Journal of Machine Learning Research 1 (2018) 1-48 Submitted ?/17; Published ?/17

Optimal classification and regression trees with hyperplanes are as powerful as classification and regression neural networks

Dimitris Bertsimas DBERTSIM@MIT.EDU
Sloan School of Management and Operations Research Center
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Rahul Mazumder RAHULMAZ@MIT.EDU
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Matthew Sobiesk MSOBIESK@MIT.EDU
Operations Research Center
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Editor: –

April 1, 2018

Abstract

We investigate the modeling power of neural networks in comparison with optimal classification and regression trees with hyperplanes (OCT-Hs and ORT-Hs). We prove 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. Conversely, we show that a given classification OCT-H can be transformed to a classification neural network with the perceptron activation function with the same accuracy in the training set, showing that a neural network is at least as powerful as OCT-Hs, that is, OCT-Hs and neural networks are equivalent in terms of modeling power. Given that our construction of optimal trees that emulate a given neural network necessitates the construction of deep trees, we explored the practical implication of these findings. We report a comparison of optimal classification trees and neural networks on seven well known data sets and show that the two methods exhibit remarkably close accuracy.

Keywords: Neural Networks, Optimal Classification and Regression Trees, Mixed Integer Optimization

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, McCulloch and Pitts (1943) 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 (Rosenblatt (1957)), the development of backpropagation, a technique to train neural networks (Werbos (1974), Rumelhart et al. (1988)), the proof that multilayer feedforward neural networks are universal approximators (Kurenkov (2015), Hornik et al. (1989)), 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 Raschka (2015), LeCun et al. (2015), and Schmidhuber (2015).

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 $i = 1, \ldots, N_\ell$.
- 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 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.

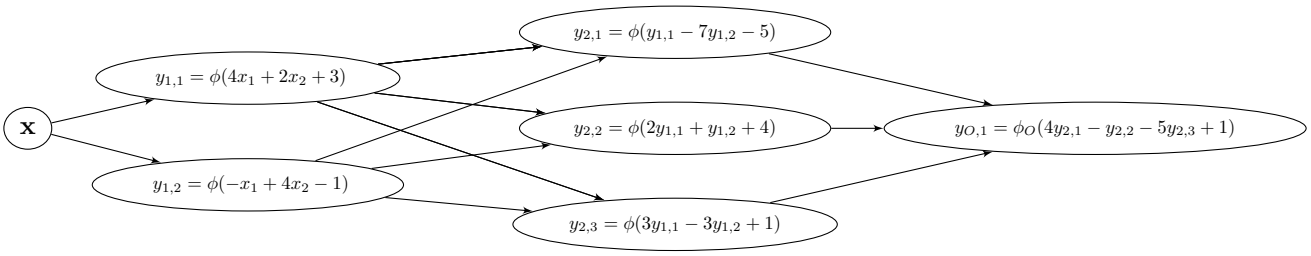

Figure 1: An example of a feedforward neural network.

Deep learning has had some great successes across a variety of applications. For example, it revolutionized image recognition (Socher et al. (2012), Graves and Schmidhuber (2009), He et al. (2015)), with competitors in the ImageNet Large Scale Visual Recognition Challenge almost exclusively using deep learning in their programs (Russakovsky et al. (2014), Krizhevsky et al. (2012)). 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 (Mikolov et al. (2010), Cho et al. (2014), Bahdanau et al. (2014), Sutskever et al. (2014)). Additionally, speech recognition technology has markedly improved with the inclusion of deep learning (Goodfellow et al. (2016), Graves et al. (2013)).

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" (Google (2018)), 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 (Srivastava et al. (2014),

Zaremba et al. (2014), Helmbold and Long (2015)) and early stopping (Raskutti et al. (2014), Zhang et al. (2014)). 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 (Bertsimas et al. (2016)). In the words of Leo Breiman, "On interpretability, trees rate an A+" (Breiman et al. (1984)). 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. 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 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.

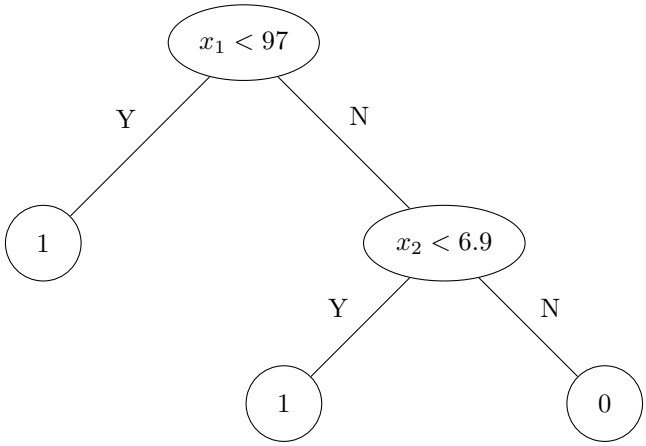

Figure 2: A sample decision tree.

Originally proposed by Breiman et al. (1984) 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 (Bertsimas and Dunn (2017, 2018)). Using mixed integer optimization and local search methods the authors find optimal classification trees (OCT) and optimal regression trees (ORT) that significantly improve upon CART. 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 (Bertsimas and Dunn (2018)). 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 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 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.

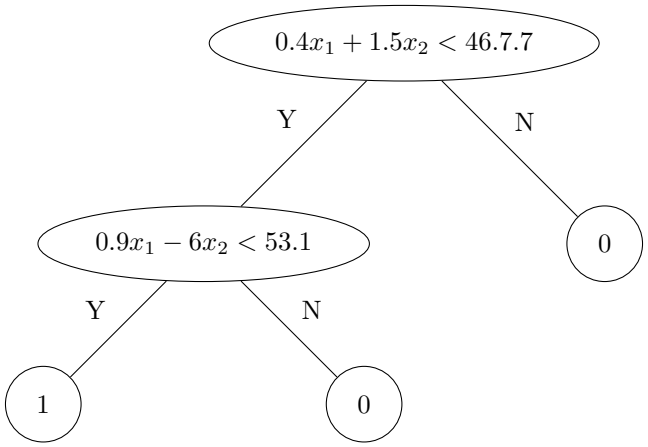

Figure 3: A sample decision tree with hyperplane splits.

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. Conversely, a given classification tree with hyperplanes can be transformed to a classification neural network with the same accuracy in the training set, showing that these neural networks are at least as powerful as OCT-Hs. Consequently, we show that OCT-Hs and neural networks are equivalent in terms of modeling power. 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 seven 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 1 we show the parameters of our construction and the section our construction appears in.

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	3.1
Classification Feedforward Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = \max(x, 0)$	OCT-H	• Depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$	3.3

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	4.1
Classification Convolutional Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = \max(x, 0)$	OCT-H	• Depth $q - 1 + \sum_{\ell=1}^{L} N_\ell$	4.3
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 $T^* \times N_1$	5.1
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 $q - 1 + T^* \times N_1$	5.3
OCT-H	• N_1 splits • N_2 leaf nodes • q classification values	Classification Feedforward Neural Network	• 2 hidden layers • N_1 nodes in the first hidden layer • N_2 nodes in the second • $\phi(x) = 1\{x \geq 0\}$	6

Table 1: Summary of the relationship of classification NNs and OCT-Hs.

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

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	3.2
Regression Feedforward Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = \max(x, 0)$	ORT-H	• Depth $\sum_{\ell=1}^{L} N_\ell$	3.4
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	4.2
Regression Convolutional Neural Network	• L hidden layers • N_ℓ nodes in each hidden layer • $\phi(x) = \max(x, 0)$	ORT-H	• Depth $\sum_{\ell=1}^{L} N_\ell$	4.4
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 $T^* \times N_1$	5.2
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 $T^* \times N_1$	5.4

Table 2: Summary of the relationship of regression NNs and ORT-Hs.

3. We show in Section 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 six 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, 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 of choice in applications where interpretability matters.

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, we review the mathematical structure of the neural networks we consider, as well as decision trees. In Sections 3, 4 and 5, we prove that we can transform feedforward, convolutional and recurrent neural networks into decision trees, respectively. In Section 6, we discuss how to transform classification decision trees into classification feedforward neural networks. In Section 7, we compare the classification accuracies of comparable decision trees and neural networks on seven data sets. Lastly, in Section 8, we include our concluding remarks.

2. 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.

2.1 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

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

  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 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").

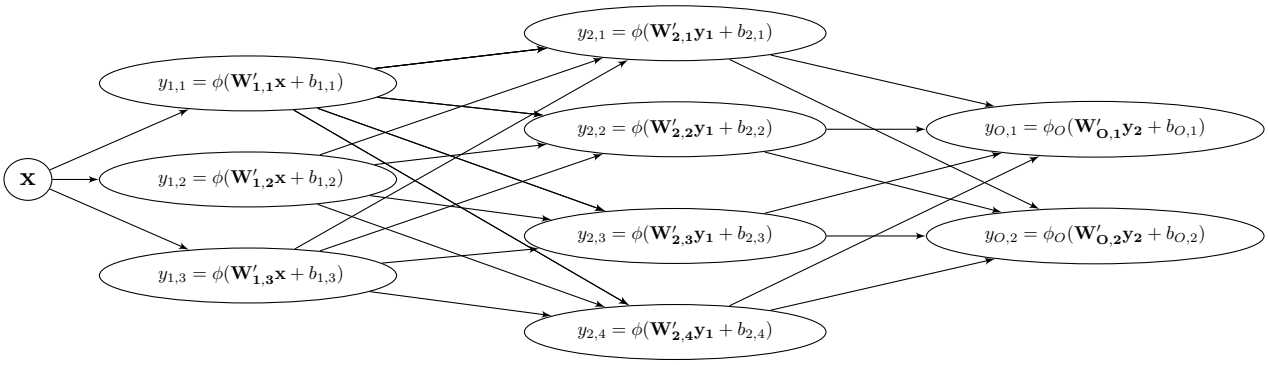

Figure 4: An example of a classification feedforward neural network.

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

$$\phi(x) = 1\{x \geq 0\}, \tag{2}$$

and the rectified linear unit or ReLU, where

$$\phi(x) = \max(x, 0). \tag{3}$$

If (2) is chosen as the activation function, then $(\phi_O(\mathbf{x}))_i = 1\{x_i \geq 0\}$. If (3) is chosen as the activation function, then $\phi_O(\mathbf{x})$ is defined as

$$(\phi_O(\mathbf{x}))_i = \begin{cases} 1, & \text{where } i = \text{argmax}_{i=1,\ldots,N_0}(x_i), \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

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 (Bottou (2010), LeCun et al. (2012)), and is typically augmented with techniques like early stopping (Raskutti et al. (2014), Zhang et al. (2014)) and dropout (Srivastava et al. (2014), Zaremba et al. (2014), Helmbold and Long (2015)). Due to the heuristic nature of these methods, we are not guaranteed an optimal solution for $\mathbf{W}_\ell$ and $\mathbf{b}_\ell$, $\ell = 1, \ldots, L, O$.

2.2 Convolutional Neural Networks

A convolutional neural network (CNN) (Schmidhuber (2015), Goodfellow et al. (2016)) 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 2.1.
- 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:

$$P_{\ell,i} = \max_{m \in S_{\ell,i}} (y_{\ell,m}),\ i = 1, \ldots, R_\ell. \tag{5}$$

- 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. (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 Recurrent Neural Networks

Recurrent neural networks (RNNs) are specialized to be able to handle sequential input data (Lipton et al. (2015)). 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. (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 $\mathbf{W}_O^T \mathbf{y}_{T^*,1} + \mathbf{b}_O$.

2.4 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 $i = 1, \ldots, \lfloor T/2 \rfloor$. Bertsimas and Dunn (2018) have shown that 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. 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 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.

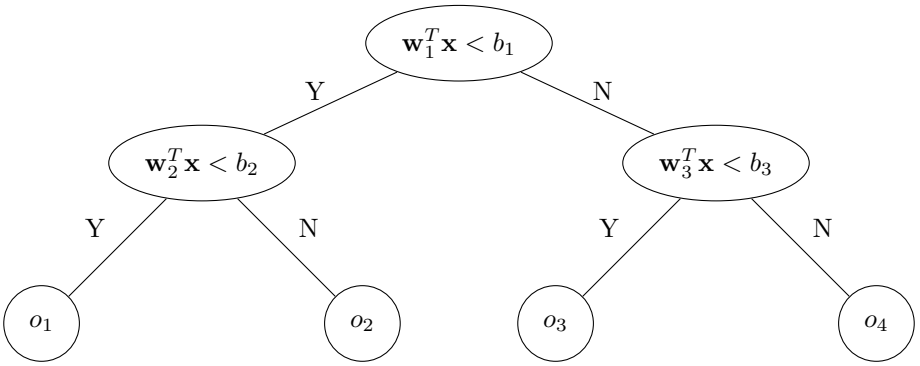

Figure 5: An OCT-H of depth 2. Data in the four leaf nodes are classified as o_1, o_2, o_3, and o_4.

3. 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.

3.1 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) 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).
- 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 output values, which we can assign to the 2^{N_1} leaf nodes of $\mathcal{T}_1$. It follows that an OCT-H of maximum depth N_1 has at least the same classification accuracy as $\mathcal{N}_1$.

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 of $\mathcal{T}_1$ as

$$\mathbf{W}_{1,1}^T\mathbf{x} + b_{1,1} < 0, \tag{6}$$

which results in the simple split seen in Figure 6.

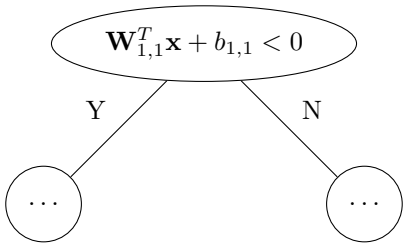

Figure 6: The first split of decision tree $\mathcal{T}_1$.

Independent of whether inequality (6) is satisfied or not, the second split is given by

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

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

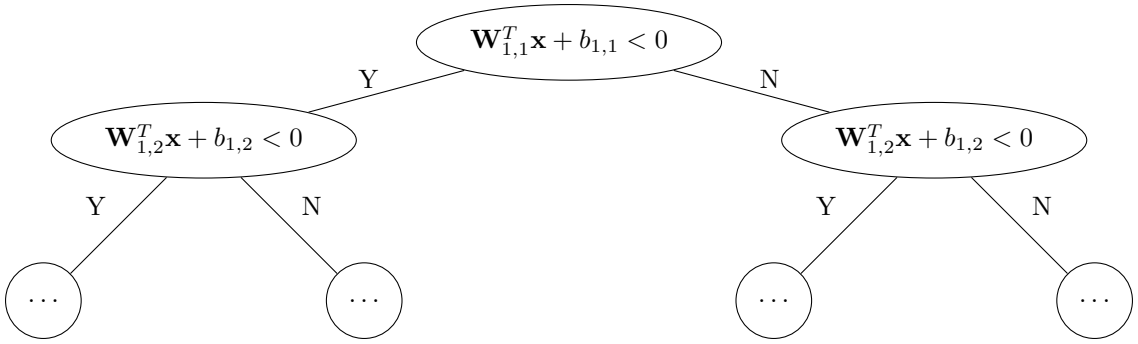

Figure 7: The first two depths of tree $\mathcal{T}_1$.

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 2.1. 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 $\phi_O(\mathbf{x})$.

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 8, and the resultant decision tree is shown in Figure 9.

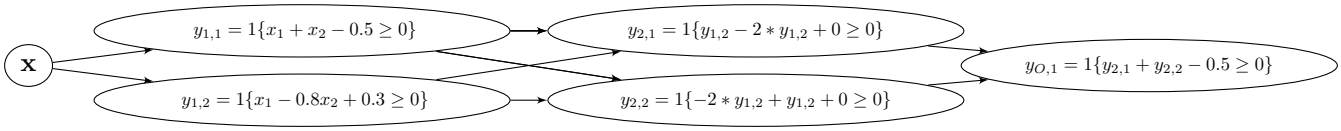

Figure 8: An FNN with the perceptron activation function performing an XOR operation.

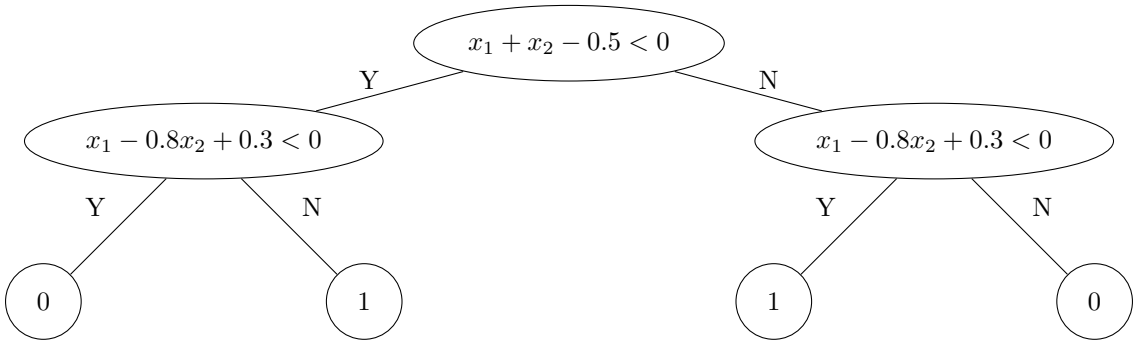

Figure 9: The reformulation of the Figure 8 neural network into a decision tree.

The network in Figure 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 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 9 tree by the optimality of the final OCT-H. Thus, the optimal tree does at least as well as the Figure 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.

3.2 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 3.1 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.

3.3 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. 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 (3), 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 (3).
- 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 $\mathcal{T}_2$ 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 3.1. This part of the tree is shown in Figure 10.

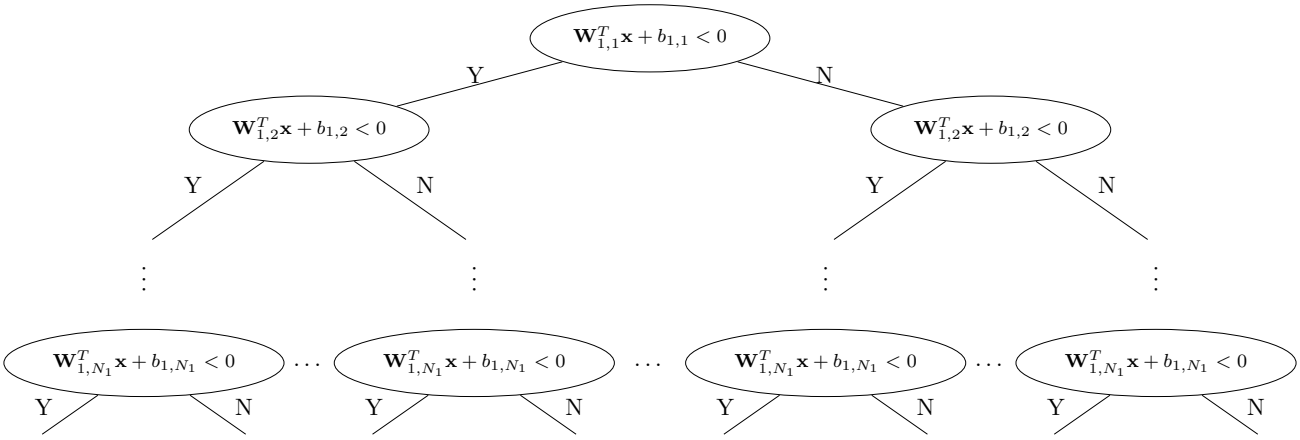

Figure 10: The decision tree $\mathcal{T}_2$ we are building up to depth N_1.

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 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 10. In each of these branches we have implicitly calculated the 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 10, but with the corresponding value of $\mathbf{y}_1$ playing the role of $\mathbf{x}$.

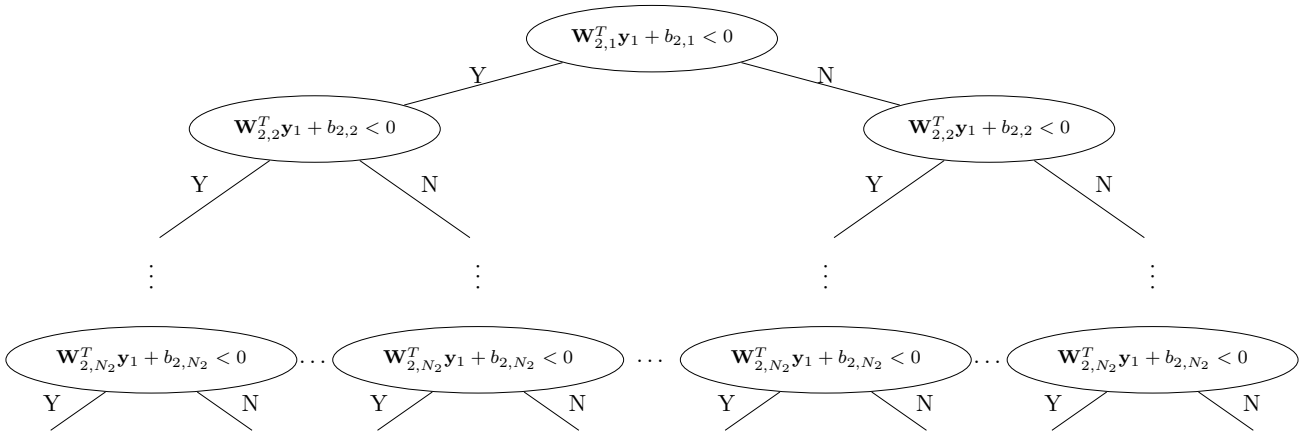

Figure 11: Subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ of depth N_2 is concatenated to the corresponding branch of the subtree depicted in Figure 10, resulting in a subtree of depth N_1+N_2.

In the subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ in Figure 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 12.

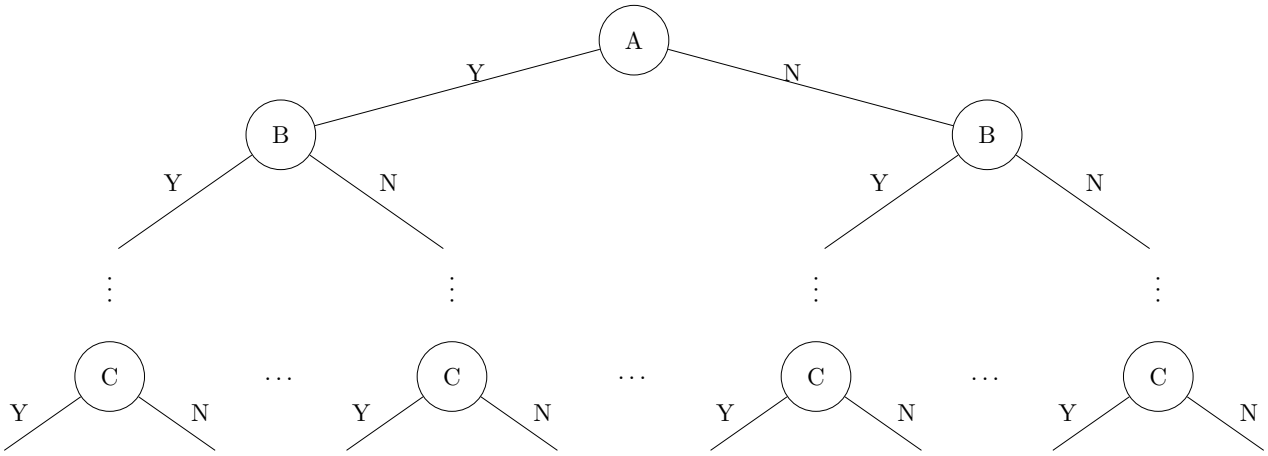

Figure 12: The resulting subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ for $\mathbf{y}_1 = (0, \ldots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})$. A, B, C are as follows:

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

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

$$\text{argmax}_{i=1,\ldots,q}(\mathbf{W}_{O,i}^T \mathbf{y}_L + b_{O,i}) \tag{7}$$

in order to find the output class. The construction of the subtree $\mathcal{T}_{2,O}(\mathbf{y}_L)$ is depicted in Figure 13.

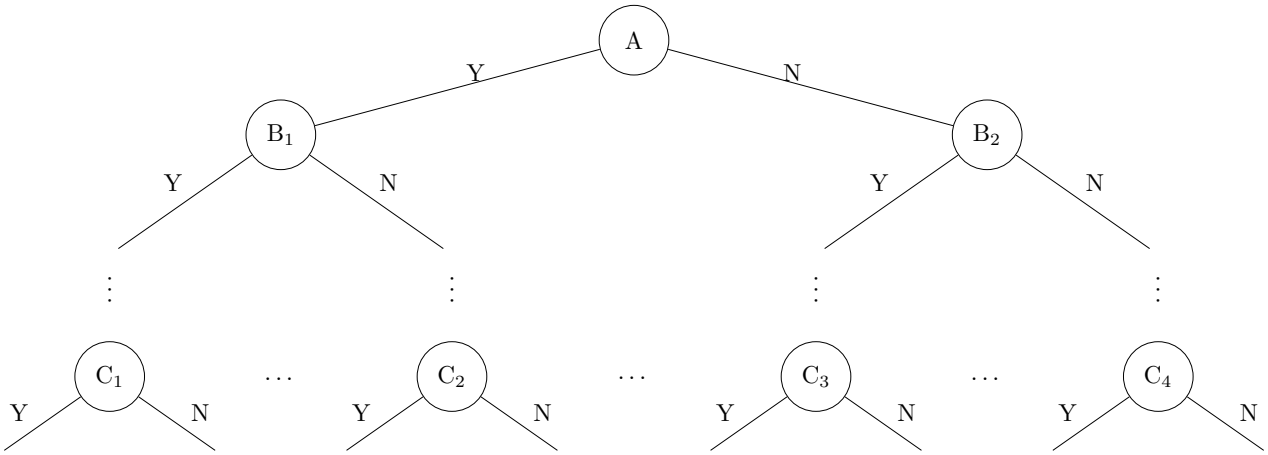

Figure 13: The subtree $\mathcal{T}_{2,O}(\mathbf{y}_L)$. The labels $A, B_1, B_2, C_1, C_2, C_3, C_4$ are as follows:

- 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 $(\mathbf{W}_{O,1} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,1} - b_{O,q} < 0$.

We simulate the calculation of (7) 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 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 14, using the output function defined in Eq.(4), and the resulting classification tree with hyperplane splits is shown in Figure 15.

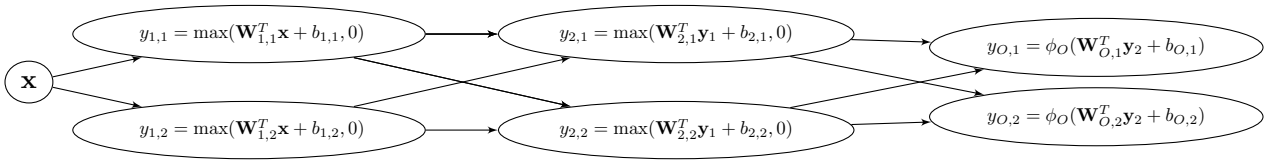

Figure 14: A ReLU FNN.

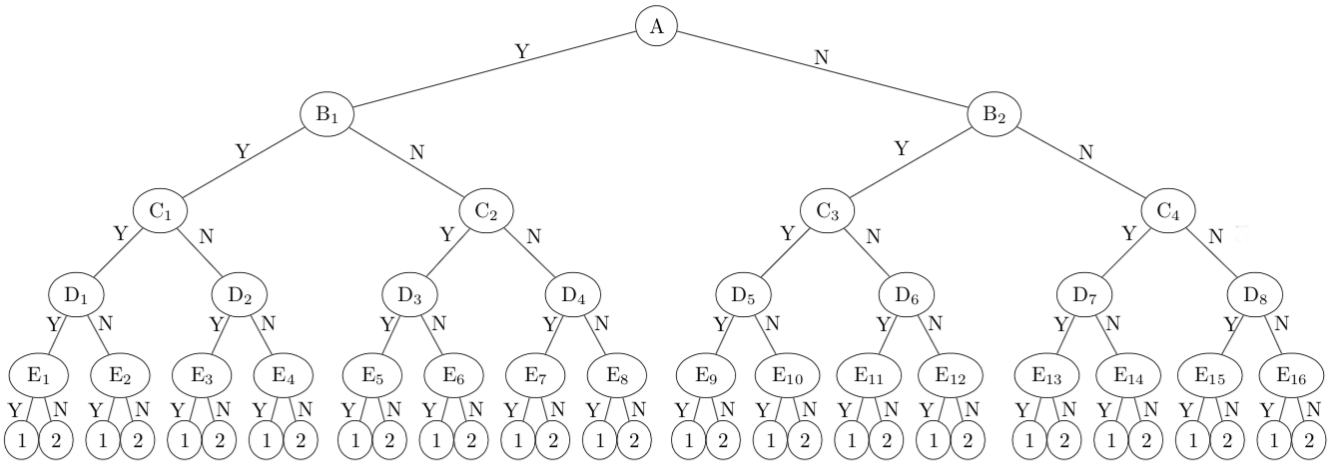

Figure 15: The reformulation of the Figure 14 FNN into a classification tree. The labels $A, B_1, \ldots, E_{16}$ are as follows:

- 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.

3.4 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 3.3 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 16.

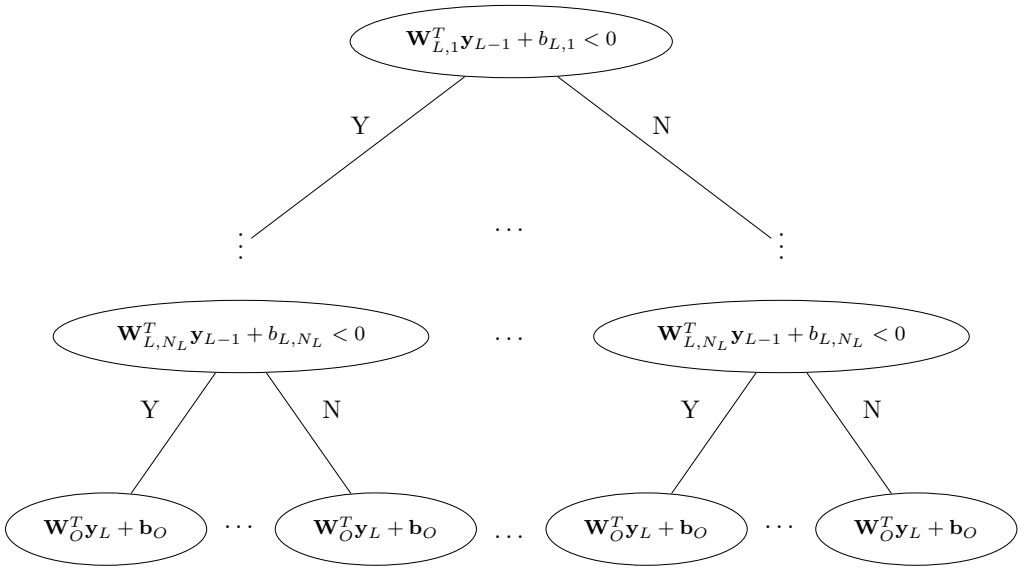

Figure 16: Subtree $\mathcal{T}_{2,L}(\mathbf{y}_{L-1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $\sum_{\ell=1}^{L} N_\ell$. Note that the leaves have linear functions $\mathbf{W}_O^T\mathbf{y}_L + \mathbf{b}_O$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

4. 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 perceptron or Rectified Linear Unit activation functions.

4.1 Perceptron Classification CNNs and OCT-Hs

The key result in this section is as follows.

Theorem 3 *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.*

Proof Our proof is constructive. We are given a CNN $\mathcal{N}_3$ with the following characteristics:

- The perceptron activation function as defined in (2).
- Output function $\phi_O(\mathbf{y}_O) : [0,1]^q \to [0,1]^q$.
- L hidden layers and one output layer, indexed $\ell = 1, \ldots, L$.
- Max pooling layers as defined in (5).
- 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 each 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 2.2.
- q nodes in the output layer.

We construct a decision tree $\mathcal{T}_3$ with hyperplane splits and maximum depth N_1 that makes the same predictions as $\mathcal{N}_3$. It follows that an OCT-H of maximum depth N_1 has at least the same classification accuracy as $\mathcal{N}_3$.

Given the inequality from the first node in the first hidden layer of $\mathcal{N}_3$ defined by the weight vector $\mathbf{W}_{1,1}$ and bias scalar $b_{1,1}$, we define the first split of $\mathcal{T}_3$ as

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

Figure 17 provides a visualization of this split.

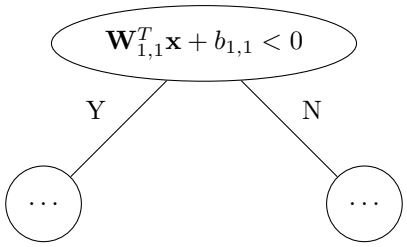

Figure 17: The first split of decision tree $\mathcal{T}_3$.

The second split is given by

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

This new split is visualized in Figure 18.

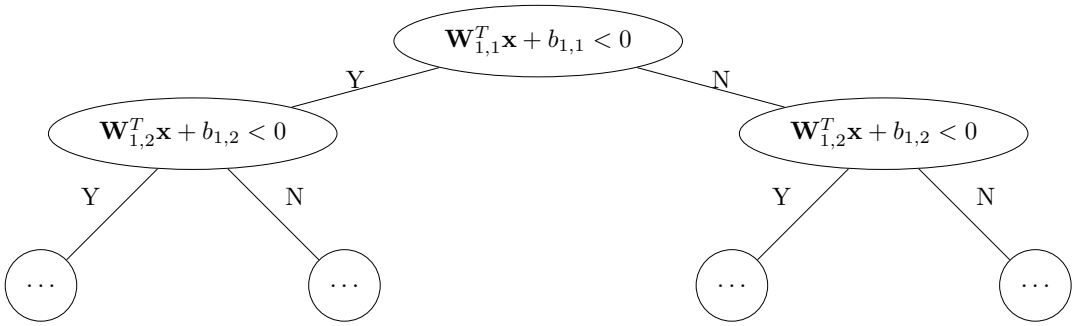

Figure 18: The decision tree we are building up to depth 2.

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

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

To complete the construction of $\mathcal{T}_3$, we need to assign a classification value to every leaf of $\mathcal{T}_3$. Given an input $\mathbf{x}$, there are 2^{N_1} binary vectors that could be output by the first hidden layer of $\mathcal{N}_3$. These 2^{N_1} vectors, by our construction of $\mathcal{T}_3$, exactly correspond to the 2^{N_1} leaves of $\mathcal{T}_3$. Given $\mathbf{y}_1^r$, the output of the first hidden layer associated with leaf node r, the final prediction of $\mathcal{N}_3$ will be $k(\mathbf{y}_1^r)$, which follows deterministically given the $\mathbf{y}_1^r$ vector and the $\mathbf{W}_{\ell,i}$, $b_{\ell,i}$ values by using the calculation process outlined in Section 2.2. In every leaf node r of the tree we assign the classification value $k(\mathbf{y}_1^r)$. Thus, by our construction, for every input $\mathbf{x}$ the output of $\mathcal{T}_3$ is the same as the output of the neural network $\mathcal{N}_3$. Since the optimal tree must do at least as well as $\mathcal{T}_3$, it must do at least as well as $\mathcal{N}_3$, and the proof is complete. ■

4.2 Perceptron Regression CNNs and ORT-Hs

One can extend Theorem 3 in Section 4.1 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 3 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.

4.3 Rectified Linear Unit Classification CNNs and OCT-Hs

While the result in Section 4.1 follows directly from Theorem 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. The next theorem illustrates a different OCT-H construction for this case.

Theorem 4 *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 (3), max pooling layers as defined in (5), and output function defined in Eq. (4).*

Proof Our proof is constructive. We are given a CNN $\mathcal{N}_4$ as described in Section 2.2 with the following characteristics:

- The rectified linear unit activation function defined in (3).
- The output function as defined in (4).
- L hidden layers and one output layer, indexed $\ell = 1, \ldots, L$.
- Max pooling layers as defined in (5).
- 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 2.2.
- 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. (5). Without loss of generality, assume that $S_{1,1} = \{1, \ldots, k_1\}$. Then the first split of $\mathcal{T}_{4,1}$ is

$$(\mathbf{W}_{1,1} - \mathbf{W}_{1,2})^T \mathbf{x} + (b_{1,1} - b_{1,2}) < 0. \tag{8}$$

It is depicted in Figure 19, and determines whether the maximum among the first two hyperplanes is $\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}$ or $\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}$.

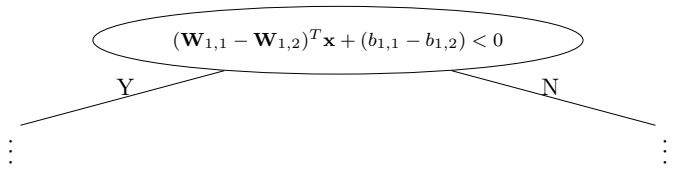

Figure 19: The first split of classification tree $\mathcal{T}_{4,1}$.

If Inequality (8) 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 (8) 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 20.

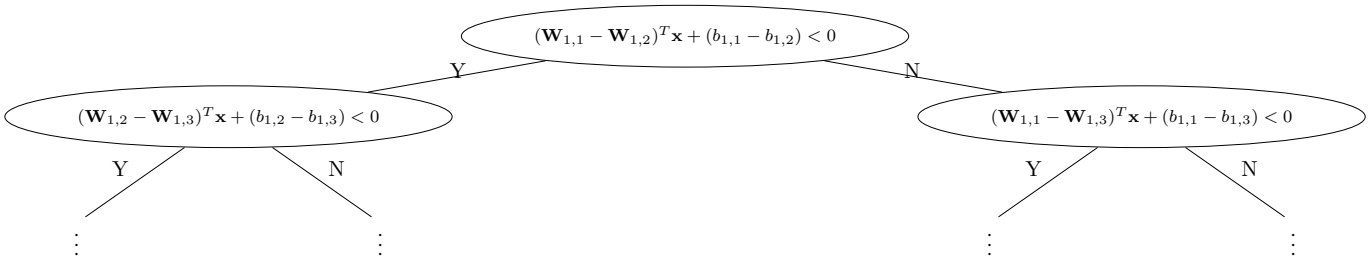

Figure 20: The decision tree $\mathcal{T}_{4,1}$ we are building up to depth 2. The identity of the maximum hyperplane so far is from left to right: 3, 2, 3 and 1.

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 21.

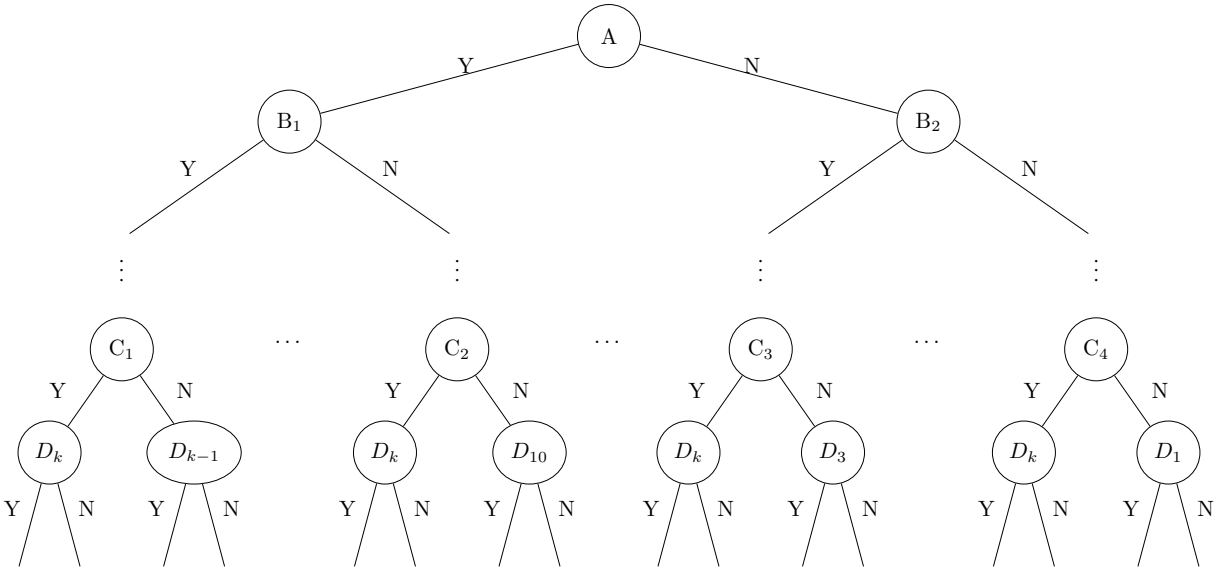

Figure 21: The resulting tree $\mathcal{T}_{4,1}$. The labels A, $B_1, \ldots, C_4$ and D_{i^*} are as follows:

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

The tree $\mathcal{T}_{4,1}$ 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 22, 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 $\mathbf{P}_1$.

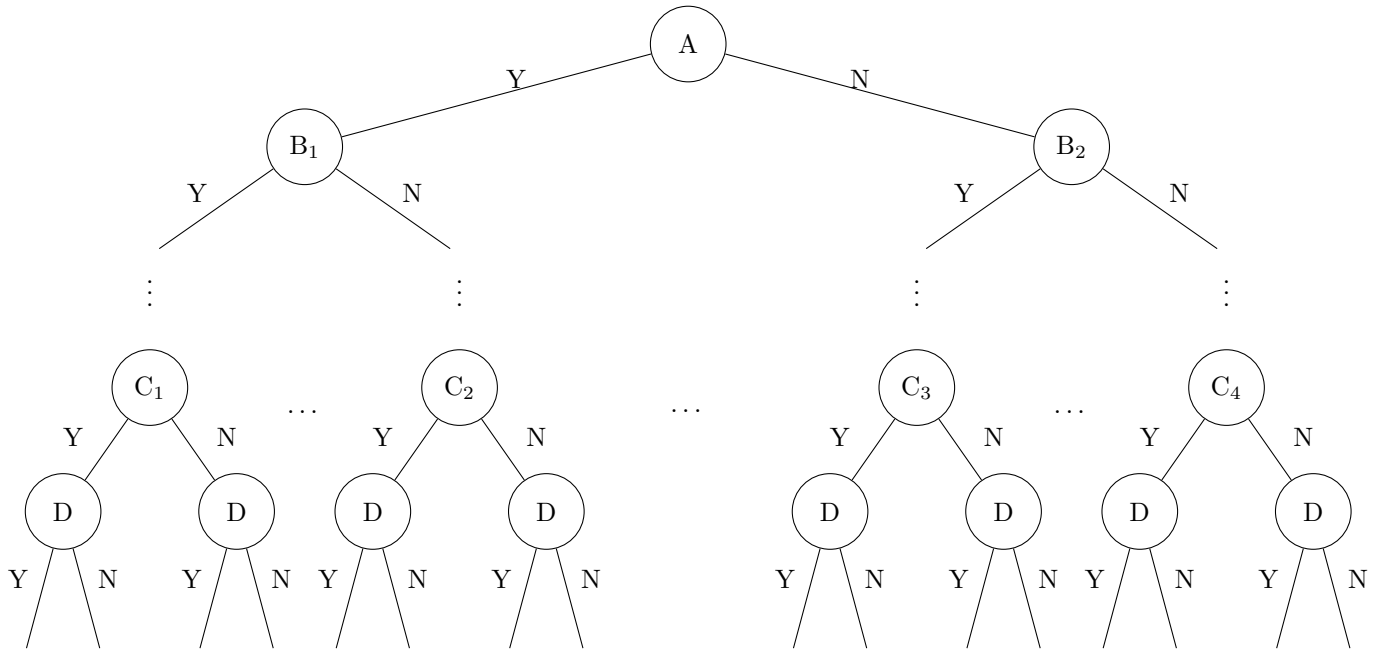

Figure 22: The subtree $\mathcal{T}_{4,2}(\mathbf{P}_1)$ up to depth k_2. The labels A, B_1, ..., D are as follows:

- 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 $\mathbf{W}_{2,i^*}^T \mathbf{P}_1 + b_{2,i^*} < 0$

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. ■

4.4 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 predictions as the neural network. We build the same decision tree as in the proof of Theorem 4 in Section 4.3 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 23.

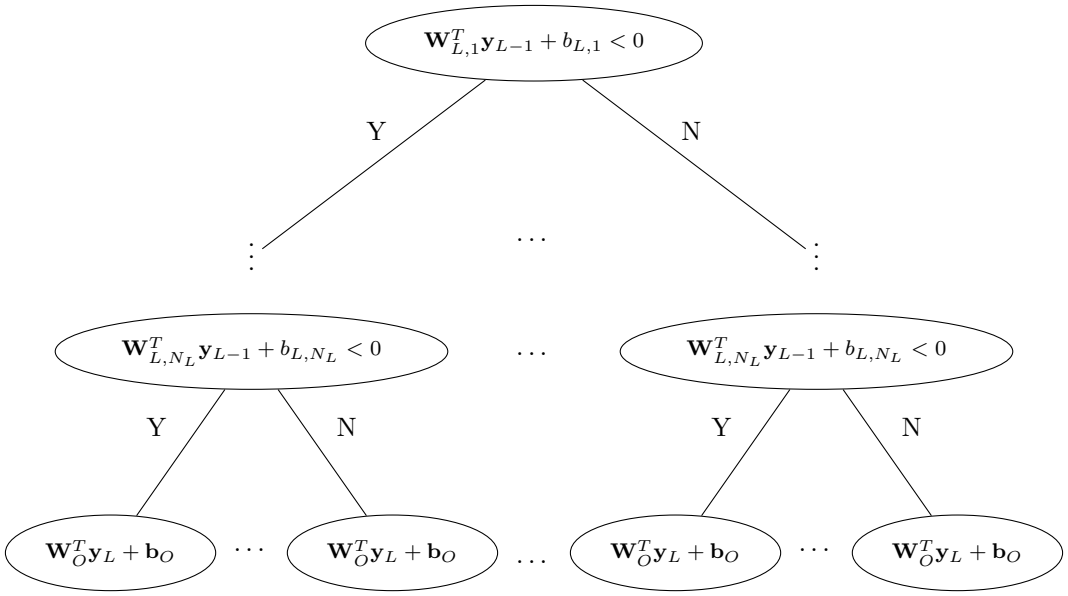

Figure 23: Subtree $\mathcal{T}_{4,L}(\mathbf{y}_{L-1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $\sum_{\ell=1}^{L} N_\ell$. Note that the leaves have linear functions $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$ as outputs.

5. 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.

5.1 Perceptron Classification RNNs and OCT-Hs

The key result in this section is as follows.

Theorem 5 *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 with the following specifications:

- The perceptron activation function as defined in (2).
- Output function $\phi_O(\mathbf{y}_O) : [0,1]^q \to [0,1]^q$.

- 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 $\mathbf{W}_{g,i}, \mathbf{W}_{h,i}, b_{1,i}$.

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

$$\mathbf{x}^* = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{T^*})^T, \tag{9}$$

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

$$(\mathbf{W}_{g,1}, \mathbf{0}, \ldots, \mathbf{0})\mathbf{x}^* + b_{1,1} = \mathbf{W}_{g,1}^T \mathbf{x}_1 + b_{1,1} < 0, \tag{10}$$

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 24.

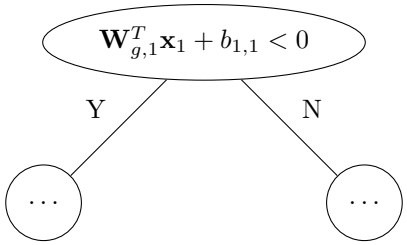

Figure 24: The first split of decision tree $\mathcal{T}_5$.

Independent of whether inequality (10) is satisfied or not, the second split is given by

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

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 25 provides a visualization of the new branches we added to the tree in Figure 24.

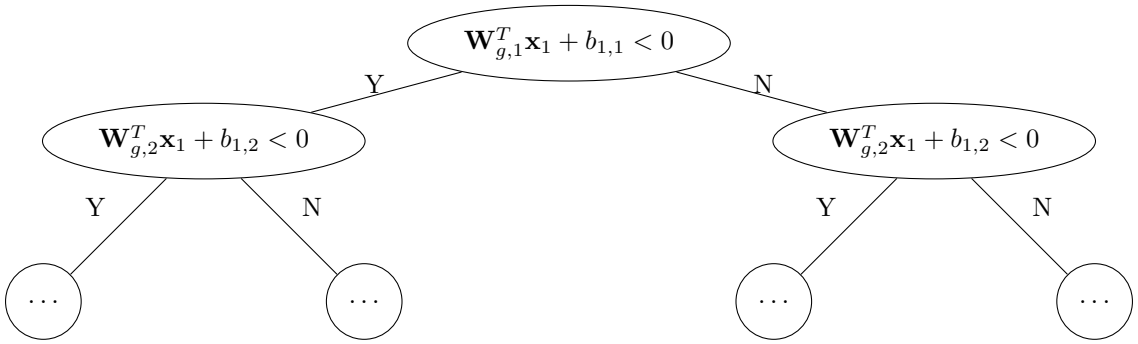

Figure 25: The first two depths of tree $\mathcal{T}_5$.

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 26

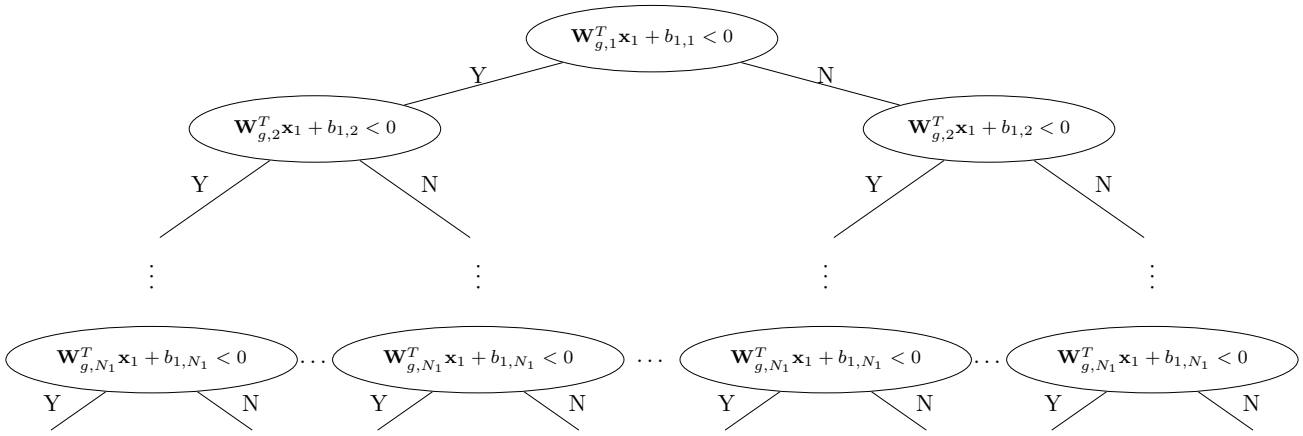

Figure 26: The decision tree $\mathcal{T}_5$ we are building up to depth N_1.

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 26. 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 26. 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 27, 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 27, resulting in the subtree

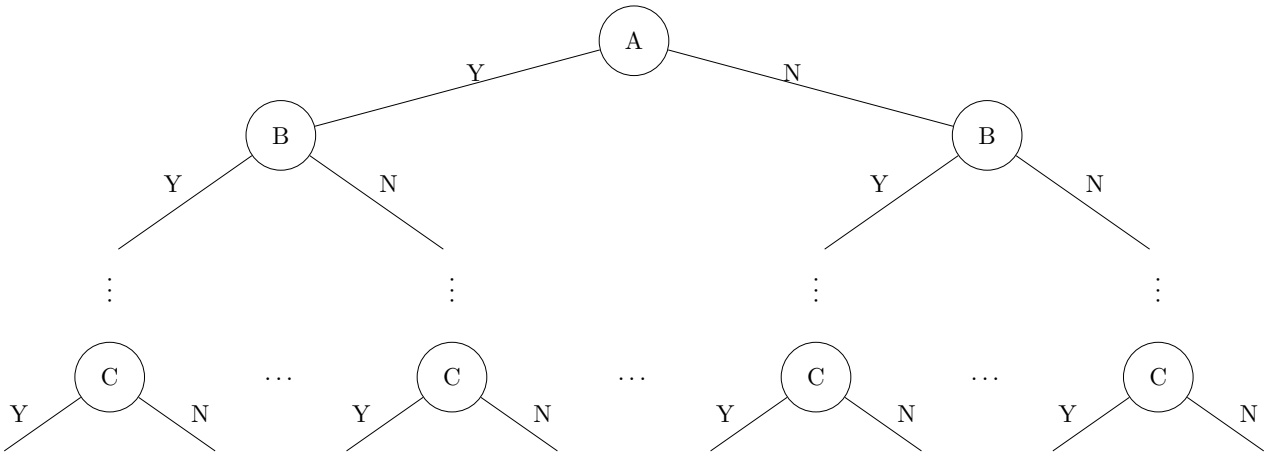

Figure 27: Subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ of depth N_1 is concatenated to the corresponding branch of the subtree depicted in Figure 26, resulting in a subtree of depth $2N_1$.

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

In the subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ in Figure 27 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 28.

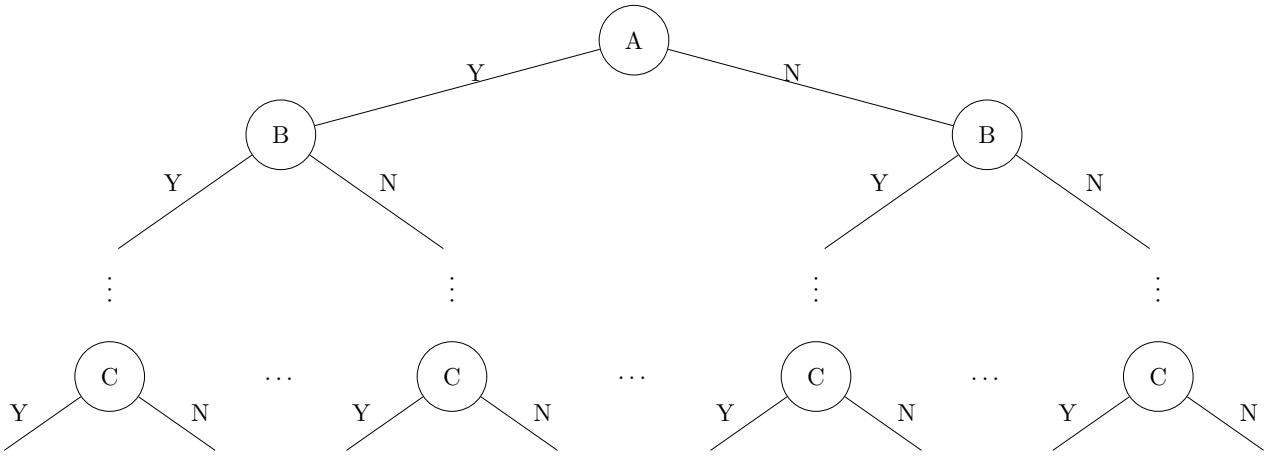

Figure 28: The resulting subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ for $\mathbf{y}_{1,1} = (0, \ldots, 0, 1)^T$. The labels of A, B, C are as follows:

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

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 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. 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 $\mathbf{y}_{T^*,1}^r = \mathbf{y}_{T^*,1}(\mathbf{x}^*)$, and is once again assigned $k(\mathbf{y}_{T^*,1}^r) = k(\mathbf{y}_{T^*,1})$ 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 $((\mathbf{X}_t)_{t=1,\dots,T^*}, \mathbf{o})$, where $T^* = 2$, and is shown in Figure 29. The resultant decision tree is shown in Figure 30.

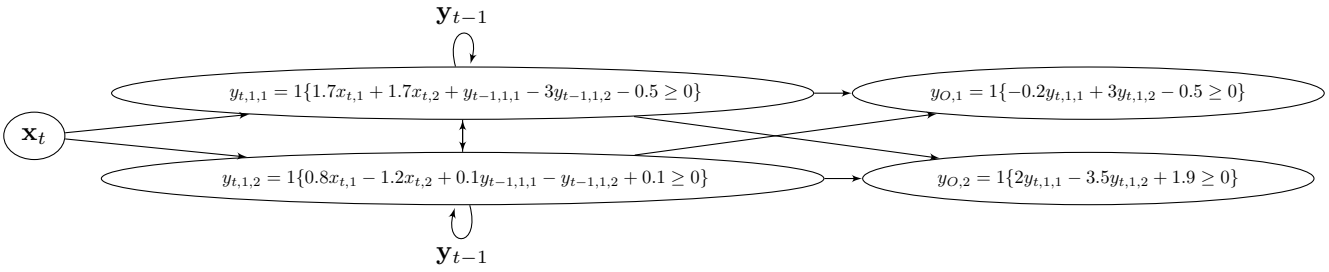

Figure 29: A RNN with the perceptron activation function.

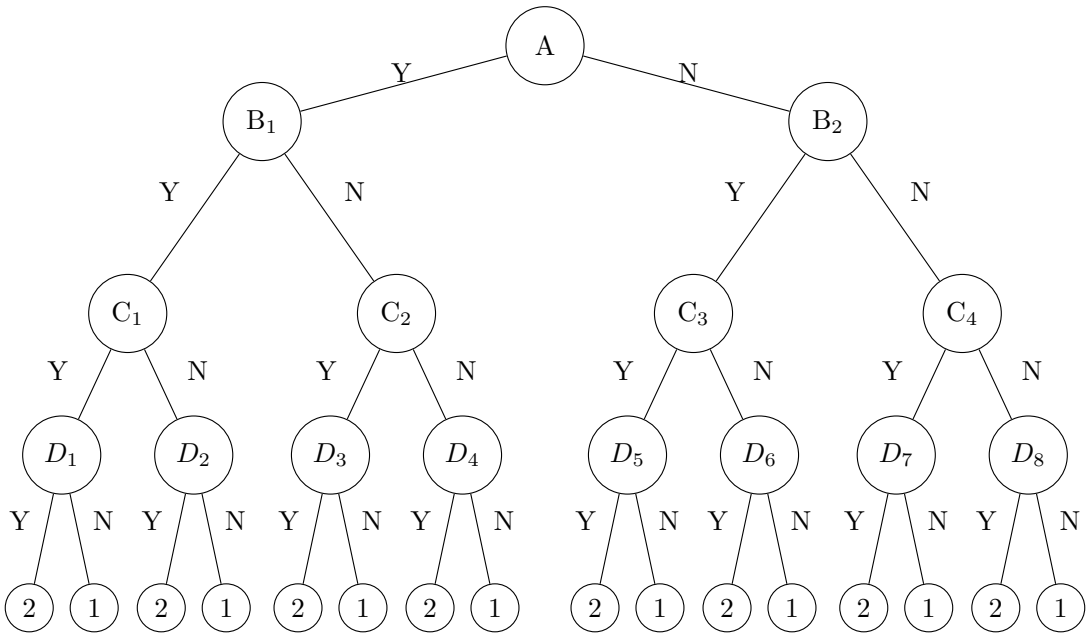

Figure 30: The resulting tree $\mathcal{T}_5$. The labels A, $B_1, \dots, D_8$ are as follows:

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

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

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

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.

5.2 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 5 in Section 5.1 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.

5.3 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 6 *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:

- 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. (9). Then we build the tree up to depth N_1 exactly as we did in the proof in Section 5.1. This part of the tree is shown in Figure 31.

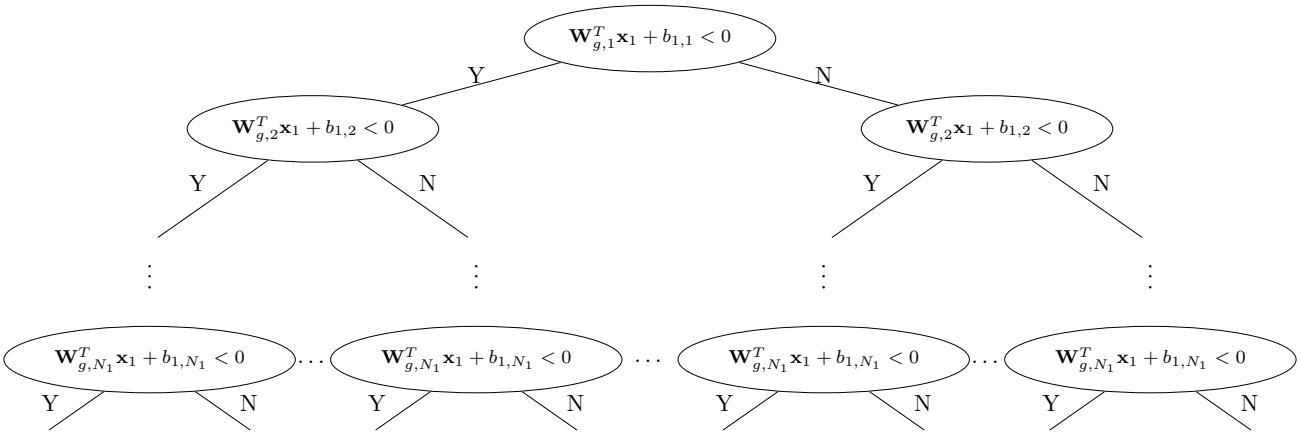

Figure 31: The decision tree $\mathcal{T}_6$ we are building up to depth N_1.

After depth N_1, there are 2^{N_1} branches, as shown in Figure 31. 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 31. 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 27, 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 32.

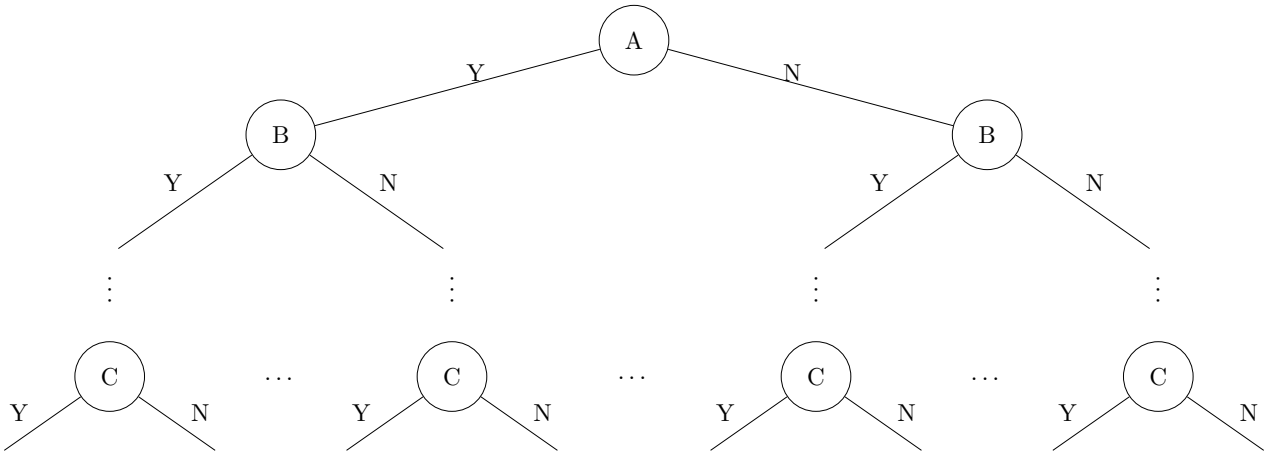

Figure 32: Subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ of depth N_1 is concatenated to the corresponding branch of the subtree depicted in Figure 31, resulting in a subtree of depth $2N_1$.

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

In the subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ in Figure 32 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 33.

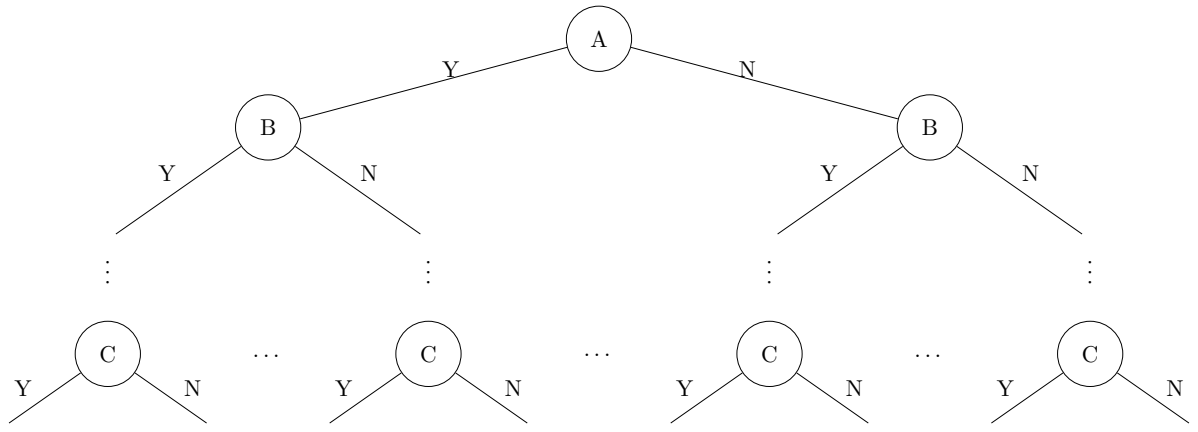

Figure 33: The resulting subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ for $\mathbf{y}_{1,1} = (0, \ldots, 0, \mathbf{W}_{g,N_1}^T\mathbf{x}_1 + b_{1,N_1})^T$. The labels of A, B, C are as follows:

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

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 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 3.2, resulting in the subtree seen in Figure 34.

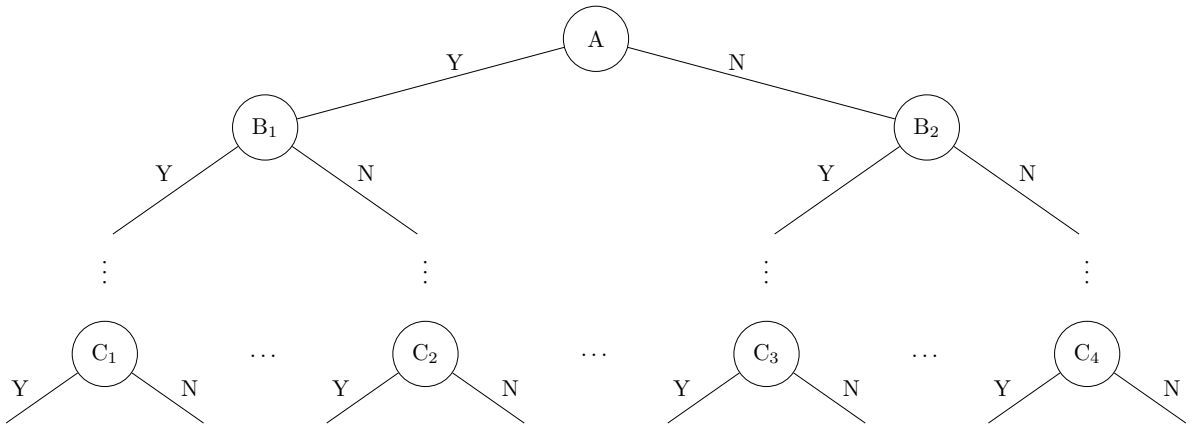

Figure 34: The subtree $\mathcal{T}_{6,O}(\mathbf{y}_1)$. The labels A, B_1, ..., C_4 are as follows:

- 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 $(\mathbf{W}_{O,1} - \mathbf{W}_{O,q})^T\mathbf{y}_{T^*,1} + b_{O,1} - b_{O,q}$

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 $\mathcal{T}_{6,O}(\mathbf{y}_{T^*,1})$ depicted in Figure 34, 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. ■

5.4 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 6 in Section 5.3 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 35.

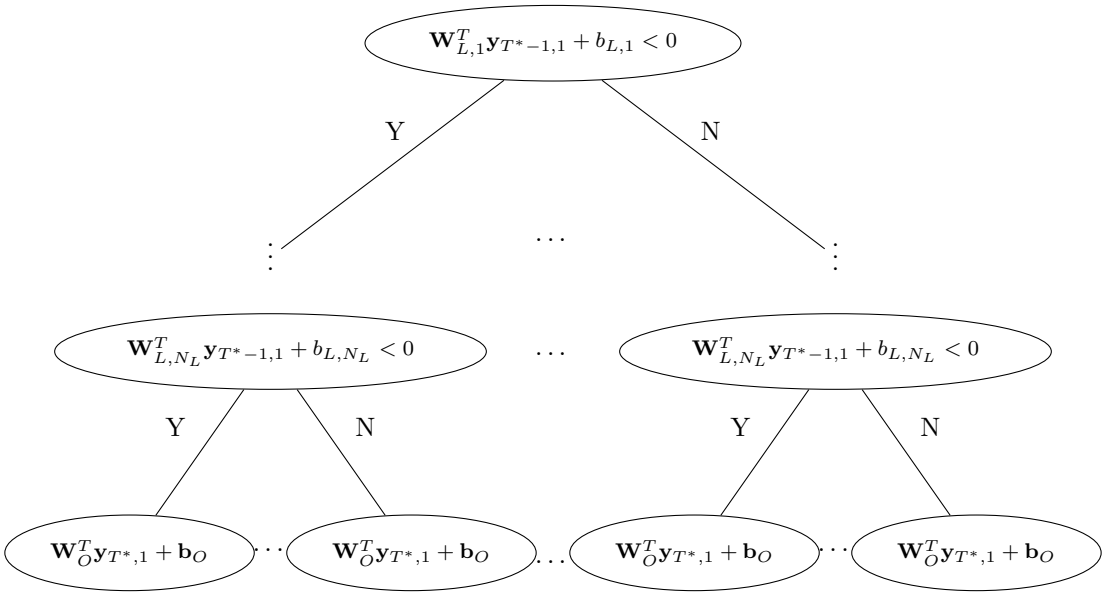

Figure 35: Subtree $\mathcal{T}_{6,L}(\mathbf{y}_{T^*-1,1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $T^* \times N_1$. Note that the leaves have linear functions $\mathbf{W}_O^T \mathbf{y}_{T^*,1} + \mathbf{b}_O$ as outputs.

6. Transforming a Classification Decision Tree into a Classification Neural Network

In this section, we prove that given an OCT-H, there exists a neural network with perceptron activation functions that can classify the given training data at least as well as it. The theorem is as follows.

Theorem 7 *A neural network with perceptron activation functions (2), two hidden layers, N_1 nodes in the first hidden layer, and N_2 nodes in the second can classify training data at least as well as a given classification decision tree with N_1 split nodes and N_2 leaf nodes.*

Proof Our proof is constructive. We are given a decision tree $\mathcal{T}_7$ with the following characteristics:

- Hyperplane splits.
- N_1 split nodes.
- Split node i characterized by split $\mathbf{w}_i^T \mathbf{x} + b_i < 0$, indexed $i = 1, \ldots, N_1$.
- N_2 leaf nodes.
- Leaf node i characterized by classification value k, $k \in \{1, \ldots, q\}$.

We show that we can find a neural network $\mathcal{N}_7$ that classifies data at least as well as the tree with the following characteristics:

- The perceptron activation function (2).
- Two hidden layers and one output layer, indexed $\ell = 1, 2, O$.
- N_ℓ nodes in each hidden 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 define the first hidden layer weights and biases of $\mathcal{N}_7$ as

$$\mathbf{W}_{1,i} \triangleq \mathbf{w}_i \text{ for } i = 1, \ldots, N_1$$

$$b_{1,i} \triangleq b_i \text{ for } i = 1, \ldots, N_1.$$

By these definitions, a node in the first hidden layer activates if and only if an input $\mathbf{x}$ would take the right branch if it was sorted to the node with the same weights in $\mathcal{T}_7$.

To construct the weights for the second hidden layer, first say that $\mathbf{x}$ is assigned to the rth leaf node of $\mathcal{T}_7$, $r = 1, \ldots, N_2$. Define

$$\mathcal{S}_{1,r} = \{i \mid \mathbf{w}_i^T \mathbf{x} + b_i \geq 0\},$$

which is the set of all nodes i of tree $\mathcal{T}_7$ where the inequality $\mathbf{w}_i^T \mathbf{x} + b_i \geq 0$ is satisfied by an input $\mathbf{x}$ sorted to the rth leaf node. Likewise define $\mathcal{S}_{2,r}$ as

$$\mathcal{S}_{2,r} = \{i \mid \mathbf{w}_i^T \mathbf{x} + b_i < 0\}.$$

Then define $N_r = |\mathcal{S}_{1,r}|$, the number of elements in $\mathcal{S}_{1,r}$. For each leaf node r, it either has $N_r \geq 1$, or $N_r = 0$ – call the set of $r \in \{1, \ldots, N_2\}$ such that $N_r \geq 1$

$$C^* = \{r \mid N_r \geq 1\}.$$

We define $\mathbf{W}_{2,r}$ and $\mathbf{b}_{2,r}$ differently in each of the two cases. For the first case, where $r \in C^*$, define

$$(W_{2,r})_i = \begin{cases} \frac{1}{N_r}, & \text{if } i \in \mathcal{S}_{1,r}, \\ -2, & \text{if } i \in \mathcal{S}_{2,r}, \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

$$b_{2,r} = -\frac{N_r - 1}{N_r} - \frac{1}{N_r + 1}, \tag{12}$$

$i = 1, \ldots, N_1$. In the second case, for $i = 1, \ldots, N_1$. and for $r \notin C^*$, we define

$$(W_{2,r})_i = \begin{cases} -1, & \text{if } i \in \mathcal{S}_{2,r}, \\ 0, & \text{otherwise,} \end{cases}$$

$$b_{2,r} = 0.1. \tag{13}$$

This completes the characterization of all the nodes in second hidden layer of $\mathcal{N}_7$.

The N_2 nodes in the second hidden layer of $\mathcal{N}_7$ have a one-to-one correspondence with the leaf nodes of $\mathcal{T}_7$. The weight definitions in equations (11) through (13) are designed to ensure that a node in the second hidden layer activates if and only if the input to $\mathcal{N}_7$ would have been sorted to the corresponding leaf node in $\mathcal{T}_7$. This means that the output of this hidden layer, $\mathbf{y}_2(\mathbf{x})$, is a vector in $[0,1]^{N_2}$ with exactly one entry that is 1, and all others zero.

We prove this correspondence between the second hidden layer nodes of $\mathcal{N}_7$ and the leaf nodes of $\mathcal{T}_7$ for the two cases where $r \in C^*$ and $r \notin C^*$. In the first case, where $r \in C^*$, note that if $\mathbf{x}$ was assigned to leaf node r we must have $y_{1,i} = 1$ for all $i \in \mathcal{S}_{1,r}$. This corresponds to $\mathbf{x}$ taking the right branch at split node i. Otherwise, if $y_{1,i} = 0$ for any $i \in \mathcal{S}_{1,r}$, we know that the input vector would take a left brach at a node in the decision tree where it would need to take a right branch to arrive at leaf node r, and so it cannot have been assigned to r. In that case, the maximum possible value of

$$\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \tag{14}$$

is

$$\frac{N_r - 1}{N_r} - \frac{N_r - 1}{N_r} - \frac{1}{N_r + 1} < 0,$$

based on the weights we defined in Eqs. (11) and (12). This means we correctly have

$$1\left\{\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \geq 0\right\} = 0 \tag{15}$$

in this case.

Likewise, we need a point sorted to leaf node $r \in C^*$ in $\mathcal{T}_7$ to have $y_{1,i} = 0$ for every first hidden layer node $i \in \mathcal{S}_{2,r}$ of $\mathcal{N}_7$, as by definition we must have had $\mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} < 0$ for $\mathbf{x}$ to be assigned to r. This corresponds to $\mathbf{x}$ taking the left branch at split node i. Thus, if we have $y_{1,i} = 1$ for any first hidden layer node $i \in \mathcal{S}_{2,r}$, we know that the point would not have been sorted to leaf node r. In the case where $y_{1,i} = 1$ for any $i \in \mathcal{S}_{2,r}$, the maximum possible value of Eq. (14) is

$$1 - 2 - \frac{N_r - 1}{N_r} - \frac{1}{N_r + 1} < 0$$

based on the weights we have defined in Eqs. (11) and (12), which again correctly has Eq. (15) holding true.

Lastly, if and only if $y_{1,i} = 1$ for all $i \in \mathcal{S}_{1,r}$ and $y_{1,i} = 0$ for every first hidden layer node $i \in \mathcal{S}_{2,r}$, we calculate that

$$\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} = \frac{1}{N_r} - \frac{1}{N_r + 1},$$

meaning that

$$1\left\{\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \geq 0\right\} = 1.$$

This proves that node r in the second hidden layer of $\mathcal{N}_7$ activates if and only if a point is sorted to the corresponding leaf node in $\mathcal{T}_7$ for the case where $r \in C^*$.

If $r \notin C^*$, note that a point is sorted to such a leaf node if and only if $\mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} < 0$ for all $i \in \mathcal{S}_{2,r}$. If $\mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} \geq 0$ for any $i \in \mathcal{S}_{2,r}$, the the maximum possible value of Eq. (14) is

$$-1 + 0.1 < 0 \tag{16}$$

which correctly has

$$1\left\{\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \geq 0\right\} = 0.$$

If and only if $y_{1,i} = 0$ for every one of these nodes, we have

$$1\left\{\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \geq 0\right\} = 1\left\{0.1 \geq 0\right\} = 1.$$

This proves that node r in the second hidden layer of $\mathcal{N}_7$ activates if and only if a point is sorted to the corresponding leaf node in $\mathcal{T}_7$ for the case where $r \notin C^*$. Thus, by construction, we have that an arbitrary node r in $\mathcal{N}_7$ activates given input $\mathbf{x}$ if and only if that input would be sorted to the corresponding leaf node in $\mathcal{T}_7$.

Lastly, we need to define the output weights and biases $\mathbf{W}_{O,i}$ and $b_{O,i}$. In $\mathcal{N}_7$ we will model $\mathcal{T}_7$'s predicted class $k_{predicted}$ for input $\mathbf{x}$ as output vector $\mathbf{y}_O(\mathbf{x})$, where $(\mathbf{y}_O(\mathbf{x}))_k = 1$ for $k = k_{predicted}$ and zero otherwise.

Then define the set of second hidden layer nodes in $\mathcal{N}_7$ where the corresponding leaf node r in $\mathcal{T}_7$ was assigned output value k, $k = 1, \ldots, q$, as

$$\mathcal{R}_k = \{r \in \{1, \ldots, N_2\} \mid \text{Leaf } r \text{ was assigned output } k\}, k = 1, \ldots, q.$$

We then define

$$(W_{O,k})_r = \begin{cases} 1, & \text{if } r \in \mathcal{R}_k, \\ 0, & \text{otherwise,} \end{cases} \tag{17}$$

$$b_{O,k} = -0.1. \tag{18}$$

We next show that the class $\mathcal{N}_7$ predicts for an input data point $\mathbf{x}$ is the same as the class predicted by $\mathcal{T}_7$. To see this, assume $\mathbf{x}$ is input into $\mathcal{T}_7$, is sorted to leaf node r, and is assigned output value $k_{predicted}$. In $\mathcal{N}_7$, by our definitions of $\mathbf{W}_{2,r}$ and $b_{2,r}$, we have $(\mathbf{y}_2(\mathbf{x}))_r = 1$ and is 0 elsewhere. By the above definitions in Eqs. (17) and (18),

$$\mathbf{W}_{O,k_{predicted}}^T \mathbf{y}_2(\mathbf{x}) + b_{O,k_{predicted}} = 0.9 \geq 0,$$

while

$$\mathbf{W}_{O,k}^T \mathbf{y}_2(\mathbf{x}) + b_{O,k} = -0.1 < 0 \text{ for } k \neq k_{predicted}.$$

Thus, using the procedure described in Section 2.1, we find that the network and the tree predict the same classification value for a given data point $\mathbf{x}$, completing the proof. ■

An example of such a transformation is as follows.

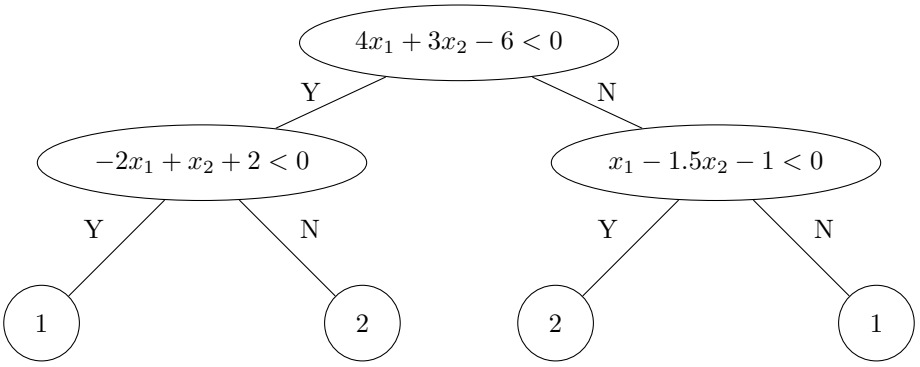

Figure 36: A decision tree that outputs 1 or 2.

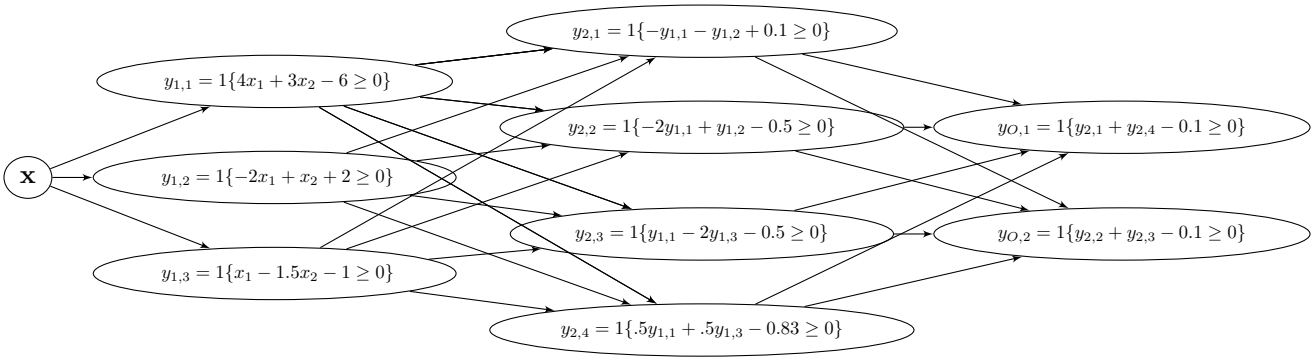

Figure 37: A neural network that does performs the same predictions as the decision tree.

In Figure 36 we have the decision tree we want to transform, and in Figure 37 we have the neural network that performs the same predictions as the tree. As one can see, the nodes in the first hidden layer of Figure 37 correspond to all the splits of the tree in Figure 36, and each node in the second hidden layer of Figure 37 corresponds to a leaf node of the tree in Figure 36. Then, the first output node activates if and only if a data point would have been sorted

to a leaf node assigned value 1, and the second output node activates if and only if a data point would have been sorted to a leaf node assigned value 2. Thus the given neural network performs the same classification task as the decision tree.

With this result, we now have that any decision tree with hyperplane splits can be modeled as a perceptron neural network, alongside the result from Section 3.1 that any perceptron neural network can be modeled as a decision tree with hyperplane splits. Thus, the two must have the exact same modeling ability, indicating that the two techniques are equivalent.

7. Computational Results with Real World Data Sets

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

To train the optimal trees, we used the Optimal Tree software (Bertsimas and Dunn (2018)). 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 – this out-of-sample accuracy is included as the "Accuracy" value in the table. The tree depth is listed in the column labeled "Size Parameters."

To train the neural networks, we used the TensorFlow code. We used sigmoid functions here as the activation functions to make the networks easier to train, as they are a continuous approximation of the perceptron activation functions. We then trained and validated neural networks with sizes based on the tree depths we used, based on the proof in Section 6 – for example, a maximal tree with depth 2 has 3 split nodes and 4 leaf nodes, so for a neural network built with a size based on the tree we would have $N_1 = 3$ and $N_2 = 4$. The N_1 and N_2 values are listed in the column labeled "Size Parameters." We validated the networks using the training parameters

$$\text{Step sizes} \in \{0.001,\ 0.01,\ 0.1,\ 1\}$$

and

$$\text{Regularization coefficients} \in \{1 \times 10^{-6},\ 1 \times 10^{-5},\ 1 \times 10^{-4},\ 0.001,\ 0.01,\ 0.1,\ 1\}.$$

For each network size, we trained 50 neural networks with different random starts using grid search 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. This out-of-sample accuracy is included as the "Accuracy" value in the table.

The seven 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, and the Bank Marketing, Image Segmentation, Letter Recognition, Magic Gamma Telescope, Skin Segmentation, and Thyroid Disease ANN data sets from the UCI Machine Learning Repository (Lichman (2013)). In Table 3, we describe some details of the data sets we used.

Data Set	# Parameters	# Class Values	# Data Points
Bank Marketing	17	2	45,211
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
Skin Segmentation	3	2	245,057
Thyroid Disease ANN	21	3	3772

Table 3: The data sets used and their parameters.

Table 4 reports the results of our experiments.

Model	Data Set	Parameters	Test Results
FNN	Bank	$N_1 = 7,\ N_2 = 8,\ q = 2$	89.6 %
FNN	Bank	$N_1 = 15,\ N_2 = 16,\ q = 2$	89.4 %
FNN	Bank	$N_1 = 63,\ N_2 = 64,\ q = 2$	89.6%
FNN	Bank	$N_1 = 255,\ N_2 = 256,\ q = 2$	89.6%
OCT	Bank	maximum depth = 4, chosen depth 4	89.3%
OCT	Bank	maximum depth = 6, chosen depth 6	89.6%
OCT	Bank	maximum depth = 8, chosen depth 6	89.6 %
OCT-H	Bank	maximum depth = 4, chosen depth 3	89.6 %
OCT-H	Bank	maximum depth = 6, chosen depth 3	89.6 %
OCT-H	Bank	maximum depth = 8, chosen depth 3	89.6 %
FNN	Framingham	$N_1 = 3,\ N_2 = 4,\ q = 2$	82.1%
FNN	Framingham	$N_1 = 15,\ N_2 = 16,\ q = 2$	82.1 %
FNN	Framingham	$N_1 = 63,\ N_2 = 64,\ q = 2$	82.1%
FNN	Framingham	$N_1 = 255,\ N_2 = 256,\ q = 2$	81.7%
OCT	Framingham	maximum depth = 6, chosen depth 6	83.1%
OCT	Framingham	maximum depth = 8, chosen depth 8	82.4%
OCT-H	Framingham	maximum depth = 4, chosen depth 2	83.3%
OCT-H	Framingham	maximum depth = 6, chosen depth 2	83.3%
OCT-H	Framingham	maximum depth = 8, chosen depth 2	83.3%
FNN	Image Segregation	$N_1 = 15,\ N_2 = 16,\ q = 7$	88.4 %
FNN	Image Segregation	$N_1 = 31,\ N_2 = 32,\ q = 7$	83.7 %
FNN	Image Segregation	$N_1 = 63,\ N_2 = 64,\ q = 7$	83.7%
FNN	Image Segregation	$N_1 = 255,\ N_2 = 256,\ q = 7$	83.7 %
OCT	Image Segregation	maximum depth = 4, chosen depth 4	88.4%
OCT	Image Segregation	maximum depth = 6, chosen depth 6	88.4%
OCT	Image Segregation	maximum depth = 8, chosen depth 6	88.4 %
OCT-H	Image Segregation	maximum depth = 4, chosen depth 4	86.0%
OCT-H	Image Segregation	maximum depth = 6, chosen depth 5	86.0%
OCT-H	Image Segregation	maximum depth = 8, chosen depth 5	86.0 %

Model	Data Set	Parameters	Test Results
FNN	Letter Recognition	$N_1 = 15,\ N_2 = 16,\ q = 26$	58.6 %
FNN	Letter Recognition	$N_1 = 63,\ N_2 = 64,\ q = 26$	66.8%
FNN	Letter Recognition	$N_1 = 255,\ N_2 = 256,\ q = 26$	67.0 %
OCT	Letter Recognition	maximum depth = 4, chosen depth 4	37.9 %
OCT	Letter Recognition	maximum depth = 6, chosen depth 6	59.1 %
OCT	Letter Recognition	maximum depth = 8, chosen depth 8	68.2 %
OCT-H	Letter Recognition	maximum depth = 4, chosen depth 4	44.6 %
OCT-H	Letter Recognition	maximum depth = 6, chosen depth 6	72.0 %
OCT-H	Letter Recognition	maximum depth = 8, chosen depth 8	80.3 %
FNN	MGT	$N_1 = 15,\ N_2 = 16,\ q = 2$	87.5 %
FNN	MGT	$N_1 = 31,\ N_2 = 32,\ q = 2$	88.4 %
FNN	MGT	$N_1 = 63,\ N_2 = 64,\ q = 2$	88.1%
FNN	MGT	$N_1 = 255,\ N_2 = 256,\ q = 2$	88.3 %
OCT	MGT	maximum depth = 4, chosen depth 4	84.1 %
OCT	MGT	maximum depth = 6, chosen depth 6	85.3 %
OCT	MGT	maximum depth = 8, chosen depth 8	85.7 %
OCT-H	MGT	maximum depth = 4, chosen depth 4	86.7 %
OCT-H	MGT	maximum depth = 6, chosen depth 5	88.6 %
OCT-H	MGT	maximum depth = 8, chosen depth 5	87.0 %
FNN	Skin Segmentation	$N_1 = 15,\ N_2 = 16,\ q = 2$	99.9 %
FNN	Skin Segmentation	$N_1 = 63,\ N_2 = 64,\ q = 2$	99.9%
FNN	Skin Segmentation	$N_1 = 127,\ N_2 = 128,\ q = 2$	99.9 %
FNN	Skin Segmentation	$N_1 = 255,\ N_2 = 256,\ q = 2$	99.9 %
OCT	Skin Segmentation	maximum depth = 4, chosen depth 4	98.9%
OCT	Skin Segmentation	maximum depth = 6, chosen depth 6	99.8 %
OCT	Skin Segmentation	maximum depth = 8, chosen depth 8	99.9 %
OCT-H	Skin Segmentation	maximum depth = 4, chosen depth 4	99.9 %
OCT-H	Skin Segmentation	maximum depth = 6, chosen depth 6	99.9 %
OCT-H	Skin Segmentation	maximum depth = 8, chosen depth 7	99.9 %
FNN	Thyroid	$N_1 = 7,\ N_2 = 8,\ q = 3$	97.7%
FNN	Thyroid	$N_1 = 15,\ N_2 = 16,\ q = 3$	98.1 %
FNN	Thyroid	$N_1 = 63,\ N_2 = 64,\ q = 3$	97.4%
FNN	Thyroid	$N_1 = 255,\ N_2 = 256,\ q = 3$	98.0%
OCT	Thyroid	maximum depth = 4, chosen depth 4	99.7 %
OCT	Thyroid	maximum depth = 6, chosen depth 4	99.7 %
OCT	Thyroid	maximum depth = 8, chosen depth 4	99.7 %
OCT-H	Thyroid	maximum depth = 4, chosen depth 3	99.9 %
OCT-H	Thyroid	maximum depth = 6, chosen depth 3	99.9%
OCT-H	Thyroid	maximum depth = 8, chosen depth 3	99.9 %

Table 4: Accuracy of FNNs, OCT-Hs and OCTs.

The results form Table 4 lead to the following conclusions:

1. In six out of the seven datasets (the exception is Letter Recognition) the FNN and the OCT-H have very similar accuracy. Moreover, in these data sets OCT and OCT-H also have very similar accuracy.
2. The accuracy of the FNN was relatively insensitive to the size of the network and similarly the accuracy of the OCT-H was relatively insensitive to the depth of the OCT-H.
3. In Letter Recognition OCT-H has a performance edge both with respect to FNN and OCT.

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 provide 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.

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 seven 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.

Acknowledgments

We would like to thank Jack Dunn for for his constructive comments that improved the paper.

References

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

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

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

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

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

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

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

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Google. Google search, 2018. URL https://scholar.google.com/scholar?q=neural+network+articles&btnG=&hl=en&as_sdt=0%2C22. Online; accessed 2018-01-11.

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. URL http://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks.pdf.

A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

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.

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

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi: http://dx.doi.org/10.1016/0893-6080(89)90020-8.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

A. Kurenkov. A 'brief' history of neural nets and deep learning, part 1, 2015. URL http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/. Online; accessed 2017-09-10.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: http://dx.doi.org/10.1038/nature14539.

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.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

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. doi: https://link.springer.com/content/pdf/10.1007\%2FBF02478259.pdf.

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

S. Raschka. Single-layer neural networks and gradient descent, 2015. URL http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html. Online; accessed 2017-09-9.

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.

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

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. URL `http://dl.acm.org/citation.cfm?id=65669.104451`.

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. URL `http://arxiv.org/abs/1409.0575`.

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

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.

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.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

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

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

Z. Zhang, P. Luo, C. C. Loy, and X. Tang. *Facial Landmark Detection by Deep Multi-task Learning*. Springer International Publishing, 2014. URL `https://doi.org/10.1007/978-3-319-10599-4_7`.