Interpreting Neural Networks with Decision Trees

By VIKAS GUPTA

WORD COUNT	6999	TIME SUBMITTED	25-JUL-2022 05:52PM
		PAPER ID	88791571

Interpreting Neural Networks with Decision Trees

Vikas Gupta

Interpreting Neural Networks with Decision Trees

Dissertation submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Computer Science

by

Vikas Gupta
[Roll No: CS2034]

under the guidance of

Debrup Chakraborty
Associate Professor
Cryptology and Security Research Unit

Ansuman Banerjee
Professor
Advanced Computing and Microelectronics Unit

Indian Statistical Institute
Kolkata-700108, India

July 2022

To my family and my supervisors

CERTIFICATE

This is to certify that the dissertation titled **"Interpreting Neural Networks with Decision Trees"** submitted by **Vikas Gupta** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under our supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Debrup Chakraborty.

Debrup Chakraborty
Associate Professor,
Cryptology and Security Research Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Ansuman Banerjee

Ansuman Banerjee
Professor,
Advanced Computing and Microelectronics Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgement

I would like to convey my highest gratitude to my supervisors, *Dr. Ansuman Banerjee*, Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, and *Dr. Debrup Chakraborty*, Cryptology and Security Research Unit, Indian Statistical Institute, Kolkata for their guidance and continuous support and encouragement. Before pursuing this thesis, I didn't have much knowledge about the research domain or any experience of good research. They aided me to hone my reasoning skills and illuminated the ways to improve my research and writing skills. Their guidance made me, what people informally call, "from zero to hero."

My deepest thanks to the faculties of Indian Statistical Institute, for their support.

Vikas Gupta
Indian Statistical Institute
Kolkata - 700108, India.

Abstract

Explainability in Artificial Intelligence is being looked at as one of the major challenges to solve before a more widespread adoption of AI can be enabled. Explainability is being mandated as the need to demonstrate safety to users and gain their trust in the rationale of automated decision making. Neural Networks, in particular, pose a classical challenge to explainability and interpretability, because of their structure and the opacity in the rationale behind their decision making [4]. In this thesis, we make an attempt to interpret the rationale behind the working of a neural network using another Machine Learning artifact, namely decision trees. Decision trees score somewhat better to Neural nets on account of interpretability with their structure, since one can extract the constraints on each tree path from root to leaf and interpret the rationale behind a decision. The internal nodes provide feature labels, which we can use to understand the rationale behind the decision that the tree takes on a given data point. In this work, we propose to approximate the concept represented by a given neural network using a decision tree and show how it performs on benchmark datasets. We implement our proposed approaches on real-world datasets and analyze the performance of our proposed algorithms to demonstrate the efficiency of our proposals. The results show that decision trees generated with our algorithm are more understandable than those generated by the trained network. We believe our work will open up a lot of new research directions and applications of learning based methods in interpretability and robustness verification with decision tree models.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent times, almost every aspect of our society is being transformed with artifacts that are being automatically learnt from data. Machine Learning (ML) and Deep Learning (DL), in particular, has made major inroads in every aspect of the world around us, with their ability of automated reasoning with close to human and often, better precision. Indeed, driven by signficant advances in ML and DL research, we are seeing an unprecedented adoption of learning based techniques in both hardware and software. However, in spite of these advances, these AI-based techniques, and DL, in particular, pose a challenge to scientists and practitioners, because of their opacity in decision making. Neural Networks, in particular, have always posed a challenge, being a black box model, and thereby, lacking interpretability. On one side, this has inspired recent research interest on explainability of ML artifacts, to make them more accountable, more diagnosable and more interpretable in their decisions to enable wider adoption. On the other side, there has been attempts to interpret neural nets with more interpretable ML models, that can approximate the rationale behind the network behavior. This work is an attempt in that direction. In particular, we propose to use decision trees, a popular ML artifact, to interpret neural networks. Decision trees have much more information in their structure that allows one to interpret the rationale behind their decision making. We therefore, attempt to build a decision tree, to approximate the behavior of a neural network, and use the tree to interpret the network rules. We show through experiments the performance of our approach in different scenarios.

In the following section in this chapter, we first provide a background of the relevant concepts used in the thesis, followed by the contribution made by us in the following chapter. We begin with a discussion on our artifact of study, namely neural networks, and then the artifact of interpretation, namely decision trees.

1.1 Background and Relevant Concepts

1.1.1 Neural network

An artificial neural network (NN) is a computational model inspired by biological neural networks, that consist of neurons which process electrical signals and synapses which connect neurons and transmit signals. An NN consists of a set of nodes (modeling neurons) that are organized in layers, and a set of edges (modeling synapses) that connect neurons belonging to successive layers. Figure 1.1 shows a neural network that consists of an input layer (nodes marked $s_{0,-}$;), two hidden layers (nodes marked $s_{1,-}$; and $s_{2,-}$) and an output layer (node marked $s_{3,1}$). The edges are annotated with weights

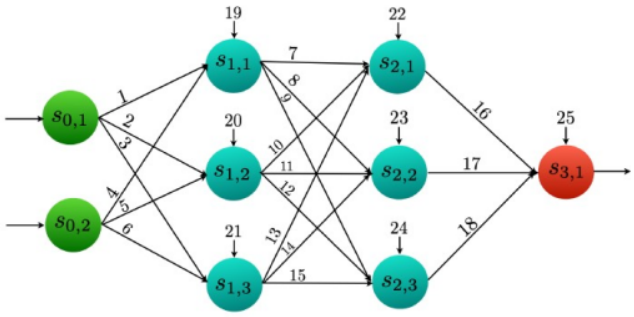

Figure 1.1: An abstract look of an Artificial Neural Network

and the nodes are annotated with biases, which are obtained by a training procedure that consists of a series of adjustments to weights and biases based on a training set (a set of input-output examples).

The computation in a neural network proceeds layer by layer. Given an assignment of values to the neurons in a particular layer, the value of a neuron s in the succeeding layer is computed by taking the sum of the values at the neurons in the proceeding layer weighted by the edge weights connecting them to the neuron s followed by adding the bias associated with s, and finally applying an activation function. For instance, given values $v_{0,1} = 1$ for node $s_{0,1}$ and $v_{0,2} = 2$ for node $s_{0,2}$, the value at node $s_{1,1}$ before applying the activation function is given by $1 \times v_{0,1} + 4 \times v_{0,2} + 19 = 28$, where 1 and 4 are the weights on the incoming edges from $s_{0,1}$ and $s_{0,2}$ into $s_{1,1}$ and 19 is the bias associated with the node $s_{1,1}$. Let v_i denote the vector of values associated with the nodes at layer i. Then v_i is computed from v_{i-1} using the weights and biases, and is often followed by an activation function, that represents the action potential of the neuron. For this example, we focus on NNs with a specific kind of activation function, called a Rectified Linear Unit (ReLU). When the ReLU function is applied to a node with a positive value, it returns the value unchanged (the active case), but when the value is negative, the ReLU function returns 0 (the inactive case). Formally, the ReLU operator σ is defined as $\sigma(\mathrm{x}) = \max(\mathrm{x}, 0)$, that is, a positive value is propagated, whereas a negative value is transformed into 0. Hence, we can represent v_i as $v_i = \sigma(W_i \times v_{i-1} + b_i)$, where W_i is a matrix capturing the weights on the edges between layers $i-1$ and i, and b_i is a vector of biases associated with nodes in layer i. Apart from RELU, other activation functions that are popularly used include:

- Absolute function

$$y(x) = |x|$$

- Sigmoid function

$$y(x) = \frac{1}{1 + e^x}$$

- Tanh function

$$y(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Neural networks have many applications including but not limited to classification, pattern recognition, prediction of values as well as decision modelling. The study of these neural networks, it's

variants and applications is one of the leading areas of research in the field of data science. With increasing power of processing systems and reducing costs of memory, the popularity of deep neural networks is increasing by a great amount. With increase in the number and diversity of application areas of neural networks, a number of variations of the fundamental neural network architecture has been proposed in literature. Some of these variations are more powerful than their simple feedforward counterparts and have the ability to recognize patterns in more complex data. Some of the popular neural network variants are mentioned in the following.

- *Convolutional Neural Networks*: These deep neural networks have a special convolutional layer along with the regular layers. These convolutional layers have specific filters that are used to filter out specific sections of the input data. This property of being able to extract specific parts of the input data for processing makes convolutional neural networks great for image processing and pattern recognition.
- Recurrent Neural Networks: These deep neural networks unlike other networks have recurrent layers, which are special layers which have synapses connecting to the next layer as well as itself. This structure allows the network to essentially remember the previous output (by propagating the output of the layer as input to the same layer). This property gives recurrent neural networks the ability to recognize sequential patterns which is often used for natural language processing problems.
- Transformers: These are special deep neural networks that are often a sequence of encoder layers and decoder layers. These layers encode the input into a sequence of numbers and decode them back to reconstruct the output. This structure allows the transformer to work well for problems like machine translation, where the encoding-decoding process can be used to successfully express the translation process mathematically.

These models are further improved by using different mechanisms and strategies like transfer learning, adaptive boosting, hyper-parameter tuning, attention, masking, negative sampling etc. These together with the discussed architectures lead to more complicated and powerful models.

1.1.2 Decision-Tree

A decision tree is one of the most popular artifacts in Machine Learning. Decision trees offer a nice way of representing the necessary criteria behind making decisions. It has been used for quite some time in different domains, to characterize and express the rationale behind taking decisions. More recently, with the introduction of statistical machine learning, decision trees are learnt. Specifically, given a set of data points, we need to take certain decisions based on the data. The decision tree that is learnt tells one which features influence the decision. This tells the user the rationale behind the decision making and provides explainable information. A decision tree takes as input an object or situation described by a set of properties, and outputs a yes/no *decision*. The decision is based on a number of features or decision variables, values of which typically lead to the yes/no answer. There are two types of nodes present in a decision tree, namely a decision node and a leaf node. Decision nodes typically appear at the non-leaf intermediate levels and take decisions on the basis of some conditions on the feature variables, while a leaf node typically represents a labeling / classification outcome. The condition embodied inside a decision node helps us interpret a decision tree.

The simplest variant of decision tree takes as input a a set of properties, and decides how to take a yes / no decision. The following example explains the idea of a decision tree in more detail. Consider the following scenario. On a weekend evening, a person wishes to dine out. He visits a restaurant and

needs to decide whether to wait for a table at a restaurant or move on to another restaurant. The following may be a list of variables

- Alternate: whether there is a suitable alternative restaurant
- Lounge: whether the restaurant has a lounge for waiting customers
- Fri/Sat: true on Fridays and Saturdays
- Hungry: whether we are hungry
- Patrons: how many people are in it (None, Some, Full)
- Price: the restaurant's rating (1-star, 2-star, 3-star, 4-star, 5-star)
- Raining: whether it is raining outside
- Reservation: whether we made a reservation
- Type: the kind of restaurant (Indian, Chinese, Thai, Fastfood)
- WaitEstimate: 0-10 mins, 10-30mins, 30-60mins, >60mins.

Consider that the following portion of the given data is present, which characterizes the yes/no decisions taken by different users in different situations, where the values of the different variables above are known. The available data based on which the decision tree is built is shown in Figure 1.1. Based on the available data, we can automatically build a decision tree such that the decision at the leaf levels show the yes/no decision taken corresponding to the different scenarios encountered (in terms of the values of the variables above) by different users. It is expected that the decision tree thus built will aid an user to predict the yes / no answer for an unknown situation not captured in the table below. However, given the values of the variables in the new situation, one case walk down the different paths of the tree to reach a leaf labeled with a yes / no answer.

Alternate	Lounge	Fri / Sat	Hungry	Patrons	Price	Raining	Reservation	Type	Wait Estimate	Decision
Yes	Yes	[illegible]	No	Full	⋆⋆	No	Yes	French	10-30	Yes
No	Yes	No	Yes	Full	⋆⋆⋆	Yes	Yes	Italian	1-10	Yes
Yes	No	Yes	No	Full	⋆	Yes	No	Burger	> 60	No
Yes	Yes	Yes	Yes	Full	⋆⋆⋆	No	Yes	French	10-30	No
Yes	[illegible]	No	Yes	Full	⋆⋆	No	No	Thai	30-60	No
No	No	No	No	Full	⋆	No	No	Thai	30-60	No
Yes	No	Yes	Yes	Some	⋆	No	No	Burger	1-10	Yes
No	No	No	No	Some	⋆	No	No	Thai	10-30	Yes
No	Yes	No	No	Some	⋆⋆	Yes	No	Thai	30-60	Yes
Yes	No	Yes	No	Some	⋆	Yes	No	Burger	10-30	Yes
No	No	Yes	Yes	None	⋆⋆	No	Yes	Italian	> 60	No
Yes	No	No	No	None	⋆	Yes	No	Burger	1-10	No

Table 1.1: Available Data points

An example decision tree built on the data in Table 1.1 is shown in Figure 1.2.

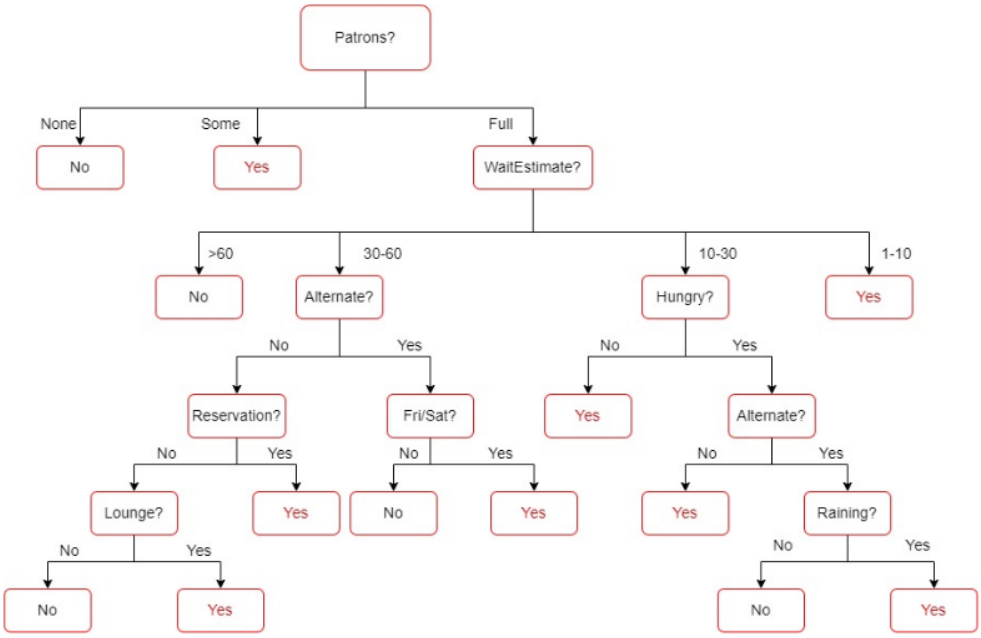

Figure 1.2: An example decision tree

In the automated process of learning decision trees, the input is data. Someone has collected the data above from various restaurants, and have found out that different people have taken different yes / no decisions for different situations at different restaurants. As we can see from the data above, for different values of the attributes, someone has taken a yes / no decision. Indeed, in a practical situation, we need a lot of meaningful data for explaining a yes / no answer. Each row of the data table concerns different situations encountered by different people. Considering the above data, an example decision tree that can be built is shown in Figure 1.2. To interpret a decision tree, we start from the variable at the root and traverse the different paths to see if we reach a yes / no leaf.

Decision tree learning is about learning how we can construct the decision tree from the data. It may be noted that the data shown in the figure above is only a subset of the full dataset. The decision variable here is whether the customer waits for a table or not, and the attributes are chosen in a way that influences the decision. For building the decision tree, we need to choose the attributes carefully. We explain a sample decision tree building session on the data above. We can see that for some rows in Table 1.1, the value of Patrons is None, for some rows it is Some, and for some, it is Full. We can thus partition the rows of the data available based on the value of Patrons. In other words, we split the data based on the values of the chosen feature Patrons. Intuitively, we have the following steps in decision tree construction.

- Identify an attribute which helps in separating the Yes answers from the No answers, and make it the root.
- Split the data based on the values of the chosen feature.
- For each subset thus created
  - If the subset contains all Yes or all No, then create a leaf node with that decision

– Otherwise, the subset contains some Yes and some No, and we recursively use the first two steps

In some cases, we need to terminate the recursion earlier even though some yes / no answers remain. This is done to avoid overfitting if we keep on expanding beyond a certain number of levels in the decision tree. The key question here is which attribute is a good discriminator between the Yes and No decisions. The best discriminator needs to be chosen at each level of the decision tree to best separation of the available data points, with an expectation that some branch will end up in leaf nodes. Before we discuss the rationale being choosing the best discriminator, we first present below a simple decision tree building algorithm, as shown in Algorithm 1.

Algorithm 1: DTL Algorithm

```
procedure function DTL(examples ,attributes ,default)
if examples is empty then
    return default
else if all examples have the same classification then
    return the classification
else if attributes is empty then
    return MODE(examples)
else
    best ← CHOOSE-ATTRIBUTE(attributes,examples);
    tree ← a new decision tree with root test best;
    for each value v_i of best do
        examples_i ← elements of examples with best = v_i ;
        subtree ← DTL(examples_i, attributes − best, MODE(examples))
        add a branch to tree with label v_i and subtree subtree ;
    return tree
end procedure
```

The aim of Algorithm 1 is to find a small tree consistent with the training examples. As we can see from the description, the algorithm returns empty if the set of examples / datapoints are all processed. If all datapoints currently remaining have the same classification, then it returns the classification. However, if the attribute / feature set is empty while more data points remain to be handled, the algorithm takes the majority (MODE) classification and returns that. If none of the above arises, the algorithm is in a situation where there is a set of data points that needs to be classified and there is a set of attributes. The main idea is to (recursively) choose the "most significant" attribute as the root of the (sub) tree. Let us now understand the rationale behind choosing a good attribute. An ideal choice for this will split the examples into subsets that are (ideally) all positive or all negative. However, that seldom happens and we have a mix of the cases. To mathematically quantify the notion of the best choice and compare between the different attributes that can be chosen as discriminators, we often use the notion of Entropy [3]. To implement the Choose-Attribute in in the DTL algorithm above, we first define the notion of Information Content (I) / Entropy as:

$$I(P(v_1, v_2, \ldots v_n)) = \sum_{i=1}^{n} -P(v_j) log_2(P(v_j)) \tag{1.1}$$

For a training set containing p Yes (positive) examples and n No (negative) examples, we have:

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n} \tag{1.2}$$

Based on the above, we choose an attribute to split on. A chosen attribute A divides the training set E into subsets $E_1, \ldots E_v$ according to their values for A, where A has v distinct values. We now define the concept of remainder(A).

$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} \times I(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}) \tag{1.3}$$

Information Gain (IG) or reduction in entropy from the attribute test is defined as follows:

$$IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - remainder(A) \tag{1.4}$$

We choose the attribute for which the information gain is the maximum. For our example above, if we work out the IG values, we will see that Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root. The rest of the decision tree is built along similar lines.

The above example shows an instance of a situation with discrete valued attributes and that too, for a 2-class (Yes / No) classification problem. Decision trees can be built for continuous valued attributes and also for other types of prediction problems. There is a rich literature on decision trees.

1.2 Explainability

Explainability in machine leaning concerns the model's decision with respect to the outputs produced. Interpretability is the degree to which a human can consistently predict the model's result. The higher the interpretability of a machine learning model, the easier it is for someone to comprehend why certain decisions or predictions have been made. A model is better interpretable than another model if its decisions are easier for a human to comprehend than decisions from the other model [1]. In ML parlance, the terms interpretable and explainable are often used interchangeably.

1.3 Motivation of this dissertation

A neural network is like a black box model. While it is possible to generate high accuracy predictions with neural networks, the major challenge is in interpreting the rationale behind their computations. As the values of the input features pass through the different hidden layers and the activations, they undergo different value transformations, which are opaque to a human user, and thus poses an arsenal of challenges for interpretation. In fact, explainability of ML models, as they learn to generalize from datasets, is one of the top challenges before the ML research community today. Unless these models are explainable and open to interpretation, it is very difficult to use them in safety critical systems where every failure counts and needs to be diagnosed. One of the main hurdles that has impeded the widespread adoption of ML models in safety critical systems is the lack of dependability. Additionally, in situations where humans and machines collaborate to achieve a given task, it is very difficult to use a neural model since humans do not know what is going on inside and how the model is taking decisions. This poses a tough challenge for widespread adoption of ML models in a number of domains where accountability, explainability and interpretability is of utmost concern [5]. This thesis is an attempt to address the explainability aspect of neural networks.

1.4 Contributions of this dissertation

In this thesis, we examine the problem of explainability of neural networks using decision trees. We take as input an already trained neural network and attempt to interpret it using it's traces, assuming that these traces are sufficiently representative of the classification decisions that it takes. We assume that the original training dataset is not available to us, all that we are aware of are the lower and upper bounds of the input features. Towards the objective of interpreting the neural network, we first create a set of random example feature vectors, and simulate the given neural network on the generated points to extract the label that it assigns to each point. From this labelled dataset, we build a decision tree following the classical decision tree construction algorithm. Finally, we examine the accuracy and performance of the trained decision tree on the original training examples. We experiment with our method on the IRIS, Wine and Breast Cancer datasets and analyze the results. Our findings show and affirm our belief that a well crafted decision tree gives an interpretable artifict towards neural network explanations.

1.5 Organization of the dissertation

This dissertation is organized into 3 chapters. A summary of the contents of the chapters is as follows:

Chapter 1: This chapter contains the introduction, relevant background and explains the motivation of this work.

Chapter 2: This chapter describes our novel approach in detail.

Chapter 3: We summarize with conclusions and discuss the future works arising out of this dissertation.

Chapter 2

Neural Network interpretation with Decision Trees

2.1 The Basic Methodology

Neural networks are universal approximators and they can learn a large class of complex non-linear functions. A main drawback of neural networks is that they act as black boxes and are not interpretable. There have been several attempts to make neural networks interpretable. In this work we aim to represent a neural network using a decision tree. In this section we will outline the basic methodology that we follow.

We fix some basic notations first. Throughout we will use bold faced characters to represent vectors. In general we will only use finite dimensional real vectors, if $\boldsymbol{x} \in \Re^p$ then $\boldsymbol{x}_j$ for $j \in \{1, 2, \ldots, p\}$ will denote the j^{th} component of $\boldsymbol{x}$ which we will also sometimes call as the j^{th} feature or the j^{th} attribute of $\boldsymbol{x}$. For a natural number n, we will denote the set $\{1, 2, \ldots, n\}$ by $[n]$. If $\mathcal{S}$ be a set, $x \xleftarrow{\$} \mathcal{S}$ will be used to denote that x is drawn uniformly at random from $\mathcal{S}$.

Suppose we have a training data set $X = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) : \boldsymbol{x}^{(i)} \in \Re^p, \boldsymbol{y}^{(i)} \in \{0, 1\}^q, i \in [N]\}$ where each $\boldsymbol{x}^{(i)}$ is a real vector representing the input features and $\boldsymbol{y}^{(i)}$ is the corresponding class label. We assume that the class labels are encoded as binary vectors: if the number of classes is q then each class label $\boldsymbol{y}$ is an element of $\{0, 1\}^q$ and if $\boldsymbol{y}$ represents class j the $\boldsymbol{y}_j = 1$ and $\boldsymbol{y}_i = 0$, for $i \neq j$.

For a data set X with input vectors in $\Re^p$, we define $U(X), L(X) \in \Re^p$ as

$$\begin{aligned} U(X)_j &= \max\left\{\boldsymbol{x}_j^{(1)}, \boldsymbol{x}_j^{(2)}, \ldots, \boldsymbol{x}_j^{(N)}\right\} \\ L(X)_j &= \min\{\boldsymbol{x}_j^{(1)}, \boldsymbol{x}_j^{(2)}, \ldots, \boldsymbol{x}_j^{(N)}\}. \end{aligned}$$

Hence, $[L(X)_j, U(X)_j]$ represents the smallest interval within which the values of the j^{th} input feature of X lies. Further by $[L(X), U(X)]$ we will denote the smallest p dimensional hyper-rectangle which contains all the input vectors in X.

Let $\mathcal{N}_X$ be a neural network trained using the data set X. Given $\mathcal{N}_X$, we want to extract rules from $\mathcal{N}_X$ without the knowledge of X. Though X is unknown, we assume that we have some basic structural information regarding X, in particular we assume the knowledge of the following:

```
ExtractRules(N_X, U(X), L(X), p, n)
01.S ← ∅;
02.for i = 1 to n,
03.    for j = 1 to p,
04.        x̃_j ←$ [L(X)_j, U(X)_j];
05.    end for
06.    ỹ ← N_X(x̃);
07.    S ← S ∪ {(x̃, ỹ)};
08.end for
09.D ← MakeDecisionTree(S);
10.return D;
```

Figure 2.1: The basic methodology to extract rules from a trained neural network $\mathcal{N}$.

1. The input dimension p of X.
2. The output dimension q of X.
3. The interval of each feature value of X, i.e., $[L(X)_j, U(X)_j]$ for each $j \in [p]$.

Note that given $\mathcal{N}_X$, for most neural network architectures the structure of the network will give us the values of p and q. For example, if the trained network is a multilayered perceptron for classification where the outputs are encoded as binary vectors as described above, the number of input nodes would correspond to p and the number of output nodes would correspond to q. As we do not assume anything regarding the architecture or the training methodology for $\mathcal{N}$ we prefer to have explicit knowledge of p and q.

The basic procedure ExtractRules to train a decision tree using the trained network $\mathcal{N}_X$ is shown in Figure 2.1. The procedure ExtractRules takes as input a neural network trained on X and some structural information regarding X in the form of the input dimension p and $L(X)$, $U(X)$ and a parameter n. The procedure generates n points uniformly at random in the hyper-rectangle $[L(X), U(X)]$ and then it labels these points using the trained neural network $\mathcal{N}_X$ and thus obtains a labelled data set $\mathcal{S}$ of n points. Further a decision tree $\mathcal{D}$ is constructed using the labeled set $\mathcal{S}$.

Some features of this procedure are worth discussing here:

1. The procedure described in Figure 2.1 is quite general. The neural network is used only for computing the labels of the generated input data. Any other classifier can be easily plugged in place of the neural network and the whole methodology should work.
2. The procedure makes minimal assumption regarding the training data. Thus this can be useful in scenarios where we have only a black-box access to a classifier and want to interpret the functioning of it.

2.2 Building the Decision Tree

We build the decision tree based on the famous C4.5 algorithm. For the algorithm we need to discuss a few concepts.

Let $X = \{(\boldsymbol{x}^{(i)}, y^{(i)}) : i \in [N]\}$, deviating from the previous discussion, we do not require the input vector $\boldsymbol{x}$ to be a real vector. We assume that for each $j \in [p]$ and $i \in [N]$, $\boldsymbol{x}_j^{(i)} \in \mathsf{Vals}(j)$, i.e, the j^{th} feature takes values from the set $\mathsf{Vals}(j)$ also for each $i \in [N]$, $y^{(i)} \in \mathcal{C} = \{c_1, c_2, \ldots, c_k\}$, i.e., the class labels come from a discrete set $\mathcal{C}$ of k classes. The specific encoding of these labels as vectors will not be important here.

For X as defined above and $i \in [k]$ we define $[X]_i$ to be a subset of X which contains all points whose class label is c_i. More precisely,

$$[X]_i = \{(\boldsymbol{x}, y) \in X : y = c_i\}.$$

Also, for a feature j and $v \in \mathsf{Vals}(j)$, we define

$$S_j(v) = \{(\boldsymbol{x}, y) \in X : \boldsymbol{x}_j = v\}.$$

We define the entropy of X as

$$H(X) = -\sum_{i \in [k]} \frac{|[X]_i|}{|X|} \lg \frac{|[X]_i|}{|X|}, \tag{2.1}$$

and the conditional entropy of X conditioned on the feature j, as

$$H(x|j) = \sum_{v \in \mathsf{Vals}(j)} \frac{|S_j(v)|}{|X|} H(S_j(v)). \tag{2.2}$$

Finally, we define the information gain $\mathsf{Gain}(X, j)$ of a feature j relative to the set X as

$$\mathsf{Gain}(X, j) = H(X) - H(X|j). \tag{2.3}$$

In most decision tree algorithms, including C4.5, information gain is used as the main criteria to select the best feature at each step to grow the tree [3]. In our variant of the decision tree induction algorithm we do the same. Additionally we control some parameters of the tree to avoid overfitting, in particular we set limits to the maximum depth (MAX-DEPTH) and the minimum number of samples per node (MIN-SAMPLES) in the tree and a threshold (THRES) on the information gain. Our algorithm is motivated by the one proposed in [6].

In the following algorithm MakeDecisionTree we explain our method to construct a decision tree. The method closely follows the procedure of C4.5 with some additional constraints on the tree structure. In particular we use constraints on the depth, minimum number of samples in each node and the maximum information gain, these constrains are MAX-DEPTH, MIN-SAMPLES and THRES respectively.

Algorithm 2: Algorithm for Building The Decision Tree

procedure MakeDecisionTree(X, T)
if X *contains only training examples of the same class* $c_j \in \mathcal{C}$ **then**
  make T a leaf node labeled with class c_j
else if $A = \emptyset$ **then**
  make T a leaf node labeled with class c_j, which is the most frequent class in X
else
  ▷ X contains examples belonging to mixture of classes. We select a single attribute to partition X into subsets so that each subset is purer
  if $(|X| \geq$ *MIN-SAMPLES*$) \bigwedge (DEPTH(T) \leq$ *MAX-DEPTH*$)$ **then**
    for $j \in [p]$ **do**
      $g(j) =$ Gain(X, j)
      $j^* = \underset{i \in [p]}{\text{argmax}} \; g(j)$
    if $g(j^*) <$ *THRES* **then**
      make T a leaf node labeled with c_j ,the most frequent class in X
    else
      Make T a decision node on j^*, let Vals$(j^*) = \{v_1, v_2, ...v_m\}$.Partition X into m Disjoint subsets $S_{j^*}(v_1), S_{j^*}(v_2), \ldots, S_{j^*}(v_m)$ based on the m values of the attribute j^*
      for $i \in [m]$ **do**
        if $S^*_j(v_i) \neq \emptyset$ **then**
          create an edge node T^*_j for v_i as a child node of T ;
          MakeDecisionTree$(S^*_j(v_i) \setminus \{j^*\}, T_j)$
            ▷ $S^*_j(v_i) \setminus \{j^*\}$ represents the set $S^*_j(v_i)$ with the attribute j^* is removed
end procedure

2.3 Experiments and Analysis of Results

We test our methodology on three real classification data sets: Iris, Wine and Breast-cancer. The basic characteristics of these data sets are depicted in Table 2.1.

The experimental protocol involves the following:

1. For each data set we divide it into training, validation and test sets and train a neural network. In all cases our networks are feed forward multilayer perceptrons. Different activation functions and hidden nodes for each data set were used. Various parameters were tried and the ones with the best result was selected for the final network.

2. Additional data was generated following the procedure ExtractRules. The number of data points

Data Sets	Number of points	Number of features	Number of classes
Iris	150	4	3
Wine	178	13	3
Breast-cancer	569	32	2

Table 2.1: The data sets used in the experiments

Data Set	Hidden Layers	Nodes in Hidden Layers	Activation Functions
Iris	2	16, 16	ReLu, ReLu, Softmax
Wine	2	13, 10	ReLu, ReLu, Softmax
Breast-cancer	2	16, 16	ReLu, Relu, Sigmoid

Table 2.2: Parameters of the trained networks. In the Activation Functions column the last entry is of the output layer and the preceding ones are of the hidden layers in order.

generated (the parameter n in the procedure) was approximately ten times the original data set. For Iris and Wine data sets 1500 points were generated and for breast cancer 2500 data points were generated. These points were labeled using the neural network as described above. Decision tree was built using these labeled data points and the performance of the decision tree was evaluated on the original data set.

The different parameters used in training the neural networks are summarized in Table 2.2. For training all networks we used the Adam Optimizer with a learning rate of 0.001.

The final performance of the trained networks in terms of training, validation and test accuracy are shown in Table 2.3.

Dataset	Train-acc	Validation-acc	Test-acc
Iris	91.67 %	91.67 %	100 %
Wine	94.69 %	100 %	97.22 %
Breast-cancer	95.60 %	93.41 %	97.37 %

Table 2.3: Neural Network accuracy

Using the method described in Figure 2.1 new data points were generated for each data sets uniformly at random from the smallest hyper-rectangle bounding the training data points. The generated points were labeled using the trained networks. Finally, with this generated labeled data, a decision tree was built.

For each data set we ran the procedure ExtractRules (see Figure 2.1) 10 times. Thus, each run generated a different decision tree as for each run the points generated are random thus in each run a decision tree is built on a different data set. The performance of each decision tree is measured on the original training data. The average performance of the ten trees along with the standard deviation is reported in Table 2.4. Note that in the table, the training accuracy is on the data set which is generated by the procedure ExtractRules and the testing accuracy is on the original data set. The average depth, the number of leaf nodes and data points associated with each leaf node is also depicted in Table 2.4.

Dataset	Generated Dataset						
	Training		Testing		# Leaf	Hyperparameter	
	Mean	St. Deviation	Mean	St. Deviation		Depth	#Leaf-examples
Iris	94.30	0.27	88.48	5.20	57	10	20
Wine	96.95	0.28	82.35	1.89	98	7	2
Breast-cancer	97.43	0.32	87.34	2.48	25	10	30

Table 2.4: Decision Tree: Results on Generated Data Points

For easy comparison, we also trained a decision tree on the original data set. The results for this experiment is shown in Table 2.5.

Data Set	Training	Validation	Test	No. of Leafs	Depth	Data Points per leaf
Iris	97.78	93.33	96.67	5	3	2
Wine	99.68	84.43	83.33	5	7	5
Breast-cancer	96.26	92.76	95.61	7	5	4

Table 2.5: Decision Tree: Results on Original Data Set

2.3.1 Analysis of the Results

Now, we present some preliminary analysis on the results obtained.

1. From Table 2.4 we see that the Decision Trees built on the generated data sets gives a reasonable accuracy of around 80% on all data sets. Thus, the decision trees can approximate the neural networks, but not to a great extent. In all cases the accuracy obtained is poorer than that obtained by the neural network. One of the reasons for this that our methodology to generate the input vectors is rather naive. We assume very little knowledge about the training data and generate random data points within the bounding box of the original data which is unlikely to capture the true distribution by which the data set gets generated. Hence it is interesting that with even such a little assumption on the training data we can get a reasonable approximation of the trained network.

2. As evident from Table 2.4 the depth of the Decision trees obtained and the number of leaf nodes is quite large. Thus, the obtained decision trees are not very interpretable in terms of the rules that can be extracted from them. For example, we show the rules extracted from a Decision Tree built on Iris Data using MakeDecisionTree on the original training data. The tree in Figure 2.2 is much smaller and thus more interpretable compared to the trees built on the generated data sets. It would have been better if we had shorter decision trees which would have been more interpretable.

3. The high standard deviation of the testing accuracy as is evident from Table 2.4 suggests that there is some degree of overfitting in the decision trees. Standard methods to reduce overfitting may help to correct this and this may also yield shorter and more interpretable trees.

4. To test the quality of the decision trees we performed another experiment. For each data set we generated 10 decision trees, we tested the accuracy of the ensemble of these ten decsion trees. The results of this experiment is shown in Table 2.6. The results show that the ensemble performs better than the individual trees as expected.

Dataset	Test-accuracy
Iris	93.33 %
Wine	85.95 %
Breast-cancer	89.98 %

Table 2.6: Decision Tree Ensembles result

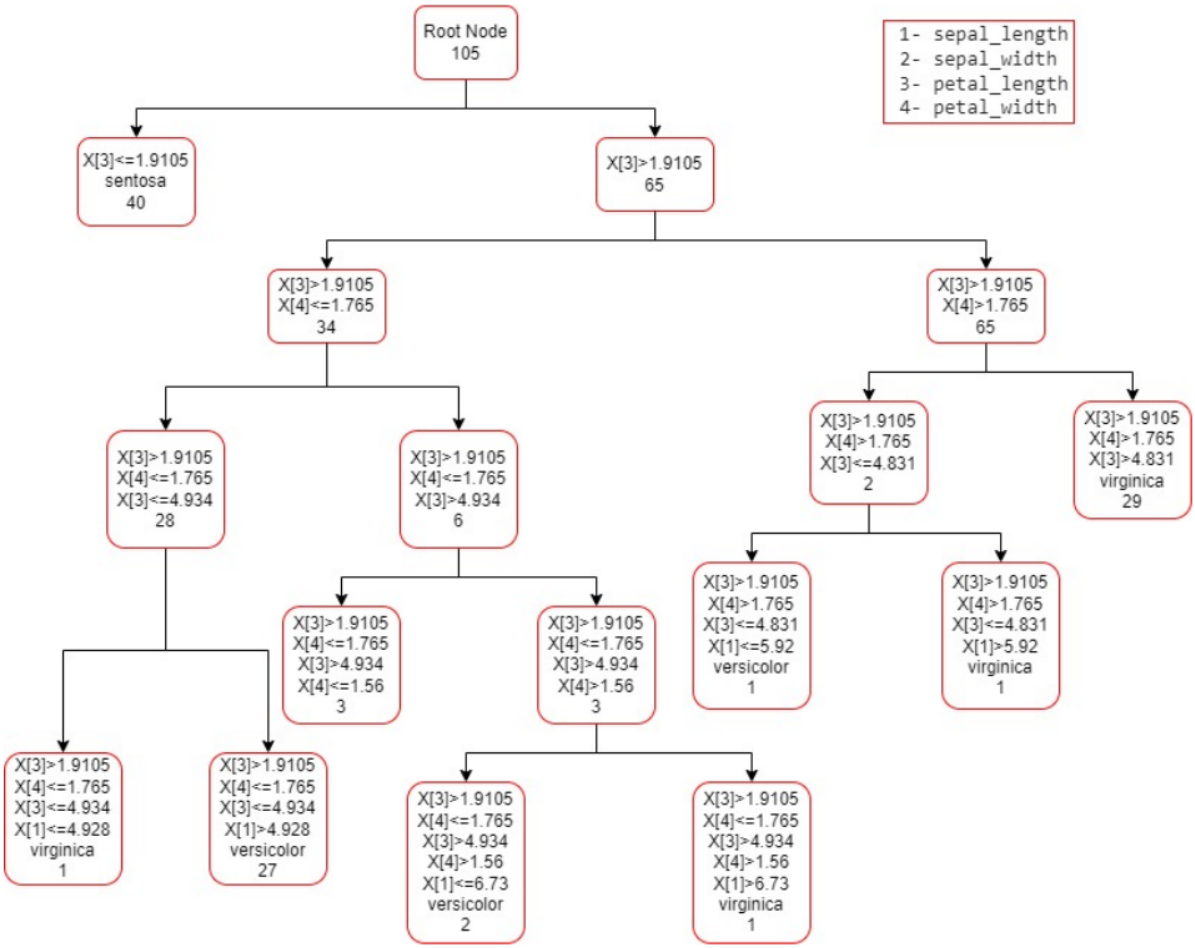

Figure 2.2: Decision Tree built using MakeDecisionTree on Iris data. We use 105 points among the 150 points for training

Chapter 3

Conclusion and Future Work

A significant limitation of neural networks is that they are not interpretable to humans. In this thesis, we propose an approach towards neural network interpretation with decision trees, which explains the black box neural models with the help of decision tree models that approximately describe the network's concept representations. From our experiments, we have results to show that decision trees indeed make the networks more interpretable [2].

As future work, we plan to study how explainability can be built into the training process, such that we have some pointers towards interpretation as the network gets built on training data. We plan to extend our approach to more neural network benchmark examples to show the efficacy of this interpretability to more complex structures. Additionally, we wish to explore other ML models and check their suitability towards the interpretability task.

Bibliography

[1] Kim, B., Khanna, R., and Koyejo, O. O. Examples are not enough, learn to criticize! criticism for interpretability. In *Advances in Neural Information Processing Systems* (2016), D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc.

[2] Pan, W., and Zhang, C. The definitions of interpretability and learning of interpretable models, 2021.

[3] Quinlan, J. R. Improved use of continuous attributes in c4.5.

[4] Ribeiro, M. T., Singh, S., and Guestrin, C. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[5] Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., and Zhong, C. Interpretable machine learning: Fundamental principles and 10 grand challenges.

[6] Xu, Z., Min, F., and Zhu, W. Cost-sensitive c4.5 with post-pruning and competition, 2012.

Interpreting Neural Networks with Decision Trees

ORIGINALITY REPORT

10%
SIMILARITY INDEX

PRIMARY SOURCES

1	www.coursehero.com Internet	127 words — 2%
2	library.isical.ac.in:8080 Internet	81 words — 1%
3	www.springer.com Internet	74 words — 1%
4	www.facweb.iitkgp.ernet.in Internet	71 words — 1%
5	ebin.pub Internet	64 words — 1%
6	www.cs.uiowa.edu Internet	53 words — 1%
7	link.springer.com Internet	51 words — 1%
8	S RUSSELL. "Machine Learning", Artificial Intelligence, 1996 Crossref	32 words — < 1%
9	www.cs.bham.ac.uk Internet	31 words — < 1%

10	de.scribd.com Internet	22 words — < 1%
11	Mohammed Alswaitti, Mohanad Albughdadi, Nor Ashidi Mat Isa. "Density-based particle swarm optimization algorithm for data clustering", Expert Systems with Applications, 2018 Crossref	19 words — < 1%
12	Ammari, Habib M. "Energy-efficient connected k-coverage, duty-cycling, and geographic forwarding in wireless sensor networks", Proquest, 20111108 ProQuest	18 words — < 1%
13	s-space.snu.ac.kr Internet	16 words — < 1%
14	creativecommons.org Internet	14 words — < 1%
15	www.neuroinf.pl Internet	14 words — < 1%

EXCLUDE QUOTES	ON	EXCLUDE SOURCES	< 14 WORDS
EXCLUDE BIBLIOGRAPHY	ON	EXCLUDE MATCHES	< 14 WORDS