the tree gives a clear summary of exactly why a certain prediction was made. However,

due to the greedy training algorithm, it also has a suboptimal performance compared to Optimal Regression Trees with Linear Predictions and XGBoost.

Optimal Regression Trees (ORT) and Optimal Regression Trees with Linear Predictions (ORT-L) ([9]) are both very interpretable decision tree methods like CART. The two methods differ in the type of prediction they output; ORT outputs point predictions, while ORT-L uses a Lasso regression model in each leaf to make predictions on data sorted to that leaf. While ORT trains at a reasonably fast speed, it is not as fast as Lasso regression or CART, and it also does not achieve the strong performance that ORT-L does. In contrast, ORT-L has cutting edge performance among interpretable regression methods, but it does not scale very well to larger datasets (on the scale of $p \geq 50$ and $n \geq 20000$).

XGBoost is scalable and performs extremely well on a variety of datasets ([68]). However, it is not directly interpretable, which we define as a black box method.

Lastly, Clus and Model Trees are two tree-based methods for interpretable clustering data that incorporate information about the features and the outcome variables in the training process. Model Trees have cutting edge performance among clustering for prediction methods and scales well, but does not perform as well as methods like XGBoost. Table 3.1 summarizes our qualitative ranking of the five methods in the categories of performance, scalability, and interpretability.

	Lasso regression	**CART**	**ORT**	**ORT-L**	**Clus**	**Model Trees**	**XGBoost**
Performance	6	7	5	3	4	2	1
Scalability	1	1	6	7	3	3	3
Interpretability	4	1	1	5	3	5	7

Table 3.1: Comparison of major machine learning methods relative to each other across the metrics of performance (out-of-sample R^2), scalability and interpretability. 1 is the best, while 7 is the worst.

From Table 3.1, we observe all existing methods have weakness in at least one category. We therefore seek to design a method that has strong performance in all three categories at the same time. Optimal Predictive Clustering (OPC) is an algorithm that uses mixed integer optimization (MIO) to simultaneously cluster the data while learning cluster specific regression models. When clustering the data, the proposed method takes into account both $\mathbf{X}$ and $\mathbf{y}$ values, in order to create clusters better suited to prediction. This process results in Lasso regression models that are specialized for the data in the cluster they are applied to, which achieve stronger out-of-sample performance than a single model would. The resulting model is also interpretable, as we can create profiles for each cluster to understand why the models make the predictions they do. In addition, it is scalable, as the MIO can be solved quickly to near optimality using strong warm starts and early stopping. The warm starts considering both $\mathbf{X}$ and $\mathbf{y}$ values are generated by a advanced version of K-Means known as K-Means++ ([3]). In Sections 3.3, 3.4, and 3.5, we show the resulting method combines strong out-of-sample performance, scalability, and interpretability.

3.1.1 Literature

Combining clustering and regression algorithms is known as clusterwise linear regression. Originally proposed by [88, 89], it is defined as finding a given number of clusters of observations such that the overall sum of squared errors within those clusters is minimized. [88] proposes a stepwise optimal solution which exchanges points in different clusters step by step to decrease the objective value. [90] and [63] extend this work to minimize the absolute deviations of the linear regression errors and provide further algorithmic improvements that speed up the learning process.

However, these methods depend a lot on their initialization, which can result in the algorithm returning sub-optimal solutions.

Other researchers have tried a variety of algorithms and metaheuristics to solve clusterwise linear regression problems. These include [27] using simulated annealing; [43] using Variable Neighborhood Search; [4] using a smoothed version of a nonlinear clusterwise linear regression model to iteratively find a solution; and [28], [92], [64], and [29] all applying mixture models trained using EM algorithms to learn the clusters and coefficients. However, these methods were either applied to small scale problems ($n \leq 500$) or involved a small number of clusters ($k \leq 3$), thus raising the question of how well they scale.

Lastly, [61] and [36] use an iterative algorithm that optimizes over both the sum of squared errors of the regression and the distance of $\mathbf{X}$ values from the centroids of the clusters they are assigned to. However, their clustering of the data does not include $\mathbf{y}$ values, and they also apply their methods to mainly smaller scale problems ($n \leq 10000$).

We next describe global optimization solutions to the clusterwise linear regression problem. [54] propose a nonlinear MIO formulation that maximizes a log-likelihood function objective and solve the problem without a guarantee of an optimal solution. [21] propose a mixed logical-quadratic programming formulation that provides a feasible method for solving the clusterwise regression problem to optimality. In later work they improve the solution by using branch and bound methods, column generation, and some other heuristics ([20], [22]). [13] apply linear mixed integer optimization to both regression and classification problems. [69] and [2] both extend this formulation to the case when there are multiple potential outputs. [69] use a mixed integer quadratic program combined with a column generation heuristic to find a solution, while [2] use a two step optimization framework to first find the

number of clusters to use, and then find the solution of their novel MIO. Overall, none of these global optimization papers consider both the closeness between $\mathbf{X}$ and the cluster centroids and the prediction error of $\mathbf{y}$ in their objectives, which can result in space to improve out-of-sample performance. It is also unclear how well the proposed formulations and algorithms scale.

There has also been work in using decision trees and random forests to cluster data using both $\mathbf{X}$ and $\mathbf{y}$ values. [71] represents an early example of using trees to cluster data into piecewise linear models, incorporating information about the standard deviation of the outcome variables into its splitting procedure during the training process. [98] furthers this work, incorporating techniques to deal effectively with enumerated attributes and with missing values. [15] adapts the basic top-down induction of decision trees method towards clustering to define a technique called Clus. This method uses a greedy training process to build a tree where each leaf is a cluster, and uses an objective function that maximizes inter-cluster distances, which is measured by comparing cluster profiles. [79] introduces Model Trees, which use a split function that takes into account both mean and covariance differences between the populations in the proposed splits to better identify subgroups in the longitudinal data while training the tree. In [102], the authors train random forests with special trees that use multivariate outputs, and then use a Partition Around Medoid (PAM) algorithm on the proximity matrix generated by the forest to create clusters of similarly expressed genes. Overall, these approaches rely on greedy methods to train the final models, which only lead to suboptimal solutions.

In summary, results from the literature indicate previous methods lack strong out-of-sample performance or scalability because they do not approach the globally optimal solution, do not consider both $\mathbf{X}$ and $\mathbf{y}$ values, or do not have strong warm starts. To solve this problem, the proposed method OPC takes into consideration not

only the distance between the data and their corresponding centroids, but also the prediction error, which significantly improves the out-of-sample performance of the method. OPC also utilizes a strong warm start based on the K-Means++ algorithm, which improves the scalability of the algorithm.

3.1.2 Contribution

In this paper, we combine ideas from clustering and prediction to propose a new method called Optimal Predictive Clustering. We model it directly as a MIO and illustrate two different ways to solve it with appropriate warm starts. Our main contribution in this paper is the two novel MIO formulations of the clusterwise linear regression problem that generalize some earlier approaches, alongside faster ways to solve them. We also show that by creating an ensemble model including all the other methods we investigate, we are to achieve even better performance that the individual models and outperforms the ensemble created without OPC. A key part of this formulation is clustering points based on both $\mathbf{X}$ and $\mathbf{y}$ values, so that the clusters are better suited for learning regression models. We further show that this method achieves strong performance in the following three categories:

1. **Performance**: We demonstrate that OPC increases out-of-sample R^2 by 0.019 (3.11%) on average compared to ORT-L on 20 datasets from UCI and LIACC machine learning Repository ([30],[96]). Furthermore, OPC increases out-of-sample R^2 by 0.134 (27.21% improvement) on average over Lasso regression and by 0.059 (10.37% improvement) on average over Clus on the same 20 datasets. An ensemble built from all the methods also increases out-of-sample R^2 by 0.007 (1.10% improvement) on average over Model Trees and by 0.007 (1.10% improvement) on average over XGBoost.

2. **Scalability**: We show the average time to solve the first version of OPC is about 800 seconds on the same 20 datasets, which is $\sim$30$\times$ faster than ORT-L. Furthermore, the maximum time to solve large scale problems ($n \sim 70000$, $p \sim 30$) is under 2000 seconds.

3. **Interpretability**: We show how to use the centroids the proposed method finds to create profiles of points in a given cluster and understand why it makes the predictions it does. We further compare OPC with ORT-L in two real world datasets, showing there is no significant loss in interpretability when using the proposed method.

For scalability, OPC is faster than ORT and ORT-L but slower than other methods. The performance of OPC is stronger than the rest methods except XGBoost and Model Trees, and an ensemble built using all the models achieves better performance than Model Trees and the ensemble without OPC as well. Finally, OPC's interpretability is similar to that of Lasso regression and ORT-L. Table 3.2 summarizes our qualitative ranking of the nine methods.

	Lasso regression	**CART**	**ORT**	**ORT-L**	**Clus**	**Model Trees**	**XGBoost**	**OPC**	**Ensemble**
Performance	8	9	7	5	6	3	2	4	1
Scalability	1	1	7	8	3	3	3	6	8
Interpretability	4	1	1	5	3	5	8	5	8

Table 3.2: Comparison of major machine learning methods and OPC relative to each other across the metrics of performance (out-of-sample R^2), scalability and interpretability. 1 is the best, and 9 is the worst.

3.1.3 Structure

The paper is structured as follows. In Section 3.2, we describe how to find an optimal clustering of points for prediction using MIO. Then we present two algorithms to solve the optimization problem using warm starts. In Section 3.3, we show experimentally on synthetic data that the algorithm recovers the truth. We also present computational results on 20 real-world datasets and compare it with other methods including Lasso regression, CART, Optimal Regression Trees (ORT, ORT-L), and XGBoost. In Section 3.4, we compare the training times of these methods to that of OPC on both synthetic data and real-world datasets. In Section 3.5, we discuss how to interpret the results of the proposed method. We then go through two real-world examples of this process, while comparing the proposed method's interpretability to that of ORT-L. We conclude in Section 3.6,

3.2 The approaches

In this section, we describe two ways of solving a MIO to find an optimal clustering of points for prediction.

3.2.1 Approach 1 – OPC-BigM

In this section, we first describe how we apply a MIO to find an optimal clustering of points for prediction. After that, we illustrate how to find a strong warm start to speed up the optimization process. Lastly, we describe how we tune the algorithm's hyper-parameters.

We are given data $(\mathbf{x}_i, y_i)$, $i \in [n]$, where $[n] = \{1, \ldots, n\}$, with $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$. For categorical variables, we use one-hot encoding to transform them into a

set of binary variables, and if there are missing values we can impute using methods described in [12]. We normalize the data by applying the transformation

$$\hat{x}_{ij} := \frac{x_{ij} - m_j}{\sigma_j},\ i \in [n],\ j \in [p], \tag{3.1}$$

where m_j and σ_j are the mean and standard deviation of dimension j of the training samples.

If we want to use clusters for prediction, one direct way is

1. Use K-Means to find the centroids for each cluster.

2. In each cluster, run a linear (Lasso) regression based on the y value.

3. When there is a new data point, assign it to the cluster with the closest centroid, then use the linear function for that cluster to predict the y value.

However, this only leads to a locally-optimal solution and also does not consider the $\mathbf{y}$ values. In order to achieve a globally optimal solution for this prediction problem, the objective function of the optimization problem should include:

1. The prediction error.

2. The distance between data and corresponding centroids.

We use a hyper-parameter α to decide the trade-off between them in the objective function.

To define the clusters in the MIO, assume we want to partition n points into K clusters. For each observation i, we introduce indicator variables $z_{ik} = \mathbb{1}\{x_i \text{ is in cluster } k\}$ to track the points assigned to each cluster. We force each point to be assigned to

exactly one cluster by adding the constraint:

$$\sum_{k\in[K]} z_{ik} = 1, \quad i \in [n]. \tag{3.2}$$

We also force each cluster to have at least $N_{\min}$ points:

$$\sum_{i\in[n]} z_{ik} \geq N_{\min}, \quad k \in [K]. \tag{3.3}$$

Define the centroid of cluster k as

$$\mathbf{h_k} = (h_{k,1}, \ldots, h_{k,p})', \quad k \in [K]. \tag{3.4}$$

We add auxiliary variables $\mathbf{l_i} = (l_{i,1}, \ldots, l_{i,p})'$ to represent the centroid of the cluster point i belongs to. We apply constraints

$$h_{k,j} - l_{i,j} \geq -(1 - z_{ik})M_1, \ i \in [n], \ j \in [p], \ k \in [K], \tag{3.5}$$

$$h_{k,j} - l_{i,j} \leq (1 - z_{ik})M_1, \ i \in [n], \ j \in [p], \ k \in [K], \tag{3.6}$$

to enforce $h_{k,j} = l_{i,j} \iff z_{i,k} = 1, \ \forall i, j, k$ where M_1 is a sufficiently large number so that the constraint $h_{k,j} - l_{i,j} \in [-M_1, M_1]$ does not restrict the values $h_{k,j} - l_{i,j}$ can take in any meaningful way. It can be chosen either through empirical experimentation or by using methods such as those found in [10].

In each cluster, we want to use regression to make a prediction. We introduce variables $\mathbf{c_k} = (c_{k,1}, \ldots, c_{k,p})', d_k$ as the coefficients and intercept for the regression model in cluster k. We add auxiliary variables g_i to represent the predicted value for

point i. We apply constraints

$$g_i - \sum_{j\in[p]} x_{i,j}c_{k,j} - d_k \geq -(1-z_{ik})M_2, \quad i \in [n],\ k \in [K], \tag{3.7}$$

$$g_i - \sum_{j\in[p]} x_{i,j}c_{k,j} - d_k \leq (1-z_{ik})M_2, \quad i \in [n],\ k \in [K], \tag{3.8}$$

enforcing $g_i = \sum_{j\in[p]} x_{i,j}c_{k,j} + d_k \iff z_{i,k} = 1,\ \forall i,j,k$ where M_2 is a sufficiently large number. We further apply L2 regularization to the coefficients $c_{k,j}$ weighted by hyper-parameter λ.

Putting all of this together gives the following MIO formulation for finding optimal clusters for prediction, which we call the OPC model:

$$\begin{aligned}
\min \quad & \sum_{i\in[n]} \left[\sum_{j\in[p]} \alpha(x_{i,j} - l_{i,j})^2 + (1-\alpha)(y_i - g_i)^2 \right] + \lambda \sum_{k\in[K]} \sum_{j\in[p]} c_{k,j}^2 \qquad (3.9)\\
\text{s.t.} \quad & \sum_{k\in[K]} z_{ik} = 1, \quad i \in [n],\\
& \sum_{i\in[n]} z_{ik} \geq N_{\min}, \quad k \in [K],\\
& h_{k,j} - l_{i,j} \geq -(1-z_{ik})M_1, \quad i \in [n],\ j \in [p],\ k \in [K],\\
& h_{k,j} - l_{i,j} \leq (1-z_{ik})M_1, \quad i \in [n],\ j \in [p],\ k \in [K],\\
& g_i - \sum_{j\in[p]} x_{i,j}c_{k,j} - d_k \geq -(1-z_{ik})M_2, \quad i \in [n],\ k \in [K],\\
& g_i - \sum_{j\in[p]} x_{i,j}c_{k,j} - d_k \leq (1-z_{ik})M_2, \quad i \in [n],\ k \in [K],\\
& z_{ik} \in \{0,1\}, \quad i \in [n], k \in [K].
\end{aligned}$$

Solving this MIO to optimality can be time consuming. We propose the following

algorithm and use the solution as a warm start. For each number of clusters k and tuning parameter α we want to tune over, we cluster $[\alpha\mathbf{X}, (1-\alpha)\mathbf{y}]$ into k clusters using K-Means++ and train Lasso regression models within each one. We then compare the performance of each of these models on the validation set, and then find the k^* and α^* parameters where the model performs the best on the validation set. We lastly use these k^* and α^* values to calculate warm starts for the cluster assignments, cluster centroids, and regression coefficients for the MIO.

Algorithm 1 Find warm starts for OPC

Select potential numbers of clusters $K \in \{K_1, K_2, K_3, \ldots, K_i\}$ and potential trade-off parameters $\alpha \in \{\alpha_1, \ldots, \alpha_j\}$ to tune over.
for $K = K_1, \ldots, K_i$ **do**
    for $\alpha = \alpha_1, \ldots, \alpha_j$ **do**
        Multiply the original $\mathbf{X}$ by α and add $(1-\alpha)\mathbf{y}$ as a new column.
        Run the K-Means++ algorithm to find K clusters for $[\alpha\mathbf{X}$,$(1-\alpha)\mathbf{y}]$
        Get the centroids of each cluster, and then remove the $\mathbf{y}$ column
        Reweight the remaining $\alpha\mathbf{X}$ values in the cluster centroids by $\frac{1}{\alpha}$
        Re-assign points to the new cluster centroids
        Run a Lasso regression in each cluster
    end for
end for
Identify the solution (K^*, α^*) with largest R^2 on the validation set.
Multiply the original $\mathbf{X}$ by α^* and add $(1-\alpha^*)\mathbf{y}$ as a new column.
Run the K-Means++ algorithm to find K^* clusters for $[\alpha^*\mathbf{X}$,$(1-\alpha^*)\mathbf{y}]$
Get the centroids of each cluster, and then remove the $\mathbf{y}$ column
Reweight the $\alpha^*\mathbf{X}$ values remaining in the cluster centroids by $\frac{1}{\alpha^*}$ to get the centroid $\mathbf{h}_j^*$ for each cluster j
Re-assign points to the new cluster centroids and get cluster assignments $\mathbf{z}^*$
Run a Lasso regression in each cluster j and get coefficients $\mathbf{c}_j^*, d_j^*$
Return $K^*, \alpha^*, \mathbf{z}^*, \mathbf{h}_j^*, \mathbf{c}_j^*, d_j^*$ $(j \in [K^*])$

Once the process in Algorithm 1 is completed, the resultant output is used as a warm start to solve problem (9).

3.2.2 Approach 2 – OPC-CP

An alternative version of the optimization problem we want to solve for OPC is as follows.

$$
\begin{aligned}
\min_{\mathbf{Z}} \min_{\mathbf{h},\mathbf{c},\mathbf{d}} \quad & \sum_{i\in[n]} \left[\sum_{j\in[p]} \alpha(x_{i,j} - \sum_{k=1}^{K} z_{i,k} h_{k,j}(\mathbf{Z}))^2 + (1-\alpha)(y_i - \sum_{k=1}^{K} z_{i,k}(\mathbf{c}_k(\mathbf{Z})'\mathbf{x}_i + d_k(\mathbf{Z})))^2 \right] \\
& + \lambda * (\sum_{k=1}^{K} ||\mathbf{c}_k||^2 + (d_k)^2) \\
\text{s.t.} \quad & \sum_{k\in[K]} z_{i,k} = 1, \quad i \in [n], \\
& \sum_{i\in[n]} z_{i,k} \geq N_{\min}, \quad k \in [K], \\
& z_{i,k} \in \{0,1\}, \quad i \in [n], k \in [K].
\end{aligned}
$$

where as before $z_{i,k}$ is one if point i is in cluster k, and 0 otherwise, h is the K by p matrix of cluster centroids, and the $\mathbf{c}_k, d_k$ variables are the linear models associated with each cluster. We define

$$
\mathbf{h}_{k,j}(\mathbf{Z}) = \frac{\sum_{i=1}^{n} x_{i,j} z_{i,k}}{\sum_{i=1}^{n} z_{i,k}}
$$

$$
[d_k(\mathbf{Z}); \mathbf{c}_k(\mathbf{Z})] = (\mathbf{X}_k' \text{Diag}(\mathbf{Z}_k)^2 \mathbf{X}_k + \lambda * \mathbf{I}_{p+1})^{-1} \mathbf{X}_k' \text{Diag}(\mathbf{Z}_k)^2 \mathbf{y}
$$

where $\mathbf{x}_i$ is the ith row of matrix $\mathbf{X}$, $\mathbf{X}_k(\mathbf{Z})$ is the rows of matrix $\mathbf{X}$ where $z_{i,k} = 1$ and with a column of one values appended to it, $\mathbf{y}_k(\mathbf{Z})$ is the rows of vector $\mathbf{y}$ where $z_{i,k} = 1$, $\text{Diag}(\mathbf{Z}_k)$ is a square matrix with the entries of the kth column of $\mathbf{Z}$ along the diagonal and zeros elsewhere, and $\mathbf{I}_{p+1}$ is the $(p+1) \times (p+1)$ identity matrix.

We can then define the objective of the above problems as $\min_{\mathbf{Z}} f(\mathbf{Z})$, where

$$f(\mathbf{Z}) = \min_{\mathbf{h},\mathbf{c},\mathbf{d}} \sum_{i\in[n]} \left[\sum_{j\in[p]} \alpha(U_{i,j})^2 + (1-\alpha)(V_i)^2 \right] + \lambda * (\sum_{k=1}^{K} ||\mathbf{c}_k||^2 + (d_k)^2)$$

where

$$U_{i,j} = x_{i,j} - \sum_{k=1}^{K} z_{i,k} h_{k,j}(\mathbf{Z})$$

and

$$V_i = y_i - \sum_{k=1}^{K} z_{i,k}(\mathbf{c}_k(\mathbf{Z})'\mathbf{x}_i + d_k(\mathbf{Z}))$$

This optimization problem can be solved using cutting planes. We start with the formulation

$$\begin{aligned} \min_{\mathbf{z},t} \quad & t \\ \text{s.t.} \quad & \sum_{k\in[K]} z_{ik} = 1, \quad i \in [n], \\ & \sum_{i\in[n]} z_{ik} \geq N_{\min}, \quad k \in [K], \\ & t >= 0 \\ & z_{ik} \in \{0,1\}, \quad i \in [n], k \in [K]. \end{aligned} \tag{3.10}$$

where an initial solution can be found using the Algorithm 1 warm start procedure. When we find a solution to the new optimization problem, we then add a

constraint of the form

$$t \geq f(\hat{\mathbf{z}}) + \sum_{i=1}^{n}\sum_{k=1}^{K} \nabla f(\hat{\mathbf{z}})_{i,k} * (z_{i,k} - \hat{z}_{i,k})$$

where $\hat{\mathbf{z}}$ is a solution to the optimization problem that we need to add a constraint to remove, and $\nabla f(\hat{\mathbf{z}})_{i,k}$ is the partial derivative of f at the variable $z_{i,k}$ at the point $\hat{\mathbf{z}}$. These partial derivative of this function is as follows

$$\begin{aligned}\frac{\partial f}{\partial z_{i,k}} = -2(\sum_{j\in[p]} \alpha(x_{i,j} - \sum_{q=1}^{K} z_{i,q}h_{q,j}) * (h_{k,j}(\mathbf{Z}) + z_{i,k}\frac{\partial}{\partial z_{i^*,k}}\mathbf{h}_{k,j}(\mathbf{Z}))\\ +(1-\alpha)(y_i - \sum_{q=1}^{K} z_{i,q}(\mathbf{c}_q(\mathbf{Z})'\mathbf{x}_i + d_q(\mathbf{Z}))) * (\mathbf{c}_k(\mathbf{Z})'\mathbf{x}_i + d_k(\mathbf{Z}))\\ +z_{i,k}\frac{\partial}{\partial z_{i^*,k}}(\mathbf{c}_k(\mathbf{Z})'\mathbf{x}_i + d_k(\mathbf{Z})))\end{aligned}$$

with the following definitions of the partial derivatives

$$\frac{\partial}{\partial z_{i^*,k}}\mathbf{h}_{k,j}(\mathbf{Z}) = \frac{x_{i^*,j}\sum_{i=1}^{n} z_{i,k} - \sum_{i=1}^{n} x_{i,j}z_{i,k}}{(\sum_{i=1}^{n} z_{i,k})^2}$$

$$\begin{aligned}\frac{\partial}{\partial z_{i^*,k}}[d_k(\mathbf{Z}); \mathbf{c}_k(\mathbf{Z})] = -2z_{i^*,k} * ((\mathbf{X}'\mathbf{Z}_k^2\mathbf{X})^{-1})' * \mathbf{x}_i' * \mathbf{x}_i * ((\mathbf{X}'\mathbf{Z}_k^2\mathbf{X})^{-1}) * \mathbf{X}'\mathbf{Z}_k^2\mathbf{y}\\ +2 * ((\mathbf{X}'\mathbf{Z}_k^2\mathbf{X}))^{-1} * \mathbf{x}_i' * y_i z_{i^*,k}\end{aligned}$$

We continue adding cutting planes until either we find an optimal solution, or we reach a time limit set at the beginning of the optimization process.

3.3 Performance

In this section, we report the performance of OPC on a variety of synthetic and real-world datasets. In 3.3.1, we discuss data preprocessing methods and the hyper-parameters we validate for OPC as well as the other ML algorithms we use as benchmarks. In 3.3.2, we present OPC's performance on synthetic data and illustrate that it recovers the true underlying model from the data. In 3.3.3, we present OPC's performance on real-world datasets using both solution algorithms and then compare them to the performances of Lasso regression, CART, ORT, ORT-L, XGBoost methods, Clus, and Model Trees. We also show the benefit of using both $\mathbf{X}$ and $\mathbf{y}$ values in the objective function by comparing OPC with methods that do not use $\mathbf{y}$ values, as well as the improvement in performance when combining different OPC models into an ensemble.

3.3.1 Data Preprocessing and Model Validation

For a given dataset, we partition it into three parts: the training (60%), validation (20%), and testing sets (20%). The data are all standardized based on the mean and standard deviation of the combined training and validation sets.

For each of the models, we tune parameters by choosing those that lead to the best performance of the model on the validation set. Ranges for the parameters are chosen based on ranges seen in other projects using these methods. For Lasso regression, we validate regularization parameter λ using values between 0.0001 and 0.1. For CART, we validate minimum number of points per leaf in {5,10,20,50,100}. For ORT and ORT-L, we tune both the depth of the tree from 1 to 10 and the complexity parameters between 0.001 and 0.1 based on the description from [49]. For OPC models, after some experimentation we found that OPC-BigM and OPC-

CP had very similar predictive performances but that OPC-BigM finished training much faster, so we present results for that method for the remainder of the paper. We use α values between 0.03 and 1.0 to investigate a broad range of ways we could wait the outcomes vs observed features, and K values between $\frac{n}{200}$ and $\frac{n}{1000}$ (for smaller datasets with $n \leq 1500$ we validate K= [1,2,3,4], for $1500 < n \leq 3000$ we validate K = [3,4,5,6,7]). These k sizes were chosen under the assumption that smaller datasets were likely to have fewer clusters, and larger datasets were likely to have more. We fix N_{min} to be 10, λ to be 0.01, and $M_1 = M_2 = 1000$ based on empirical observations that these parameters are effective choices across models. The ensemble of all methods was a weighted average of the predictions of the different models, where the weights were chosen to give higher performing models more weight compared to lower performing models. The no-OPC Ensemble had Model Trees, ORL-L, and XGBoost predictions weighted twice as much as the other predictions (so XGBoost predictions had a weight of 1/5 vs CART predictions having a coefficient of 1/10), and in the All-Emsemble OPC also had twice the weight of the lower performing methods (so OPC predictions had a weight of 1/6 vs 1/12 for models like CART). In simulation experiments, we fix K a range of values centered around the true value, using [3,4,5,6,7] for $k = 5$, [6, 8, 10,12,14] for $k = 10$, and [12, 16, 20, 24, 28] for $k = 20$. For XGBoost, we tune both the depth of the tree from 1 to 10 and the learning parameters between 0.1 and 0.5. The Clus package does not allow for using a separate training and validation set to be defined, so for this method the two sets are combined into one, and the package automatically cross validates model parameters and prunes the resulting trees.

All problems except for Clus and Model Trees are solved in parallel using the MIT Engaging Cluster (Intel Xeon 2.1 GHz) with 1 CPU core and 32GB of memory, and using the Python and Julia programming languages. Lasso regression and CART

models are trained using the sklearn package in Python ([70]). ORT/ORT-L models are trained using Interpretable AI's Julia Interface ([49]), which applies the training algorithms found in Chapters 10 and 11 of [9]. XGBoost models are trained using the XGBoost package in Python ([24]). The Clus package uses a Java implementation, and is run on a Macbook Pro with a 3.5 GHz Dual-Core Intel Core i7 processor and 16 GB of memory. The Model Trees were run on that same Macbook Pro using an implementation in Weka.

In order to avoid outcomes being influenced by one particular split of the data into training, validation, and test sets, we conduct the experiments five times with different splits each time. The final out-of-sample R^2 reported is the average of the five runs.

3.3.2 Performance on synthetic datasets

In this section, we investigate experimentally whether the algorithm converges to the ground truth we generate using synthetic datasets.

To create the datasets, we generate several different cluster centroids randomly. We then generate points nearby the centroids by adding noise terms to the centroids drawn from a normal distribution with mean 0, resulting in a cluster of points around the centroid. Lastly, we assign random linear regression weights to each cluster and use them to define y values for each point in it. We use the following values of (K, n, p) to define the ground truths:

- Number of clusters K: 5, 10, 20
- Size of the data n: 1000, 10000, 100000
- Number of variables p: 10, 25, 40

Index	K	p	n	Test R^2	Index	K	p	n	Test R^2
1	5	10	1000	1	15	20	25	10000	1
2	10	10	1000	1	16	5	40	10000	1
3	20	10	1000	1	17	10	40	10000	1
4	5	25	1000	1	18	20	40	10000	1
5	10	25	1000	1	19	5	10	100000	1
6	20	25	1000	1	20	10	10	100000	1
7	5	40	1000	1	21	20	10	100000	1
8	10	40	1000	1	22	5	25	100000	1
9	20	40	1000	1	23	10	25	100000	1
10	5	10	10000	1	24	20	25	100000	1
11	10	10	10000	1	25	5	40	100000	1
12	20	10	10000	1	26	10	40	100000	1
13	5	25	10000	1	27	20	40	100000	1
14	10	25	10000	1					

Table 3.3: Average out-of-sample R^2 of OPC-BigM on the synthetic data.

Given this construction, the proposed method should be able to perfectly learn the true model, as we generate the data to match what the proposed method learns. The performance of the proposed method is shown in Table 3.3, where the Index columns are simply a numerical index of the datasets, columns K, p, and n are defined as above, and the Test R^2 column contain the test set R^2 of the proposed method.

Based on the results in Table 3.3, we observe the test R^2 for the method is 1. This illustrates that the proposed method is consistently able to find the ground truth clusters in the data when such clusters exist. For OPC models, the warm starts play a key role here because they help our algorithm find the ground truth within the solving time. Empirically, when we give the optimization solver more than 300 seconds to solve the problem, the final solution does not change. Thus, the algorithm finds the true optimum in that time span, and uses the rest of the time to guarantee

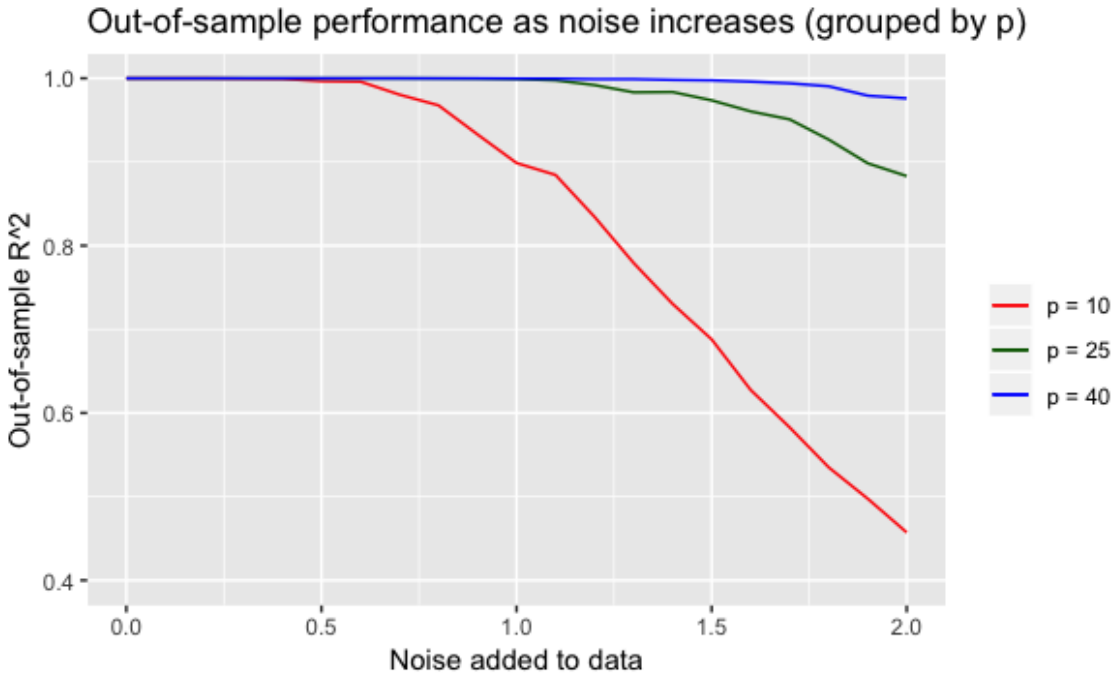

Figure 3-1: Out-of-sample performance of the model with increasing levels of noise stratified by number of variables in the simulated data.

the optimality. OPC is therefore able to recover the underlying structure of the data when such a structure exists, aided by using high quality warm starts.

We then experimented with adding noise to the $\mathbf{X}$ values of the data, to get better intuition of how model performance would change when applied to real world data. This noise was drawn from a uniform random variable Uniform$(-\epsilon, \epsilon)$, where $\epsilon = 0, 0.1, 0.2, \ldots, 1.9, 2$. This resulted in the following performance, aggregated by number of variables in the data, in Figure 3-1. As one can see, the method is generally less affected by increasing noise in the higher dimensional datasets. This makes sense, because in higher dimensional models the noise we add to the data gets added together in the linear model. Because the noise is Uniform$(-\epsilon, \epsilon)$, adding it together causes the summed noise to tend towards zero, lessening its effect.

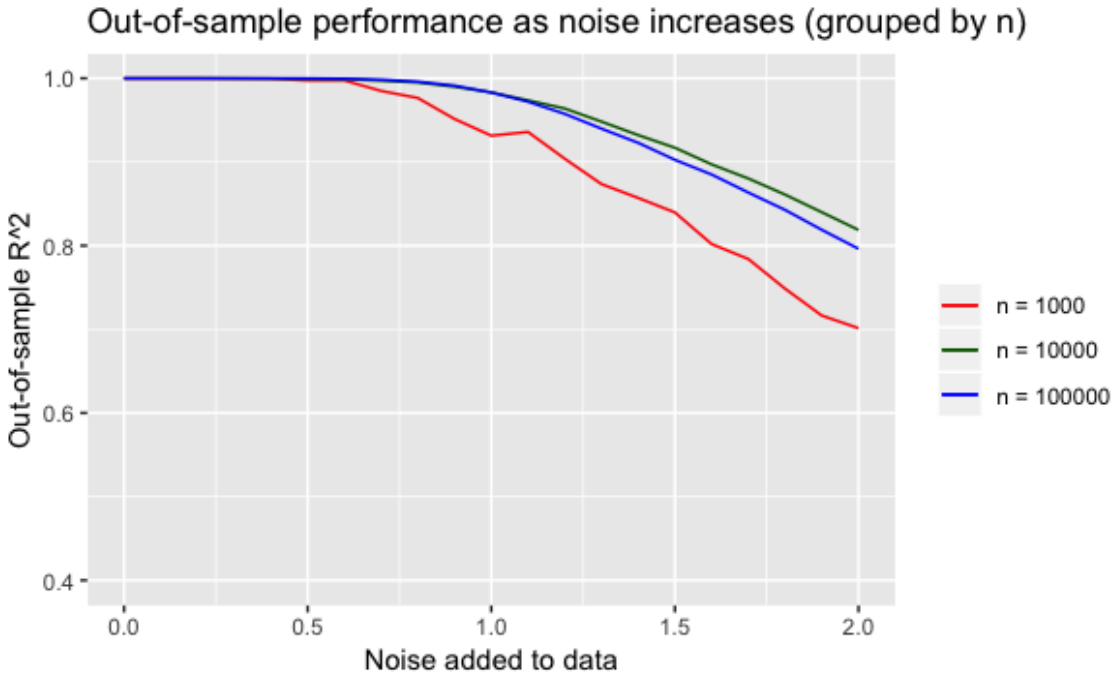

Figure 3-2: Out-of-sample performance of the model with increasing levels of noise stratified by number of points in the simulated data.

Next, we analyze the performance of the model aggregated by number of data points, as seen in Figure 3-2. Here we see a much more striking decrease in performance as the amount of noise increases in all cases, although in general larger amounts of data still help the model make better inferences.

Overall, we observe in both graphs that for larger n and p values, the model is more robust to noise.

3.3.3 Performance on real-world datasets

In this section, we report the out-of-sample R^2 of OPC and other machine learning methods on 20 real-world datasets obtained from the UCI and LIACC Machine

Dataset	n	p	CART	Lasso regression	ORT	ORT-L	Clus	Model Trees	XGBoost	OPC	No-OPC Ensemble	All-Ensemble
housing	505	13	0.730	0.752	0.818	0.831	0.801	0.859	0.866	0.863	0.891	**0.898**
geographic-origin	1059	68	0.452	0.657	0.467	0.650	0.477	0.637	0.592	0.658	0.664	**0.671**
wine-quality-red	1598	11	0.298	0.375	0.312	0.378	0.318	0.377	0.424	0.394	**0.428**	**0.428**
vote-for-clinton	2703	9	0.323	0.311	0.294	0.398	0.341	0.390	**0.417**	0.342	0.414	0.413
abalone	4176	7	0.442	0.525	0.469	0.527	0.490	0.551	0.535	0.551	0.555	**0.563**
wine-quality-white	4897	11	0.251	0.281	0.274	0.334	0.290	0.338	**0.432**	0.339	0.398	0.403
PTM	5874	16	0.103	0.089	0.161	0.146	0.191	0.235	**0.299**	0.215	0.270	0.284
PTT	5874	16	0.116	0.092	0.174	0.174	0.184	0.246	**0.323**	0.230	0.286	0.305
ailerons	7153	40	0.760	0.824	0.752	0.824	0.766	0.846	0.828	0.839	0.843	**0.847**
cpu-act	8191	21	0.967	0.722	0.968	0.977	0.971	**0.979**	**0.979**	0.936	0.978	0.977
cpu-small	8191	12	0.959	0.712	0.962	0.969	0.971	**0.975**	0.973	0.960	0.972	0.972
kin8nm	8191	8	0.445	0.422	0.518	0.641	0.474	0.615	0.684	0.655	0.684	**0.721**
elevators	8751	18	0.696	0.829	0.691	0.828	0.460	**0.894**	0.829	0.864	0.870	0.883
pole	15000	26	0.970	0.474	0.966	0.967	0.972	0.974	**0.984**	0.911	0.979	0.973
elevatorlarge	16559	17	0.717	0.816	0.707	0.815	0.724	**0.895**	0.838	0.859	0.875	0.883
energy	19735	29	0.201	0.161	0.218	0.255	0.301	0.294	**0.447**	0.351	0.390	0.404
californiahousing	20460	8	0.691	0.631	0.747	0.658	0.737	0.759	**0.800**	0.713	0.781	0.781
censusdomain	22784	16	0.378	0.265	0.397	0.429	0.453	0.580	**0.603**	0.439	0.570	0.575
CASP	45730	9	0.435	0.283	0.461	0.427	0.484	0.425	**0.595**	0.475	0.561	0.556
online_video	68784	25	0.954	0.651	0.956	0.961	0.970	0.842	**0.988**	0.947	0.966	0.965
AVERAGE			0.544	0.494	0.566	0.609	0.569	0.636	0.672	0.627	0.669	0.675

Table 3.4: Average out-of-sample R^2 of all methods on real-world datasets

Learning Repositories. These results can be found in Table 3.4. The abbreviations PTM and PTT stand for parkinsons-telemonitoring-motor and parkinsons-telemonitoring-total, two UCI datasets. While none of these datasets contain missing values, any missing values could be imputed prior to the clustering and prediction process. In the left column the names of all datasets are listed from smallest n to largest n. The next two columns to the right show the number of data points n and the number of variables p in each dataset. The remaining columns contain the out-of-sample R^2 of all methods.

Based on these results, the ensemble combination of all the methods (All-Ensemble) has the best average out-of-sample performance of all methods. Compared with Lasso regression, the ensemble increases out-of-sample R^2 by 0.149 (36.16% improvement) on average in these datasets. Compared with ORT-L, All-Ensemble outperforms ORT-L in 19 out of 20 datasets with an average improvement of 0.034 (13.4% improvement). Compared with Model Trees, the best performing non-OPC method that clusters data using both $\mathbf{X}$ and $\mathbf{y}$ values, All-Ensemble outperforms it on 13

datasets with an average improvement of 0.007 (6.1% improvement). Compared with XGBoost, OPC-Ensemble outperforms it in 8 out of 20 datasets. However, its average performance is actually still higher than that method's, with average improvement of 0.003 (0.4%).

Significantly, the ensemble of methods including OPC outperforms the ensemble of methods without OPC, with average improvement of 0.006 (0.9%). This shows that OPC, in addition to being one of the best performing methods on its own, is useful in conjuction with other methods.

In order to understand the benefit of considering $\mathbf{y}$ values when building the clusters, we compare the performance of OPC with other clusterwise regression methods, in particular those that do not incorporate tree structures (as methods such as Clus do), in Table 3.5 . A basic algorithm for building a clusterwise regression model is first running the K-Means++ algorithm to cluster all points, and then building a Lasso regression model in each cluster. The result of this basic algorithm is shown in the column named Basic. In [2], the authors propose a new MIO formulation for multiple response clustering that expanded upon previous formulations. We implement their algorithm with warm starts, and the result is shown in the column named Nested regression. The time constraints for solving MIO and the possible number of clusters are the same as for OPC. The regularization parameter is chosen to be either 0.01 or 0.1. We observe that the [2] result is similar to the basic method. Alongside this, OPC outperforms the basic method significantly in 10 out of 20 datasets and performs similarly to the basic method in the others, with an average improvement of 0.011 (1.79% improvement). Taking $\mathbf{y}$ values into account in the clustering is therefore a useful technique to improve the performance of the method.

In order to understand the sensitivity of the method to the choice of k, we looked at the performance of OPC on the Red Wine, Abalone, and Pole datasets when just

Dataset	Basic	Nested regression	OPC
housing	0.846	0.846	**0.863**
geographic-origin	0.634	0.623	**0.658**
wine-quality-red	0.375	0.38	**0.394**
vote-for-clinton	0.343	0.335	0.342
abalone	0.55	0.545	**0.551**
wine-quality-white	**0.343**	**0.343**	0.339
PTM	**0.216**	**0.216**	0.215
PTT	0.21	0.21	**0.23**
ailerons	**0.841**	**0.841**	0.839
cpu-act	0.929	0.929	**0.936**
cpu-small	0.959	0.959	**0.960**
kin8nm	0.629	0.629	**0.655**
elevators	0.862	0.862	**0.864**
pole	0.876	0.876	**0.911**
elevatorlarge	0.857	0.857	**0.859**
energy	0.331	0.331	**0.351**
californiahousing	0.711	0.711	**0.713**
censusdomain	**0.44**	**0.44**	0.439
CASP	0.446	0.446	**0.475**
online_video	0.924	0.924	**0.947**
AVERAGE	0.616	0.615	0.627

Table 3.5: Average out-of-sample R^2 of K-Means++ with regression, Nested regression, and OPC on real-world datasets. Bolded values are the highest values in a given row.

Dataset	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
wine-quality-red	0.276	0.380	0.408	0.389	0.355
abalone	0.524	0.560	0.555	0.554	0.546
pole	0.889	0.902	0.903	0.901	0.911

Table 3.6: Model performance when just choosing a single k value and training the OPC model using that choice.

setting $k = 3, 4, \ldots, 7$ instead of cross validating. The out-of-sample results on those datasets are in Table 3.6.

We see that while there are a couple of large leaps (such as from $k = 3$ to $k = 4$ for the wine-quality-red data set), the model performance does not change much for most of the incremental changes in k. Cross validation should therefore be sufficient to identify optimal choices for the k values.

3.4 Scalability

In order to better understand the scalability of the proposed method, we also measure how long the training and validation process takes for each dataset. The training times for OPC on the synthetic data are in Table 3.7. Like Table 3.3, the Index columns are simply a numerical index of the datasets, and columns K, p, and n are defined as the number of clusters, the number of variables and the size of the data. The Time column contains the time it takes to build the various models. For the majority of the datasets, the OPC training process (including calculating warm starts and training the optimization formulation) takes less than ten minutes, and for all datasets it finishes within 3 hours.

Meanwhile, the training times of all methods for the real-world datasets are in Table 3.8. For these datasets, OPC always finishes running in an hour, finishes faster

Index	K	p	n	Time	Index	K	p	n	Time
1	5	10	1000	305.2	15	20	25	10000	823.1
2	10	10	1000	316.8	16	5	40	10000	478.0
3	20	10	1000	322.0	17	10	40	10000	692.4
4	5	25	1000	312.6	18	20	40	10000	1095.8
5	10	25	1000	325.2	19	5	10	100000	770.6
6	20	25	1000	353.1	20	10	10	100000	1452.8
7	5	40	1000	318.0	21	20	10	100000	2510.9
8	10	40	1000	340.1	22	5	25	100000	1516.5
9	20	40	1000	380.4	23	10	25	100000	2756.2
10	5	10	10000	348.0	24	20	25	100000	5525.3
11	10	10	10000	404.1	25	5	40	100000	2137.2
12	20	10	10000	518.9	26	10	40	100000	4253.0
13	5	25	10000	411.9	27	20	40	100000	8699.6
14	10	25	10000	546.4					

Table 3.7: Training times of OPC on synthetic data.

Dataset	n	p	CART	Lasso regression	ORT	ORT-L	Clus	Model Trees	XGBoost	OPC	No-OPC Ensemble	All-Ensemble
housing	505	13	0.01	0	13.5	1902	0.524	0.054	0.1	357.1	1903	1903
geographic-origin	1059	68	0.173	0	78.8	104134.3	0.773	0.634	0.9	339	104135.3	104135.3
wine-quality-red	1598	11	0.03	0	31.8	2697.9	0.583	0.54	0.3	305.3	2698.9	2698.9
vote-for-clinton	2703	9	0.069	0	93.5	6698.6	0.75	0.422	0.5	339.4	6699.6	6699.6
abalone	4176	7	0.062	0	128.5	2734.6	0.818	0.534	0.5	351.9	2735.6	2735.6
wine-quality-white	4897	11	0.11	0	113.3	6720.6	0.86	0.518	0.8	477.9	6721.6	6721.6
PTM	5874	16	0.29	0.1	431.5	8432.6	1.161	0.924	1.6	676.1	8433.6	8433.6
PTT	5874	16	0.311	0.1	352.7	10815.6	1.25	0.604	1.6	665.3	10816.6	10816.6
ailerons	7153	40	0.261	0.5	765.6	7300.3	1.801	1.786	2	702.2	7301.3	7301.3
cpu-act	8191	21	0.394	0	771.2	25221.8	1.393	1.77	2	661.4	25222.8	25222.8
cpu-small	8191	12	0.252	0	536.7	14242.2	1.218	0.802	1.4	689.2	14243.2	14243.2
kin8nm	8191	8	0.246	0	786.8	25759.9	1.175	1.734	1.5	720.9	25760.9	25760.9
elevators	8751	18	0.249	0.3	470.5	1399.8	1.044	1.172	1.5	873.8	1400.8	1400.8
pole	15000	26	0.279	0.2	1762.8	38081.1	1.44	1.974	2.9	1009.1	38082.1	38082.1
elevatorlarge	16559	17	0.465	0.6	1506.2	5084.9	2.043	3.226	2.2	907.2	5085.9	5085.9
energy	19735	29	1.874	0.4	2003.2	87423.9	3.097	3.71	5.1	1219.5	87424.9	87424.9
californiahousing	20460	8	0.545	0.1	2209.6	28424.2	1.941	1.442	2.4	1075.3	28425.2	28425.2
censusdomain	22784	16	1.421	0.2	1616.9	49912.7	2.651	2.47	4.9	1317.6	49913.7	49913.7
CASP	45730	9	1.7	1.96	5781.4	75781.3	6.546	3.146	6.6	1589	75782.3	75782.3
online_video	68784	25	1.576	1.8	17933	10100.9	1.821	5.748	10.2	1995.1	17934	17934
AVERAGE			0.51585	0.313	1869.375	25643.46	1.64445	1.6605	2.45	813.615	26036.065	26036.065

Table 3.8: Training times of the all methods on real-world datasets.

than ORT on average, and often finishes running an order of magnitude faster than ORT-L. Thus, the method achieves both strong out-of-sample performance and faster training times than comparable methods. It is still, however, slower than XGBoost and Model Trees (which finish running in seconds), as well as CART and Lasso regression (which finish running almost instantaneously). Aggregating the ensemble models generally takes a second or less, so the main time constraint with them is training the other models to include in the ensemble in parallel.

3.5 Interpretability

3.5.1 How to interpret the OPC model

Interpreting the results of OPC is a two-step process. First, we can analyze the clusters to create a profile of which points are sorted to which cluster. After that, we can study the regression coefficients to understand the particular model being used within a cluster.

To perform the first step, we can better understand why a point is sorted to a given cluster by comparing the centroids of the clusters. By looking at the differences in feature values for each centroid, we can find the key features that cause a point to be sorted to one cluster over another. Given that empirically we have found that the model can learn the variations in the data effectively using a small number of clusters, this process is easily scalable. This process can be done by inspection, or algorithmically. The inspection approach is self explanatory – a person can look at all of the cluster centroids and find differences within them, and use these differences to interpret the points assigned to a particular cluster. For the algorithmic approach, one can look at the cluster centroids feature by feature, sort them, and identify which

centroids have larger or smaller average values for each category.

Then, once a point is assigned to a given cluster, we can investigate the coefficients of that cluster's regression model. By analyzing their magnitude and sign, we're able to understand how different features contribute to the final prediction. Thus, obtaining a prediction for a new data point is mainly deciding which cluster profile it is most similar to and then using a simple and easily understandable cluster-specific model to make a prediction.

One can clearly see the process is interpretable. To illustrate this, in the following subsections, we will walk through two examples of this process to show empirically how interpretable the method is. We will also compare OPC models with ORT-L trees trained on the same datasets, to show how the proposed method does not lose a significant amount of interpretability compared to decision tree models.

3.5.2 Housing Dataset Results

In this section, we illustrate how to interpret the OPC model on a real-world example about predicting housing values from [44]. We also compare the interpretability of OPC with ORT-L. There are 505 samples and 13 variables in this dataset. The description of each variable is shown in Table 3.9. The target is to predict the housing value in suburbs of Boston.

For the hyper-parameters of OPC we use α between 0.03 and 1.0 and K between 3 and 7. We fix N_{min} to be 10 and λ to be 0.01. After we run the OPC algorithm, it returns 6 clusters and the final out-of-sample R^2 is 0.921. The information about each cluster OPC finds is shown in Table 3.10. Each row represents the centroid of a given cluster and the last row shows the number of data points in the cluster.

In this model, we observe that all data points are nearly equally separated between

Variable	Meaning
CRIM	Per capita crime rate by town
ZN	Proportion of residential land zoned for lots over 25,000 sq.ft
INDUS	Proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
N0X	Nitric oxides concentration (parts per 10 million)
RM	Average number of rooms per dwelling
AGE	Proportion of owner-occupied units built prior to 1940
DIS	Weighted distances to five Boston employment centres
RAD	Index of accessibility to radial highways
TAX	Full-value property-tax rate per $10,000
PTRATIO	pupil-teacher ratio by town
B	$1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
LSTAT	Percentage of lower status of the population

Table 3.9: The meaning of each variable in the housing dataset, which has 505 data points.

	Centroid 1	Centroid 2	Centroid 3	Centroid 4	Centroid 5	Centroid 6
CRIM	19.09	0.96	7.18	0.20	0.05	0.89
ZN	0.00	0.34	0.00	6.25	60.29	20.06
INDUS	18.49	15.86	18.10	7.30	3.40	6.00
CHAS	0.00	0.12	0.09	0.05	0.00	0.18
N0X	0.68	0.62	0.67	0.48	0.42	0.51
RM	5.79	5.89	6.22	6.19	6.66	7.31
AGE	94.17	92.10	86.53	55.29	28.27	63.27
DIS	1.79	2.56	2.35	4.61	7.25	3.60
RAD	23.20	4.46	24.00	4.62	3.84	6.16
TAX	667.80	371.92	666.00	292.67	312.64	294.47
PTRATIO	20.20	18.44	20.20	18.42	16.99	16.38
B	212.65	353.71	358.77	390.38	390.15	388.38
LSTAT	23.86	16.97	15.29	10.15	5.89	5.30
Housing value	12.10	17.98	19.51	22.54	29.33	36.34
Number of data in this cluster	45	74	60	125	58	42

Table 3.10: Information about the centroid of each cluster and the number of data points within it.

	Characteristics of the house
Centroid 1	Extremely high crime rate, old house, close to employment centers, high property-tax rate, larger population of people classified as lower status
Centroid 2	Old house, less rooms in the house
Centroid 3	High crime rate, high property-tax rate
Centroid 4	Normal house
Centroid 5	Very large house, far from employment centres, better education environment, less lower status population, new house
Centroid 6	Large house, More rooms in the house, better education environment, much less lower status population, low property-tax rate

Table 3.11: Characteristics of the centroids of each cluster found using inspection.

clusters. There are also significant differences between the average housing values of each cluster. Looking at the centroids closely, we observe that each cluster can be interpreted as representing a specific type of house. In Table 3.11, we describe the characteristics of each centroid that we find using inspection, while in Table 3.12, we describe the characteristics we find using an algorithmic approach. As one can see, the two methods get nearly identical results, illustrating that extracting insights from the model does not require human inspection, but can be achieved automatically and algorithmically. When a new house comes, the model first indicates which type of house this new house is most similar to based on the distance to each centroid. After it is assigned to a specific cluster, we use linear regression to predict the exact housing value of this new house. The coefficients of linear regression in each cluster are shown in Table 3.13. In different clusters, variables have different levels of importance. For Cluster 1, CRIM, N0X and AGE are the most important variables in the Lasso regression, as they have the largest magnitude. For Cluster 6, however, the most important coefficients are INDUS, AGE, DIS, TAX, PTRATIO and LSTAT.

As a comparison, we also train an ORT-L model on this data. We tune both the depth of the tree from 1 to 10 and the complexity parameter to be between 0.001 and

	Characteristics of the house
Centroid 1	Highest crime rate, lowest number of rooms, highest property tax, oldest houses, closest to employment centers, highest population of people classified as lower status, lowest proportion of tracts bounding Charles
Centroid 2	Second oldest houses, second lowest number of rooms in the house, second highest proportion of tracts bounding the Charles
Centroid 3	Second highest crime rate, second highest property-tax rate, second lowest distance to Boston employment centers, second highest proportion of non-retail business acres per town
Centroid 4	Second lowest crime rate, second lowest age of house, lowest property tax rate, second highest distance from the Boston employment centers
Centroid 5	Second highest number of rooms, lowest crime rate, newest houses, furthest distance from Boston employment centers, second lowest proportion of lower status population
Centroid 6	Largest number of rooms in the house, lowest pupil teacher ratio, lowest proportion of lower status population, second lowest property-tax rate

Table 3.12: Characteristics of the centroids of each cluster found algorithmically.

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
CRIM	-0.081	0.000	-0.363	0	0	0.000
ZN	0	0	0	0	0.053	-0.023
INDUS	0	0.066	0	-0.144	0	0.459
CHAS	0	0.691	3.453	0	3.107	0
N0X	-38.996	-12.999	-30.585	-3.823	0	0
RM	-0.530	4.241	-1.855	5.964	11.397	9.409
AGE	0.109	0	0	-0.045	-0.006	-0.116
DIS	0	-0.217	-4.769	-0.954	-0.520	-1.561
RAD	0	0	0	0.010	0	0
TAX	0	-0.003	0	-0.011	-0.003	-0.020
PTRATIO	0	-0.871	0	-0.332	0.083	-1.120
B	-0.002	0.011	0	0	0	0
LSTAT	-0.253	-0.228	-1.303	0	-0.354	-0.696

Table 3.13: Coefficients of the linear regression in each cluster.

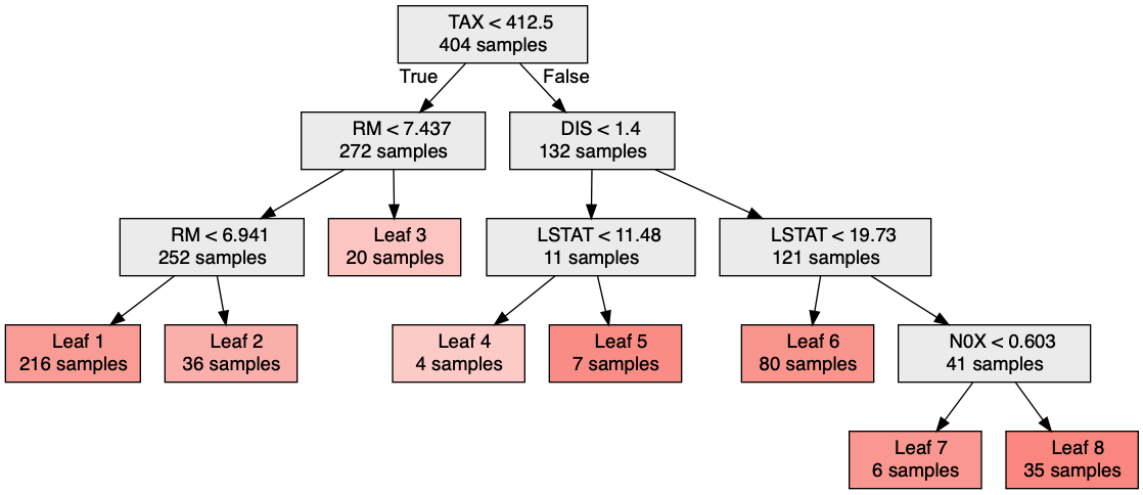

Figure 3-3: ORT-L for predicting house values.

0.1. The out-of-sample R^2 of this ORT is 0.884. The exact tree is shown in Figure 3-3 and the coefficients of the linear regression model of each leaf are shown in Table 3.14. Similarly to OPC, ORT-L also finds several clusters (leaves). In each leaf, the prediction is made by a linear regression. However, while the ORT-L's splits are easy to follow, the profiles they create at each leaf node are not always as detailed as the ones created by OPC, as if a variable does not appear in a split, it is basically ignored by the model. The data are also spread around much more unevenly in the tree, with some leaves having a single digit number of points while others have many more. Overall, though, observing a point's distance from different centroids is not significantly different from following the path a point takes down the tree, and there is thus not a significant loss of interpretability.

3.5.3 Wine Quality Dataset Results

In this section, we illustrate how to interpret OPC on another real-world example about wine quality from [26], and again compare the interpretability of OPC with

	Leaf 1	Leaf 2	Leaf 3	Leaf 4	Leaf 5	Leaf 6	Leaf 7	Leaf 8
CRIM	0	0	0	0	0	-0.050	0	-0.029
ZN	0.009	0.004	0.007	0	0	0	0	0
INDUS	-0.089	0	0	0	0	0	0	0
CHAS	0.953	-1.680	0	0	0	0	0	0
N0X	-7.244	0	0	0	0	0	0	0
RM	4.086	0	0	0	0	-0.499	0	0
AGE	-0.037	0	0	0	0	-0.001	0	0
DIS	-0.815	0	0	0	0	0	0	0
RAD	0.021	0	0	0	0	0.033	0	0
TAX	-0.008	0	0	0	0	0	0	-0.006
PTRATIO	-0.564	0	-0.731	0	0	0	0	0
B	0.008	0	0	0	0	0.009	0	0
LSTAT	-0.187	0	0	0	0	-0.501	0	0

Table 3.14: Coefficients of the linear regression in each leaf.

ORT-L. There are 1598 samples and 11 variables in this dataset. The target is to predict the quality of red wine.

We tune both number of clusters K from 3 to 7 and α between 0.03 and 1.0. We fix N_{min} to be 10 and λ to be 0.01. After we run the OPC algorithm, it returns 4 clusters and the out-of-sample R^2 of OPC is 0.404. The information about each cluster OPC finds is shown in Table 3.15. Each row represents the centroid for a given cluster and the last row shows the number of data points in each cluster.

In this model, we observe all data points are once again nearly equally spread among each cluster. There is also a significant difference between the average wine quality values of each cluster. Looking at the centroids closely, we observe that each cluster represents a specific type of red wine. In Table 3.16, we describe the characteristics of each centroid, while in table 3.17, we describe the characteristics we find using an algorithmic approach. As one can see, the two methods once more get nearly identical results, further illustrating that interpretable insights can be

	Centroid 1	Centroid 2	Centroid 3	Centroid 4	Centroid 5
Fixed acidity	8.11	7.96	8.39	8.88	8.84
Volatile acidity	0.58	0.73	0.50	0.41	0.40
Citric acid	0.24	0.17	0.28	0.37	0.41
Residual sugar	2.57	2.56	2.46	2.74	2.76
Chlorides	0.09	0.10	0.09	0.08	0.07
Free sulfur dioxide	17.07	12.23	15.32	14.00	14.00
Total sulfur dioxide	55.96	33.51	40.35	34.18	30.86
Density	1.00	1.00	1.00	1.00	1.00
PH	3.31	3.38	3.31	3.29	3.23
Sulphates	0.62	0.60	0.68	0.74	0.78
Alcohol	9.94	10.20	10.63	11.45	11.95
Quality	5.25	5.34	5.68	6.11	6.55
Amount of data in this cluster	371	330	236	205	136

Table 3.15: Information about the centroids of each cluster and the number of data points within it.

extracted from the model without human inspection. When we want to make a prediction about a new wine, the model will first indicate which type of wine this new wine is most similar to based on the distance to each centroid. After it is assigned to a specific cluster, we use linear regression to predict the exact quality of this new wine. The coefficients of the cluster-specific linear regression models are shown in Table 3.18. We can see in different clusters, different variables are important in the Lasso regression. For example, alcohol and total sulfur dioxide are the most important variables in Cluster 1. However, fixed acidity, chlorides, and PH are the most important variables in Cluster 5.

As a comparison, we also train an ORT-L model. We tune both the depth of the tree from 1 to 10 and the complexity parameter to be between 0.001 and 0.1. The out-of-sample R^2 of this ORT-L is 0.376. The tree is shown in Figure 3-4 and the coefficients of the linear regression model of each leaf are shown in Table 3.19.

Like in the last example, the tree's splits are easy to follow, but the resulting profiles are not the most informative compared to the OPC centroid profiles. The

	Characteristics of the red wine
Centroid 1	High free sulfur dioxide, high total sulfur dioxide, low sulphates, low alcohol
Centroid 2	High volatile acidity, Low fixed acidity, high sulphates
Centroid 3	Normal Wine
Centroid 4	High fixed acidity, low volatile acidity
Centroid 5	High fixed acidity, low volatile acidity, low pH high citric acid, low chlorides,high sulphates, high alcohol

Table 3.16: Characteristics of the centroids of each cluster found using inspection.

	Characteristics of the wine
Centroid 1	Lowest alcohol content, highest total sulfur dioxide, highest free sulfur dioxide
Centroid 2	Second lowest alcohol content, second lowest amount of residual sugar, highest PH, lowest acidity, second lowest sulphates value, lowest free sulfur dioxide, highest volatile acidity
Centroid 3	Lowest residual sugar level, second highest free and total sulfur dioxide levels, second highest PH values
Centroid 4	Second highest residual sugar levels, second highest alcohol values, second highest citric acid level, second lowest PH value, second lowest volatile acidity, highest fixed acidity
Centroid 5	Highest residual sugar value, highest alcohol content level, lowest chloride level, lowest PH value, second highest fixed acidity, lowest volatile acidity, highest sulphate value, highest citric acid values

Table 3.17: Characteristics of the centroids of each cluster found algorithmically.

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Fixed acidity	0	0.117	0	0.065	-0.033
Volatile acidity	-0.226	-1.404	-0.725	-0.181	-0.679
Citric acid	-0.167	0	-0.500	0	0.375
Residual sugar	0.023	0	0.200	0.029	0.035
Chlorides	0	0	-0.655	-1.638	-3.603
Free sulfur dioxide	0	0.014	0.004	0	0
Total sulfur dioxide	-0.003	0	0	-0.004	-0.002
Density	8.633	-56.112	0	-107.908	25.898
PH	0	-0.312	-0.156	-0.052	0.364
Sulphates	0	1.872	0.576	1.728	0.096
Alcohol	0.284	0.184	0.154	0.192	0.008

Table 3.18: Coefficients of the regression in each cluster.

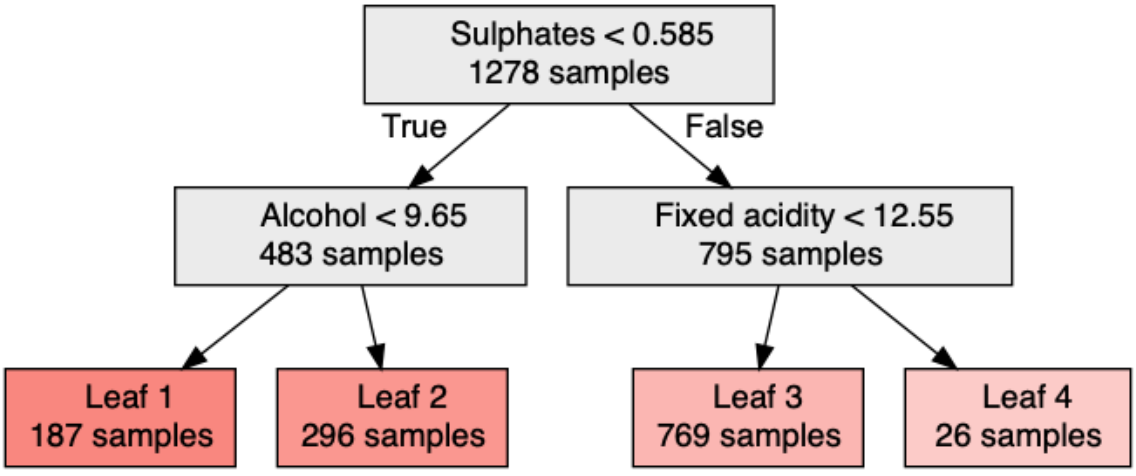

Figure 3-4: The Optimal Tree for predicting wine quality.

	Leaf 1	Leaf 2	Leaf 3	Leaf 4
Fixed acidity	0	0	0	0
Volatile acidity	0	-1.179	-0.939	0
Citric acid	-0.1443	0	-0.246	0
Residual sugar	0	-0.013	0.0213	0
Chlorides	0	0	-0.709	0
Free sulfur dioxide	0	0.009	0.002	0
Total sulfur dioxide	0	0	-0.006	0
Density	0	0	-0.435	0
PH	0	-0.652	-0.651	0
Sulphates	0	0	0.203	0
Alcohol	0	0.200	0.351	-0.123

Table 3.19: Coefficients of the linear regression in each leaf.

data are also spread more unevenly in the tree as in the last example, with the number of points sorted to a particular leaf ranging from 26 to 769. Overall, though, this process again shows that classifying points based on their distance from different centroids is still not significantly less interpretable than following the path a point takes down the tree.

These two real-world examples therefore illustrate how to interpret OPC models. In summary, OPC first finds which cluster the new data point belongs to and then uses linear regression to make a prediction. Compared to Optimal Regression Trees, OPC does not have the apparent geometric structures of splits to differentiate the clusters. However, given the profiles one can create using the cluster centroids, OPC provides faster training times and better performance without losing too much interpretability.

3.6 Conclusions

In this paper, we introduced a methodology known as optimal predictive clustering that generalizes MIO-based approaches to the problem of clusterwise regression. This method clusters data using both $\mathbf{X}$ and $\mathbf{y}$ values, while also learning cluster specific models to make predictions. In terms of performance, scalability and interpretability, we illustrate OPC is competitive with previous methods, as shown in Table 3.2.

Experiments with synthetic data show strong evidence that OPC can recover the underlying structure of the data when such a structure exists. For real-world datasets, we show that the proposed algorithm achieves cutting edge performance at a reasonable speed. Furthermore, the All-Ensemble increases out-of-sample R^2 by 0.034 (13.4% improvement) on average compared to ORT-L, by 0.007 (6.1% improvement) compared to Model Trees, and by 0.003 (0.4% improvement) compared to XGBoost. Through a general description of the interpretation process and two examples, we also show that OPC maintains interpretability while achieving these results. The fact that this process can be accomplished algorithmically further emphasizes the interpretability of the method, as manual inspection is no longer requires to extract meanings from the clusters.

OPC therefore brings us closer to a current significant objective in machine learning – to create and implement models with state-of-the-art performance, scalability, and interpretability.

Chapter 4

On the Limitations of Predicting Post Transplant Outcomes

4.1 Introduction

Although organ transplantation is often a life-saving procedure, the demand for organs outstrips the supply. Around 13,500 people were added to the transplant waiting list in 2019 and only around 9000 organs were transplanted that same year. Therefore, deciding how to allocate the limited number of organs available is a major challenge.

There are multiple factors one can use to decide who should be offered an organ. One option is looking at which potential recipient would have the greatest predicted post-transplant survival time if they received the organ. This approach has some historical precedent, as predicted post-transplant survival is already incorporated into the Lung Allocation Score ([33]), which is a key factor in deciding who is offered a donated lung or pair of lungs when one becomes available. Other authors have pro-

posed using such a metric for other organs like the liver ([101],[19]). With the Organ Procurement and Transplantation Network (OPTN) switching over to a continuous distribution model, which is intended to be a "more equitable system of allocating deceased donor organs," there will likely be discussion of how post-transplant survival could be used as a factor in the system.

However, it is not clear how well one could predict post-transplant survival at the time of allocation. If one cannot do so accurately enough, then it could be questionable to base allocation policies on it. In this chapter we seek to explore the question how well one can predict post-transplant survival for liver, renal, and lung transplantation using a comprehensive data set and state-of-the-art machine learning models, to understand the potential value of its inclusion as a factor in the new allocation system.

We find that overall predicting one-year graft outcomes is a difficult task. The best performing model across all the organs has an AUC of 0.67, which means the models do not effectively differentiate recipient-donor pairs who will have successful grafts from those who will experience a failure event in one year. We therefore believe that predicted post-transplant survival outcomes should not be included as a criterion for organ allocation at this time.

4.2 Data and Exclusion Criteria

We obtained data for the different organs from the Organ Procurement and Transplantation Network's Standard Transplant Analysis and Research (STAR) dataset. The data included the following categories:

- Waitlist information, such as patient demographics, medical lab values, etc.

- Deceased-donor information including demographics, medical lab values, cause of death, etc.

- Transplant information, including which donors were matched to which recipients, cold ischemia time, location codes for the transplant, etc.

- Follow-up information for the patient, such as if the patient's graft failed or not, if it failed when it failed, etc.

4.2.1 Liver Data

Waitlist, deceased-donor, transplant, and follow-up information for liver transplants was obtained for the period January 1st, 2002 to September 5th, 2016 from the Organ Procurement and Transplantation Network Standard Transplant Analysis and Research (STAR) dataset. The data were filtered using the exclusion criteria that the recipient must have been over 18 years old and not have received any previous transplant, that the donor was deceased, that the transplant involved a whole liver, and that the recipient received no other organ (ex. a kidney).

4.2.2 Lung Data

Waitlist, deceased-donor, transplant, and follow-up information for lung transplants was obtained for the period January 1st, 2006 to September 5th, 2016 from the STAR dataset. The data were filtered using the exclusion criteria that the recipient must have been over 18 years old and not have received any previous transplant, that the donor was deceased, that the transplant involved either a single lung or a pair, and that the recipient received no other organ (ex. a kidney).

4.2.3 Kidney Data

Waitlist, deceased-donor, transplant, and follow-up information for kidney transplants was obtained for the period January 1st, 2002 to September 5th, 2016 from the STAR dataset. The data were filtered using the exclusion criteria that the recipient must have been over 18 years old and not have received any previous transplant, that the donor was deceased, that the transplant involved a kidney, that the recipient received no other organ such as a liver, and that the recipient had either one year of follow up or a recorded date of graft failure or death.

A notable criterion that we do not use for liver and lung data but do use for kidney data is requiring that the patient have either follow up records for at least one year past their transplantation date or have a graft failure or death date recorded. This is because if a liver or lung graft fails, the recipient will either need medical attention or will die. The first case will result in a medical follow-up visit and a record of the graft failure existing. The second case will also result in a record, since the OPTN independently seeks out and obtains information about patient deaths. We can therefore assume that a liver or lung patient's graft is still functioning at the one-year mark if information about the follow up status of a patient ends prior to a year, but they do not have graft failure or death events recorded. For a kidney transplant, though, it is possible for the graft to start to fail and the recipient to just resume the dialysis, without needing to visit a hospital and having the graft failure event recorded. We therefore require kidney graft recipients to have enough follow-up information to ensure we can accurately identify their transplant outcome in one year.

4.3 Observations, Dependent and Independent Variables

4.3.1 Liver Data

An observation corresponded to a donor-recipient pair at the time of transplant. All such available observations that were not excluded based on the previously discussed criteria were retrieved and totaled 69,248 observations. For each observation, the dependent variable was set to 1 if the patient experienced graft failure or died within one year of their transplant, and to 0 otherwise. A total of 210 independent variables were recorded for each observation. This included variables in the medical records at time of transplant such as lab values, as well as additional variables that were calculated from some of the more granular features. Examples of constructed variables include the change in lab values like bilirubin and serum creatinine from a recipient's previous medical appointment to the transplant and the ratio of donor and recipient BMIs, among others.

Missing values were imputed using OptImpute, a machine learning approach which has demonstrated the ability to outperform other related extant methods ([12]). 38 features have missing values, with maximum percentage of missing values per feature being 21.5%. 27.4% of the pairs in the liver data have some missing values among their features, with up to 8.6% of the features being missing for a given donor-recipient pair.

4.3.2 Lung Data

An observation corresponded to a donor-recipient pair at the time of transplant. All such available observations that were not excluded based on the previously discussed

criteria were retrieved and totaled 18,096 observations. For each observation, the dependent variable was set to 1 if the patient experienced graft failure or died within one year of their transplant, and to 0 otherwise. A total of 148 independent variables were recorded for each observation. As with the liver data, additional variables were calculated from some of the more granular features such as the change in lab values like bilirubin and serum creatinine from a recipient's previous medical appointment to the transplant, the ratio of donor and recipient BMIs, etc.

Missing values were again imputed using OptImpute. 44 features have missing values, with maximum percentage of missing values per feature being 33.3%. 60.7% of the pairs in the lung data have some missing values among their features, with up to 20.2% of the features being missing for a given donor-recipient pair.

4.3.3 Kidney Data

An observation corresponded to a donor-recipient pair at the time of transplant. All such available observations that were not excluded based on the previously discussed criteria were retrieved and totaled 124,675 observations. For each observation, the dependent variable was set to 1 if the patient experienced graft failure or died within one year of their transplant, and to 0 otherwise. A total of 114 independent variables were recorded for each observation. As with the liver data, additional variables were calculated from some of the more granular features such as the change in lab values like bilirubin and serum creatinine from a recipient's previous medical appointment to the transplant, the ratio of donor and recipient BMIs, etc.

Missing values were again imputed using OptImpute. 17 features have missing values, with maximum percentage of features missing being 40.6%. 99.3% of the pairs in the data have some missing values among their features, but only up to

11.4% of the features were missing for a given donor-recipient pair.

4.4 Methods

4.4.1 Predictive methods

The prediction problem was addressed using models that were trained on historical data. These models included interpretable Optimal Classification Trees (OCTs) and three popular black box methods called random forests, XGBoost, and neural networks. OCTs are a state-of-the-art machine learning prediction method that affords interpretability and high prediction accuracy ([7]). On the other hand, random forests, XGBoost, and neural networks lack interpretability but are also highly effective and widely used predictive models ([55],[24],[17]). These models predict the probability of a patient dying or experiencing graft failure within one year (the dependent variable), given his or her characteristics (the independent variables). To make such predictions, the models were first trained on historical observations of independent variables and their associated dependent variables. Once trained, the models predict the dependent variables, given observations of the independent variables, which were potentially previously unseen by the model.

Beyond these machine learning models, models that were constructed by others or proposed in the literature were also investigated. For livers this includes the DRI, which was reported and used in SRTR, and the D-MELD and BAR scores, which were proposed in the literature ([41],[32],[34]). For lung data the LAS score was used, and for kidney data KDPI was used ([67]). For these cases logistic regression and tree models were trained with the scores as features predicting graft failure.

4.4.2 Model Calibration

Observations were randomly split into training, validation, and testing sets. Specifically, 50% of observations were assigned to the training set, 20% to the validation set and 30% to the testing set. The models were fit on the training set and then the out-of-sample AUC value for the validation set was computed. The models that yielded the highest AUC for the validation set were selected and applied to the testing data. For OCTs and OPTs, depths from 1 to 8 were used in the validation procedure. The package itself also tunes the complexity parameters and the minimum number of points per leaf of the tree. For XGBoost, depths from 1 to 8 and eta parameters from .001 to 3 were used in the validation procedure. For random forests, depths from 1 to 8 and number of trees from 10 to 1000 were used in the validation procedure. Lastly, for neural networks, the number of hidden layers was tuned from 2 to 6, the number of nodes per hidden layer from 50 to 500, the gradient descent step size from 0.001 to 1, and regularization parameter from 0.00001 to 1.

4.4.3 Out-of-sample AUC

Performance was evaluated by measuring out-of-sample Area Under the Curve (AUC) on the testing set. A model's AUC corresponds to the probability that a randomly drawn observation whose dependent value was 1 (i.e., a patient who died or experienced graft failure) has a higher score under that model than a randomly drawn observation whose dependent values was 0 ([42]). Therefore, the models' AUC values measures their ability to differentiate patients who will die or experience graft failure within one year from ones who will not.

4.4.4 Disclaimer

This study used data from the Standard Transplant Analysis and Research (STAR) dataset. The STAR data system includes data on all donor, wait-listed candidates, and transplant recipients in the US, submitted by the members of the Organ Procurement and Transplantation Network (OPTN). The Health Resources and Services Administration (HRSA), U.S. Department of Health and Human Services provides oversight to the activities of the OPTN contractors.

4.5 Results

4.5.1 Liver Data

The results for the models trained on liver data are as follows in Table 4.1.

Method	**Out-of-sample R^2**
D-MELD	0.58
BAR	0.61
DRI	0.54
Random Forest	0.64
XGBoost	0.64
Neural Networks	0.66
OCTs	0.61

Table 4.1: Results for liver transplant models.

For liver transplant recipients, neural networks had the best out-of-sample AUC for predicting one-year graft failure, followed closely by the other black box models,

the OCT model, and the model only using the BAR score. The best AUC is the low 0.66, indicating there is not very much signal in the liver data for predicting post-transplant outcomes.

4.5.2 Lung Data

The results for the models trained on lung data are as follows in Table 4.2.

Method	**Out-of-sample R^2**
LAS	0.55
Random Forest	0.62
XGBoost	0.61
Neural Networks	0.60
OCTs	0.60

Table 4.2: Results for lung transplant models.

For lung transplant recipients, the best model is the random forest, followed closely by XGBoost and then the OCT and neural network models. The best AUC is 0.62, indicating there is not very much signal in the lung data either.

4.5.3 Kidney Data

The results for the models trained on kidney data are as follows in Table 4.3.

Method	**Out-of-sample R^2**
KDPI	0.57
Random Forest	0.65
XGBoost	0.65
Neural Networks	0.67
OCTs	0.62

Table 4.3: Results for kidney transplant models.

For kidney transplant recipients, the best model is neural networks, followed by XGBoost and random forests and then the OCT model. The best AUC of any of the models is another low value, in this case 0.67.

4.6 Discussion

By constructing new models, by evaluating existing models, and by studying comprehensive datasets across a variety of organs, a common picture emerged across the board in this systematic study. An AUC of 0.67 means that when it comes to allocation these models would very frequently misclassify or fail to stratify the patient who has a higher post-transplant survival compared to one who has a lower chance of post-graft survival. The models therefore do not effectively predict post-transplant outcomes for patients.

The machine learning methods used to predict post-transplant survival are state-of-the-art modeling techniques. Neural networks are amongst the most widely used machine learning methods with a wide variety of successful applications, and optimal classification trees are a state-of-the-art interpretable machine learning method. Simpler versions of these models such as logistic regression have been applied ef-

fectively in the transplant area (ex. in defining MELD ([100]). Others have also used these methods to improve upon those earlier successes, such as applying optimal classification trees to predict mortality for patients on the liver transplantation waiting list ([11]), which was shown to reduce waitlist mortality compared to MELD in an analysis involving the Liver Simulated Allocation Model (LSAM). However, the application of these models to post-transplant survival resulted in poor performance. This suggests that with the current machine learning technology and data that are available at the time of allocation, it does not seem plausible that there will be a model that will be accurate enough to include as part of the decision/allocation process in the near future.

Our hypothesis is that the key limitation comes from data. Given the successes all the machine learning methods have had across applications and even within transplantation, it seems unlikely to us that they would be the primary issue. There are several hypotheses as to why the data at time of transplantation might not be sufficient for making predictions.

One hypothesis is that the post-transplant outcome could be a very complicated function of the recipient, donor, and procedure information, and fundamentally at time of allocation there isn't any procedure data. Trying to make a prediction about post-transplant survival outcomes without procedure information might be akin to trying to predict who will win the Super Bowl at the beginning of the season, prior to observing how the teams perform or any injuries that might occur. There is data about the players themselves, but not about how they perform in action, so the performance of predictive models would be limited. After the regular season there would probably be much better data for making predictions. Likewise, after surgery, it would likely be possible to make a much better prediction about post-transplant outcomes with the data about how the surgery went and any complications a patient

experienced after the procedure.

A second hypothesis is that the current data collected about donors and recipients might not be diverse enough to explain post-transplant outcomes. Other data, such as biopsies, scans of the patients' bodies, or genetic information might be crucial influences in post-transplant outcomes. In that case trying to predict survival outcomes without them would achieve limited success.

Overall, though, the results indicate post-transplant survival predictions should most likely not be used to stratify patients or as a factor in organ allocation decisions for now.

Chapter 5

Conclusion

In this thesis, we have studied various aspects of interpretable machine learning and its application in the health care domain.

Chapter 2 compares decision trees and neural networks both theoretically and empirically. We prove that given a neural network one can construct an equivalent decision tree. In some cases this necessitates decision trees of great depths, so we further empirically compare the performances of the two methods on a variety of data sets and find that even without theoretical guarantees the two methods achieve similar performances in practice. We therefore show that in many applications decision trees can be used without loss of predictive performance compared to a cutting edge black box method, but with a significant gain in interpretability.

Chapter 3 proposes a novel machine learning method called Optimal Predictive Clustering (OPC), which simultaneously clusters and learns cluster-specific regression models utilizing a combination of mixed-integer optimization methods and strong warm starts. We find that OPC achieved a combination of predictive performance, scalability, and interpretability that is competitive with other cutting edge

interpretable machine learning methods, and that when included in an ensemble of models alongside other models such as XGBoost and Model Trees it achieves a predictive performance that outperforms all other methods. The proposed method therefore improves upon the state-of-the art for both interpretable methods and overall predictive performance for regression problems.

Chapter 4 investigates how well one can predict transplant outcomes given pre-transplant information for liver, lung, and kidney transplants. We find that with the current data and machine learning models that we are unable to train models that would reliably differentiate between patients who would experience graft failure within one year and those who would not. We therefore provide important evidence that post-transplant survival predictions should most likely not be used as a factor in organ allocation decisions for now.

Together, these chapters illustrate the theoretical and practical effectiveness of interpretable methods, especially in the health care domain. We hope that the strengths and efficacy of these interpretable techniques will encourage their scientific usage and in particular support the overall goal in health care of achieving better outcomes for patients so that more people can live longer lives with higher quality of life.

Bibliography

[1] M. W. Ahmad, M. Mourshed, and Y. Rezgui. Trees vs Neurons: Comparison between random forest and ANN for high-resolution prediction of building energy consumption. *Energy and Buildings*, 2017.

[2] E. Angün and A. Altınoy. A new mixed-integer linear programming formulation for multiple responses regression clustering. In *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 1634–1639, April 2019.

[3] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[4] A. M. Bagirov, J. Ugon, and H. G. Mirzayeva. An algorithm for clusterwise linear regression based on smoothing techniques. *Optimization Letters*, 9(2):375–390, Feb 2015.

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] D. Bertsimas, K. O. Allison, and W. R. Pulleyblank. *The Analytics Edge*. Dynamic Ideas, 2016.

[7] D. Bertsimas and J. Dunn. Optimal classification trees. *Machine Learning*, pages 1–44, 2017.

[8] D. Bertsimas and J. Dunn. *Optimal Trees for Prediction and Prescription*. Dynamic Ideas, 2018.

[9] D. Bertsimas and J. Dunn. *Machine Learning under a Modern Optimization Lens*. Dynamic Ideas Press, 2019.

[10] D. Bertsimas, A. King, and R. Mazumder. Best subset selection via a modern optimization lens. *Ann. Statist.*, 44(2):813–852, 04 2016.

[11] D. Bertsimas, J. Kung, N. Trichakis, Y. Wang, R. Hirose, and P. A. Vagefi. Development and validation of an optimized prediction of mortality for candidates awaiting liver transplantation. *American Journal of Transplantation*, 19(4):1109–1118, 2019.

[12] D. Bertsimas, C. Pawlowski, and Y. Zhuo. From predictive methods to missing data imputation: an optimization approach. *The Journal of Machine Learning Research*, 18(1):7133–7171, 2017.

[13] D. Bertsimas and R. Shioda. Classification and regression via integer optimization. *Operations Research*, 55:252–271, 04 2007.

[14] G. Biau, E. Scornet, and J. Welbl. Neural Random Forests, 2018.

[15] H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees, 2000.

[16] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[17] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[18] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[19] J. Briceño, M. Cruz-Ramírez, M. Prieto, M. Navasa, J. Urbina, R. Orti, M. Á. Bravo, A. Otero, E. Varo, S. Tome, G. Clemente, R. Bañares, R. Bárcena, V. Cuervas-Mons, G. Solorzano, C. Vinaixa, A. Rubín, J. Colmenero, A. Valdivieso, and M. García. Use of artificial intelligence as an innovative donor-recipient matching model for liver transplantation: Results from a multicenter Spanish study. *Journal of hepatology*, J Hepatol. 2014 Nov:1020–8, 11 2014.

[20] R. Carbonneau, G. Caporossi, and P. Hansen. Extensions to the repetitive branch and bound algorithm for globally optimal clusterwise regression. *Computers & Operations Research*, 39:2748–, 11 2012.

[21] R. A. Carbonneau, G. Caporossi, and P. Hansen. Globally optimal clusterwise regression by mixed logical-quadratic programming. *European Journal of Operational Research*, 212:213–222, 2010.

[22] R.A. Carbonneau, G. Caporossi, and P. Hansen. Globally optimal clusterwise regression by column generation enhanced with heuristics, sequencing and ending subset optimization. *Journal of Classification*, 31(2):219–241, Jul 2014.

[23] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd international conference on Machine learning*, 2006.

[24] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[25] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[26] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[27] W. Desarbo, R. Oliver, and A. Rangaswamy. A simulated annealing methodology for clusterwise linear regression. *Psychometrika*, 54:707–736, 02 1989.

[28] W. S. DeSarbo and W. L. Cron. A maximum likelihood methodology for clusterwise linear regression. *Journal of Classification*, 5(2):249–282, Sep 1988.

[29] E. Devijver. Model-based regression clustering for high-dimensional data: application to functional data. *Advances in Data Analysis and Classification*, 11:243–279, 2016.

[30] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.

[31] D. Dua and C. Graff. UCI machine learning repository, 2017.

[32] P. Dutkowski, C. E. Oberkofler, K. Slankamenac, M. A. Puhan, E. Schadde, B. Müllhaupt, A. Geier, and P. A. Clavien. Are there better guidelines for allocation in liver transplantation? *Annals of Surgery*, 254(5):745–754, 2011.

[33] T. M. Egan, S. Murray, R. T. Bustami, T. H. Shearon, K. P. McCullough, L. B. Edwards, M. A. Coke, E. R. Garrity, S. C. Sweet, D. A. Heiney, and F. L. Grover. Development of the new lung allocation system in the united states. *American Journal of Transplantation*, 6(5p2):1212–1227, 2006.

[34] S. Feng, N. P. Goodrich, J. L. Bragg-Gresham, D. M. Dykstra, J. D. Punch, M. A. DebRoy, S. M. Greenstein, and R. M. Merion. Characteristics associated with liver graft failure: The concept of a donor risk index. *American Journal of Transplantation*, 6(4):783–790, 2006.

[35] N. Frosst and G. Hinton. Distilling a Neural Network Into a Soft Decision Tree. *arXiv*, 2017.

[36] I. Gitman, J. Chen, E. Lei, and A. Dubrawski. Novel prediction techniques based on clusterwise linear regression. *ArXiv*, abs/1804.10742, 2018.

[37] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[38] Google. Google search, 2018. Online; accessed 2018-01-11.

[39] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6645–6649. IEEE, 2013.

[40] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc., 2009.

[41] J. B. Halldorson, R. Bakthavatsalam, O. Fix, J. D. Reyes, and J. D. Perkins. D-meld, a simple predictor of post liver transplant mortality for optimization of donor/recipient matching. *American Journal of Transplantation*, 9(2):318–326, 2009.

[42] J. A. Hanley and B. J. Mcneil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 1982.

[43] P Hansen and G Caporossi. Variable neighborhood search for least squares clusterwise regression. 2005.

[44] D. Harrison Jr and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[45] K. He, X. Zhang, S. Ren, and J. 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.

[46] D. P. Helmbold and P. M. Long. On the inductive bias of dropout. *Journal of Machine Learning Research*, 16:3403–3454, 2015.

[47] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[48] K. D. Humbird, J. L. Peterson, and R. G. McClarren. Deep neural network initialization with decision trees. 2018.

[49] LLC Interpretable AI. Interpretable AI documentation, 2019.

[50] I. Ivanova and M. Kubat. Initialization of neural networks by means of decision trees. *Knowledge-Based Systems*, 8(6):333–344, 1995.

[51] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò. Deep neural decision forests. In *IJCAI International Joint Conference on Artificial Intelligence*, 2016.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[53] A. Kurenkov. A 'brief' history of neural nets and deep learning, part 1, 2015. Online; accessed 2017-09-10.

[54] K. Lau, P. L. Leung, and K. Tse. A mathematical programming approach to clusterwise regression model and its extensions. *European Journal of Operational Research*, 116:640–652, 1999.

[55] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[56] Y. A. LeCun, L. Bottou, G. B. Orr, and K. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[57] J. Lee, M. Kang, and J. Kang. Ensemble of binary tree structured deep convolutional network for image classification. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1448–1451, Dec 2017.

[58] A. Li, S. Luo, Y. Liu, and H. Yu. The equivalency between a decision tree for classification and a feedback neural network. *Proceedings 7th International Conference on Signal Processing, 2004. Proceedings. ICSP 04. 2004.*, 2004.

[59] M. Lichman. UCI machine learning repository, 2013.

[60] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[61] N. Manwani and P. S. Sastry. K-Plane regression. *CoRR*, abs/1211.1513, 2012.

[62] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[63] J. Meier. A fast algorithm for clusterwise linear absolute deviations regression. *Operations-Research-Spektrum*, 9(3):187–189, Sep 1987.

[64] C. Meynet and C. Maugis-Rabusseau. A sparse variable selection procedure in model-based clustering. 2012.

[65] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[66] V. N. Murthy, V. Singh, T. Chen, R. Manmatha, and D. Comaniciu. Deep decision network for multi-class image classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2240–2248, June 2016.

[67] U.S. Department of Health and Human Services. KDPI calculator. 2017.

[68] R. S. Olson, W. La Cava, Z. Mustahsan, Z. Varik, and J. H. Moore. Data-driven advice for applying machine learning to bioinformatics problems, 2018.

[69] Y. W. Park, W. Jiang, D. Klabjan, and L. Williams. Algorithms for generalized clusterwise linear regression. *INFORMS Journal on Computing*, 29:301–317, 2017.

[70] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[71] J. R. Quinlan. Learning with continuous classes. pages 343–348. World Scientific, 1992.

[72] S. Raschka. Single-layer neural networks and gradient descent, 2015. Online; accessed 2017-09-9.

[73] G. Raskutti, M. J. Wainwright, and B. Yu. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *Journal of Machine Learning Research*, 15(1):335–366, 2014.

[74] D. L. Richmond, D. Kainmüller, M. Y. Yang, E. W. Myers, and C. Rother. Relating cascaded random forests to deep convolutional neural networks for semantic segmentation. *CoRR*, abs/1507.07583, 2015.

[75] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton project para. Technical report, Cornell Aeronautical Laboratory, 1957.

[76] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[77] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[78] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[79] M. R. Segal. Tree-structured methods for longitudinal data. *Journal of the American Statistical Association*, 87(418):407–418, 1992.

[80] I. K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, Oct 1990.

[81] I. K. Sethi. Decision tree performance enhancement using an artificial neural network implementation. *Artificial Neural Networks and Statistical Pattern Recognition - Old and New Connections Machine Intelligence and Pattern Recognition*, page 71?88, 1991.

[82] I. K. Sethi. Neural implementation of tree classifiers. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(8):1243–1249, Aug 1995.

[83] I. K. Sethi and J. H. Yoo. Structure-driven induction of decision tree classifiers through neural learning. *Pattern Recognition*, 30(11):1893–1904, 1997.

[84] I. K. Sethi, J. H. Yoo, and C. M. Brickman. Extraction of diagnostic rules using neural networks. *[1993] Computer-Based Medical Systems-Proceedings of the Sixth Annual IEEE Symposium*, 1993.

[85] R. Setiono and W. K. Leow. On mapping decision trees and neural networks. *Knowledge-Based Systems*, 12(3):95–99, 1999.

[86] R. Setiono and H. Liu. Understanding neural networks via rule extraction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 480–485, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[87] R. Socher, B. Huval, B. Bhat, C. D. Manning, and A. Y. Ng. Convolutional-Recursive Deep Learning for 3D Object Classification. In *Advances in Neural Information Processing Systems 25*. 2012.

[88] H. Späth. Algorithm 39 clusterwise linear regression. *Computing*, 22(4):367–373, Dec 1979.

[89] H. Späth. Cluster analysis algorithms for data reduction and classification of objects. 1980.

[90] H. Späth. Clusterwise linear least absolute deviations regression. *Computing*, 37(4):371–377, Dec 1986.

[91] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[92] N. Städler, P. Bühlmann, and S. A. van de Geer. L1-penalization for mixture regression models. *TEST*, 19:209–256, 2010.

[93] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[94] S. Thrun. Extracting Rules from Artificial Neural Networks with Distributed Representations. *Advances in Neural Information Processing Systems 7*, 1995.

[95] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[96] L. Torgo. LIACC regression data sets, 2019.

[97] G. G. Towell and J. W. Shavlik. Extracting Refined Rules from Knowledge-Based Neural Networks. *Machine Learning*, 1993.

[98] Y. Wang and I. Witten. Induction of model trees for predicting continuous classes. *Induction of Model Trees for Predicting Continuous Classes*, 01 1997.

[99] P. Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.

[100] R. Wiesner, E. Edwards, R. Freeman, A. Harper, R. Kim, P. Kamath, W. Kremers, J. Lake, T. Howard, R. M. Merion, and et al. Model for end-stage liver disease (MELD) and allocation of donor livers. *Gastroenterology*, 124(1):91–96, 2003.

[101] L. R. Wingfield, C. Ceresa, S. Thorogood, J. Fleuriot, and S. Knight. Using artificial intelligence for predicting survival of individual grafts in liver transplantation: A systematic review. *Liver Transplantation*, 26(7):922–934, 2020.

[102] Y. Xiao and M. R. Segal. Identification of yeast transcriptional regulation networks using multivariate random forests. *PLoS Comput Biol.*, 06 2009.

[103] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[104] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. *Facial Landmark Detection by Deep Multi-task Learning*. Springer International Publishing, 2014.