15.095 Machine Learning Under a Modern Optimization Lens

Lecture 12: Neural Networks and Trees

Table of Contents

Table of Contents

What Are Neural Networks?

- Neural networks, a supervised learning technique, have become one of the most widely used machine learning techniques today
- Increased computational power, advances in optimization (stochastic gradient methods), and the massive availability of data sets have made it possible to train large neural networks with many hidden layers
- This methodology, in particular, is known as deep learning
- It has had great successes in the fields of image recognition, natural language processing, and speech recognition

A Sample Neural Network

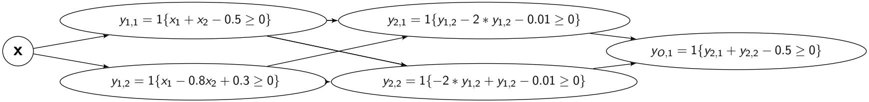


Figure 1: An example of a neural network.

Challenges with Neural Networks

- They rely on heuristics in their training process, like dropout and early stopping
- 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

Optimal Decision Trees

- However, there has recently been significant progress in finding trees that are near optimal, as discussed in the paper *Optimal Classification Trees*, by Bertsimas and Dunn (2017)
- Using mixed integer optimization and local search methods the authors find optimal classification trees (OCT) that significantly improve upon CART
- Furthermore, their approach allows one to consider hyperplane splits, leading to optimal classification trees with hyperplanes (OCT-Hs), which generalize support vector machines.
- OCT-Hs are less interpretable than trees whose splits rely on only one variable (OCTs), but are still more interpretable than neural networks

Sample Tree with Hyperplane Splits

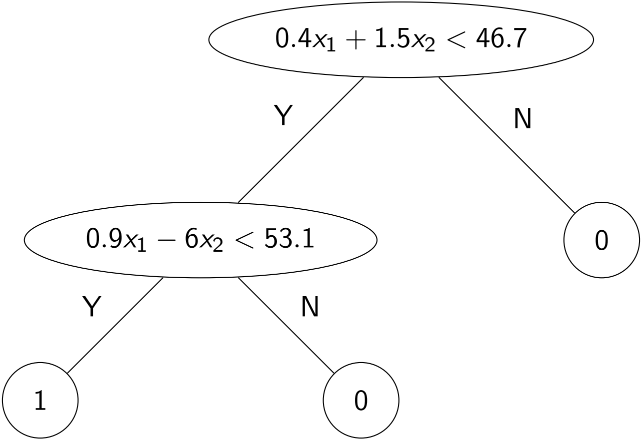


Figure 2: A sample decision tree with hyperplane splits.

Goals

- We investigate the modeling power of neural networks in comparison with OCT-Hs
- We prove that a set of neural networks can be transformed to classification trees with hyperplane splits with the same accuracy in the training set, showing that OCT-Hs are at least as powerful as neural networks
- Conversely, a given classification tree with hyperplane splits 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

Implications

- These results link two of the most popular and widely utilized machine learning methods, shedding new light on their strengths and weaknesses
- 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
- 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

Table of Contents

Neural Network Structure Overview

- 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
- Each node $n_{\ell,i}$ in the neural network has associated weight vector $\mathbf{W}_{\ell,i}$ and bias scalar $b_{\ell,i}$
- After deciding on the values for the architecture parameters, we solve for the weight vectors and bias scalars using stochastic gradient descent
- An example of a neural network can be seen in Figure 3

Sample FNN

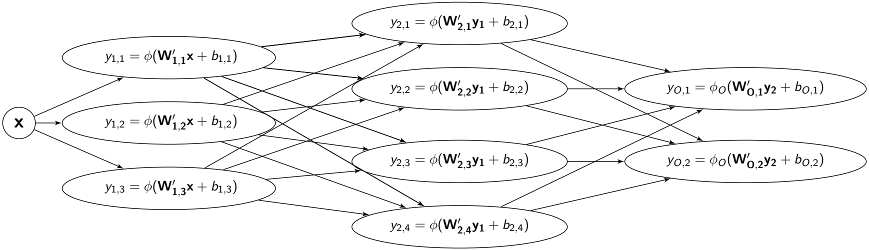


Figure 3: An example of a classification feedforward neural network. It has 2 hidden layers, with 3 nodes in the first hidden layer, 4 nodes in the second, and 2 nodes in the output layer. Each node in this network has its own unique weights $\mathbf{W}_{\ell,i}$ and $b_{\ell,i}$. Also, the nodes in one layer have directed edges leading to all the nodes in the next layer (a trait known as being "fully connected").

Feedforward Neural Network Characteristics (1)

- Node $n_{\ell,i}$ in hidden layer ℓ calculates

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

- Here $\phi(x)$ 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
- Common choices are for $\phi(x)$ are
  1. $1\{x \geq 0\}$, the perceptron function
  2. $\max(x, 0)$, the ReLU function

Feedforward Neural Network Characteristics (2)

- Node $n_{O,i}$ in the output layer calculates

$$y_{O,i} = \phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_L + b_{O,i}) \tag{2}$$

- The only difference between $\phi(x)$ and $\phi_O(x)$ is that $\phi_O(x)$ does not have to be non-linear
- If the perceptron is chosen as the activation function, then typically

$$\phi_O(\mathbf{x}) = 1\{\mathbf{x} \geq 0\}$$

- If the ReLU function is chosen as the activation function, then $\phi_O(\mathbf{x})$ can be defined as

$$(\phi_O(\mathbf{x}))_i = \begin{cases} 1, & \text{where } i = \text{argmax}_{i=1,\ldots,N_0}(x_i), \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Feedforward Neural Network Characteristics (3)

- 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 if there is a tie, the smallest index k is our choice
- We then use k as our predicted class value

Tree Parameters (1)

- A tree has depth N_1 if N_1 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 weight vector and bias scalar $\mathbf{w}_i, b_i$

Tree Parameters (2)

- 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 $k \in \{1, \dots, q\}$ 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, $r \in \{\lfloor T/2 \rfloor + 1, \dots, T\}$ and $k_r \in \{1, \dots, q\}$, the network outputs k_r as the classification value for $\mathbf{x}$

Example Tree

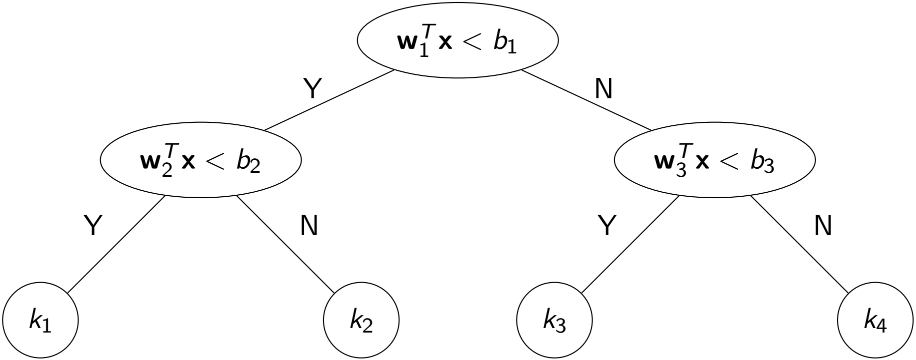


Figure 4: An OCT-H of depth 2. Data in the four leaf nodes are classified as k_1, k_2, k_3, and k_4.

Table of Contents

Transforming a FNN with the Perceptron Activation Function into a DT

- Given a FNN $\mathcal{N}_1$ with the perceptron activation function, we are able to construct a decision tree $\mathcal{T}_1$ with hyperplane splits and 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
- We can assign these values to the 2^{N_1} leaf nodes of $\mathcal{T}_1$
- Since we know that an OCT-H must classify training data at least as well as $\mathcal{T}_1$, we know that an OCT-H must do at least as well as $\mathcal{N}_1$ too
- This leads to the following theorem

FNN with Perceptron to DT Theorem

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 and N_1 nodes in the first hidden layer.

regression?
using finiteness?

Constructing $\mathcal{T}_1$ (FNN Perceptron)

- We are given a feedforward neural network $\mathcal{N}_1$ with the following characteristics:
  - The perceptron activation function, defined as $\phi(x) = 1\{x \geq 0\}$
  - 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 $i = 1, \ldots, N_\ell$
  - Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$
- 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

$$\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1} < 0 \tag{4}$$

- This results in the simple split seen in Figure 5

The First Split of $\mathcal{T}_1$

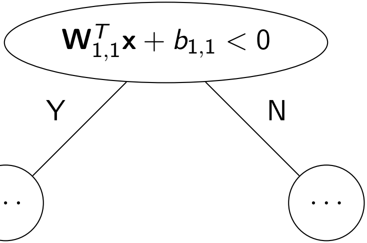


Figure 5: The first split of decision tree $\mathcal{T}_1$.

Continuing building $\mathcal{T}_1$

- Independent of whether inequality (4) is satisfied or not, the second split is given by
$$\mathbf{W}_{1,2}^T\mathbf{x} + b_{1,2} < 0 \tag{5}$$
- 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
$$\mathbf{W}_{1,N_1}^T\mathbf{x} + b_{1,N_1} < 0 \tag{6}$$
- This results in the subtree seen in Figure 6

Completed Branches of $\mathcal{T}_1$

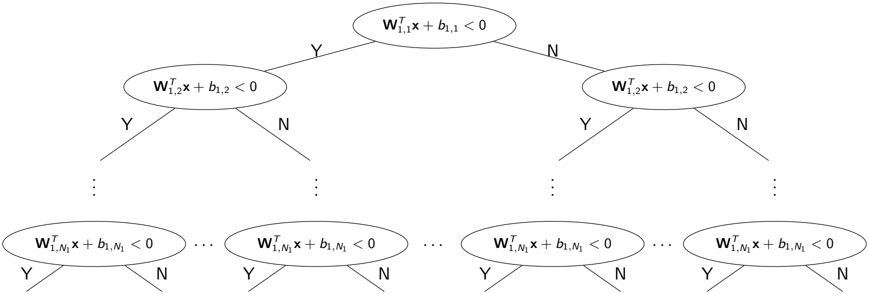


Figure 6: The decision tree $\mathcal{T}_1$. We still need to define the classification values assigned to the leaf nodes.

Defining the Leaf Nodes of $\mathcal{T}_1$

- To complete the construction of $\mathcal{T}_1$, we need to assign a classification value to every leaf node
- 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 $\mathbf{W}_{\ell,i}$, $b_{\ell,i}$, and $\mathbf{y}_1^r$ (the output of the first hidden layer associated with leaf node r) one can deterministically calculate the the final prediction of $\mathcal{N}_1$, $k(\mathbf{y}_1^r)$, by using the process outlined in the section Overview of Neural Networks
- In every node r of the tree we assign the classification value $k(\mathbf{y}_1^r)$ associated with the corresponding first hidden layer output

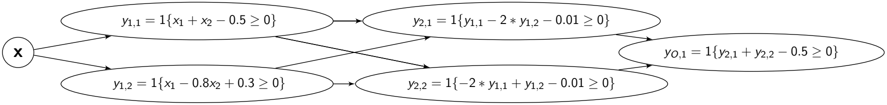
Proving that $\mathcal{T}_1$ and $\mathcal{N}_1$ make the same predictions

- To see that the output of $\mathcal{T}_1$ is the same as the output of $\mathcal{N}_1$, note that if $\mathbf{x}$ is input into $\mathcal{N}_1$, the first hidden layer outputs $\mathbf{y}_1(\mathbf{x})$
- This results in the final network output $k(\mathbf{y}_1)$
- However, in the decision tree, $\mathbf{x}$ is assigned to the leaf node corresponding to $\mathbf{y}_1^r = \mathbf{y}_1(\mathbf{x})$
- There, it 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, completing the proof of the theorem

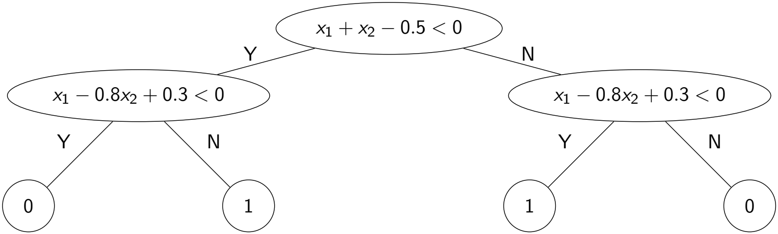
Notes on Theorem 1

- The construction of tree $\mathcal{T}_1$ is independent of L, the number of hidden layers
- 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 $\phi_O(\mathbf{x})$

Example: An NN and the Equivalent Tree



(a)



(b)

Figure 7: (a) An FNN with the perceptron activation function performing an XOR operation. (b) The corresponding decision tree.

Transforming a FNN with the ReLU Activation Function into a DT

- Next, we extend Theorem 1 to the case of FNNs with the rectified linear unit activation function (where $\phi(x) = \max(x, 0)$)
- Other than the new activation function, the only other difference in network architecture from the perceptron case is that we assume we use the output function defined in Equation (3)
- We can extend the theorem by constructing a decision tree with maximum depth $q - 1 + \sum_{i=1}^{L} N_\ell$ that makes the same predictions as the given neural network $\mathcal{N}_2$

FNN with ReLU to DT Theorem

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, L hidden layers, N_ℓ nodes in layer $\ell = 1, \dots, L$, and q nodes in the output layer.

The First Sub-Tree of $\mathcal{T}_2$

- The first subtree is built exactly as it was in the case of an FNN with the perceptron activation function, and can be seen in Figure 8

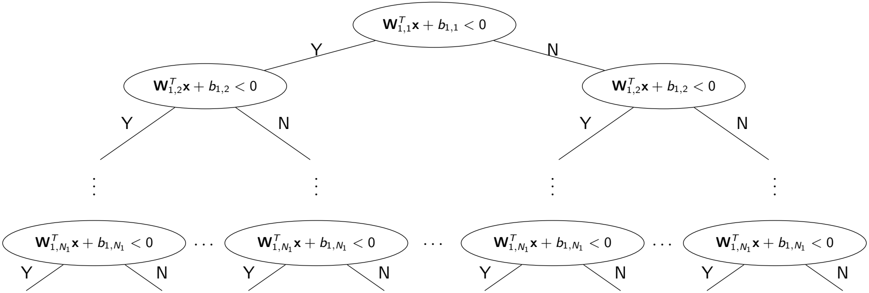


Figure 8: The decision tree $\mathcal{T}_2$ we are building up to depth N_1.

Constructing the Second Sub-Tree of $\mathcal{T}_2$

- After depth N_1, there are 2^{N_1} branches
- These branches correspond to the 2^{N_1} possible output vectors $\mathbf{y}_1$ of the first layer of $\mathcal{N}_2$,

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

- We model the second layer of $\mathcal{N}_2$ by constructing after each branch of the first subtree a new subtree of depth N_2 as in Figure 8, but with the corresponding value of $\mathbf{y}_1$ playing the role of $\mathbf{x}$.

The Second Sub-Tree of $\mathcal{T}_2$

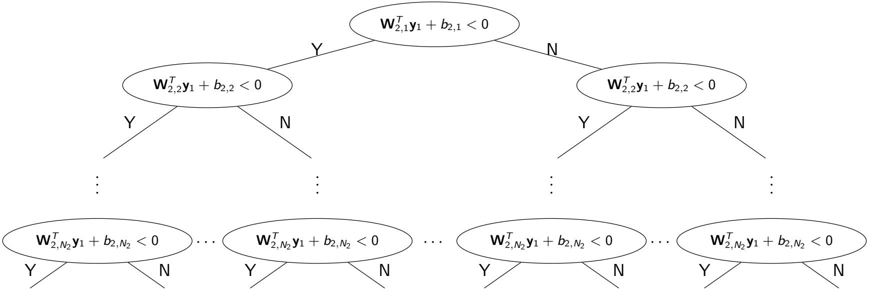


Figure 9: Subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ of depth N_2 is concatenated to the corresponding branch of the subtree depicted in the previous slide, resulting in a subtree of depth $N_1 + N_2$.

The Second Sub-Tree of $\mathcal{T}_2$ Example

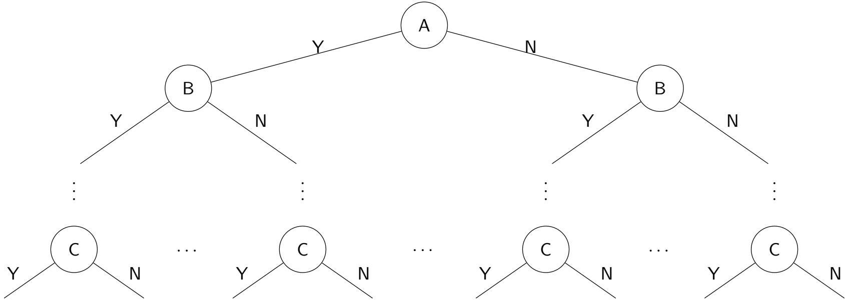


Figure 10: The resulting subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ for $\mathbf{y}_1 = (0, \ldots, \mathbf{W}_{1,N_1}^T\mathbf{x} + b_{1,N_1})$. A, B, C are as follows:

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

Building the Final Sub-Tree of $\mathcal{T}_2$

- This process continues for the remaining L hidden layers of the network
- We then need to model the output layer, which calculates

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

- We simulate this calculation by building a new subtree $\mathcal{T}_{2,O}(\mathbf{y}_L)$, where at each branch we perform some pairwise comparison of the entries of the vector

$$\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$$

First branches of the Final Sub-Tree of $\mathcal{T}_2$

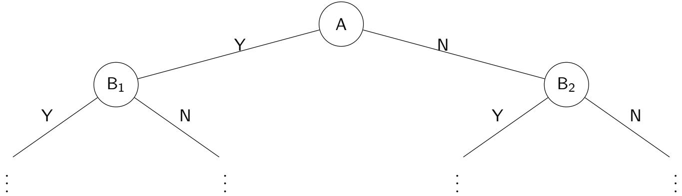


Figure 11: A portion of the subtree $\mathcal{T}_{2,O}(\mathbf{y}_L)$. The labels A, B_1, B_2 are as follows:

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

Completing $\mathcal{T}_2$

- Each branch of the tree 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 linear 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 $\mathbf{x}$
- Once we append it to $\mathcal{T}_2$, by construction we have a decision tree that makes the same predictions as $\mathcal{N}_2$
- Since an OCT-H does at least as well as $\mathcal{T}_2$ in classifying the training data, it must do at least as well as $\mathcal{N}_2$ too, completing the proof of the theorem

Table of Contents

Transforming a Classification DT into a Classification FNN

- We have shown that it is possible to take a neural network and find an equivalent decision tree
- Here we show that the converse is also possible, specifically that one can take a decision tree and find an equivalent FNN with the perceptron activation function

Classification DT to Classification FNN Theorem

Theorem 3

A neural network with perceptron activation functions, 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.

Example Decision Tree

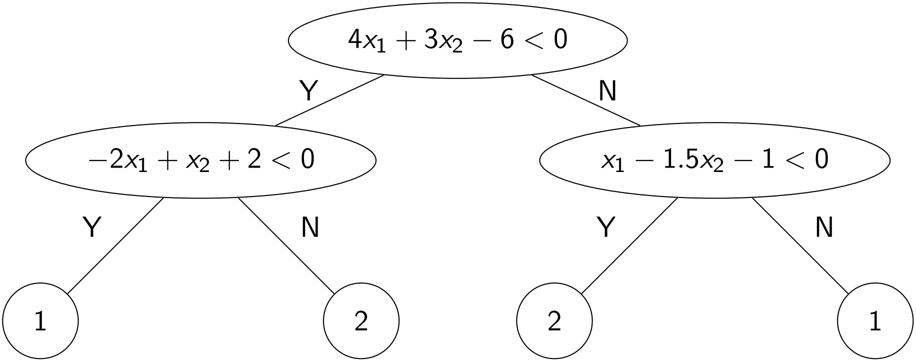


Figure 12: A decision tree that outputs 1 or 2.

The Corresponding Neural Network

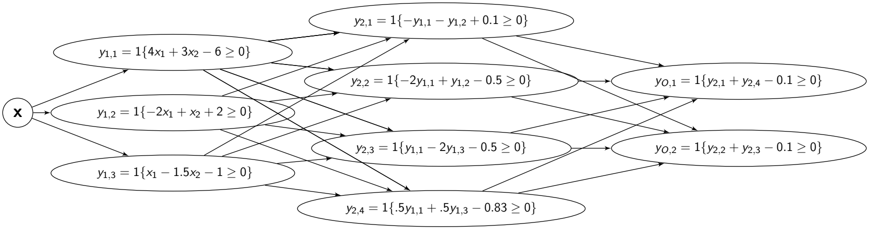


Figure 13: A neural network that performs the same predictions as the decision tree.

Table of Contents

The Data

- We examined the performance of FNNs and OCT-Hs on seven datasets
- We trained several different formulations of each model, and compared their out-of-sample accuracies
- The Data Sets are

Dataset	# 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 1: The datasets used and their parameters.

Computational Result Models – Trees

- To train the optimal trees, we used the Optimal Tree software from Jack Dunn
- For each data set, we trained and cross validated OCTs and OCT-Hs of varying depths, 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"

Computational Result Models – Neural Networks

- To train the neural networks, we used 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 the Transforming a Decision Tree into a Neural Network section
- 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 $N_2 = 4$

Computational Result Models – Neural Network Training Parameters

- 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

$$\text{Regularization coefficients} \in \{1 \times 10^{-6},\ 1 \times 10^{-5}, \ldots, 0.1,\ 1\}$$

Computational Result Models – Neural Network Optimization

- 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

Computational Results – Abridged

Model	Dataset	Parameters	Test Results
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	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 %

Computational Result Conclusions

- 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 datasets OCT and OCT-H also have very similar accuracy
- 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.
- In Letter Recognition OCT-H has a performance edge both with respect to FNN and OCT

Computational Result Implications

- 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 applications
- The fact that OCTs give similar results to FNNs is particularly noteworthy as OCTs are very interpretable

Table of Contents

Concluding Remarks

- We have shown that optimal decision trees are at least as powerful as neural networks in terms of modeling power
- 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 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

Table of Contents

Other Computational Results (1)

Model	Dataset	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%

Other Computational Results (2)

Model	Dataset	Parameters	Test Results
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 %
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 %

Other Computational Results (3)

Model	Dataset	Parameters	Test Results
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 %

Bibliography

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

______, *Machine learning under a modern optimization lens*, Dynamic Ideas, 2018.

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`.

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

A. Graves and J. Schmidhuber, *Offline handwriting recognition with multidimensional recurrent neural networks*, Advances in Neural Information Processing Systems 21 (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), Curran Associates, Inc., 2009, pp. 545–552.

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

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

M. Lichman, *UCI machine learning repository*, 2013.

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** (2014), no. 1, 335–366.

Tensorflow.