Neural Networks are Decision Trees

Caglar Aytekin
AI Lead
AAC Technologies
caglaraytekin@aactechnologies.com, cagosmail@gmail.com

Abstract

In this manuscript, we show that any neural network with any activation function can be represented as a decision tree. The representation is equivalence and not an approximation, thus keeping the accuracy of the neural network exactly as is. We believe that this work provides better understanding of neural networks and paves the way to tackle their black-box nature. We share equivalent trees of some neural networks and show that besides providing interpretability, tree representation can also achieve some computational advantages for small networks. The analysis holds both for fully connected and convolutional networks, which may or may not also include skip connections and/or normalizations.

1. Introduction

Despite the immense success of neural networks over the past decade, the black-box nature of their predictions prevent their wider and more reliable adoption in many industries, such as health and security. This fact led researchers to investigate ways to explain neural network decisions. The efforts in explaining neural network decisions can be categorized into several approaches: saliency maps, approximation by interpretable methods and joint models.

Saliency maps are ways of highlighting areas on the input, of which a neural network make use of the most while prediction. An earlier work [20] takes the gradient of the neural network output with respect to the input in order to visualize an input-specific linearization of the entire network. Another work [26] uses a deconvnet to go back to features from decisions. The saliency maps obtained via these methods are often noisy and prevent a clear understanding of the decisions made. Another track of methods [29], [18], [4], [6] make use of the derivative of a neural network output with respect to an activation, usually the one right before fully connected layers. This saliency maps obtained by this track, and some other works [27], [11], [5] are clearer in the sense of highlighting areas related to the predicted class. Although useful for purposes such as checking whether the support area for decisions are sound, these methods still lack a detailed logical reasoning of why such decision is made.

Conversion between neural networks and interpretable by-design models -such as decision trees- has been a topic of interest. In [8], a method was devised to initialize neural networks with decision trees. [9, 19, 25] also provides neural network equivalents of decision trees. The neural networks in these works have specific architectures, thus the conversion lacks generalization to any model. In [24], neural networks were trained in such a way that their decision boundaries can be approximated by trees. This work does not provide a correspondence between neural networks and decision trees, and merely uses the latter as a regularization. In [7], a neural network was used to train a decision tree. Such tree distillation is an approximation of a neural network and not a direct conversion, thus performs poorly on the tasks that the neural network was trained on.

Joint neural network and decision tree models [12], [16], [13], [14], [17], [2], [10], [22] genarally use deep learning to assists some trees, or come up with a neural network structure so it resembles a tree. A recent work [23] replaces the final fully connected layer of a neural network with a decision tree. Since the backbone features are still that of neural networks, the explanation is sought to be achieved via providing a means to humans to validate the decision as a good or bad one, rather than a complete logical reasoning of the decision.

In this paper, we show that any neural network having any activations has a directly equivalent decision tree representation. Thus, the induced tree output is exactly the same with that of the neural network and tree representation doesn't limit or require altering of the neural architecture in any way. We believe that the decision tree equivalence provides better understanding of neural networks and paves the way to tackle the black-box nature of neural networks, e.g. via analyzing the category that a test sample belongs to, which can be extracted by the node rules that a sample is categorized. We show that the decision tree equivalent of

a neural network can be found for either fully connected or convolutional neural networks which may include skip layers and normalizations as well. Besides the interpretability aspect, we show that the induced tree is also advantageous to the corresponding neural network in terms of computational complexity, at the expense of increased storage memory.

Upon writing this paper, we have noticed the following works having overlaps with ours [28], [3], [15], [21], especially for feedforward ReLU networks. We extend the findings in these works to any activation function and also recurrent neural networks.

2. Decision Tree Analysis of Neural Networks

The derivations in this section will be first made for feed-forward neural networks with piece-wise linear activation functions such as ReLU, Leaky ReLU, etc. Next, the analysis will be extended to any neural network with any activation function.

2.1. Fully Connected Networks

Let $\mathbf{W}_i$ be the weight matrix of a network's i^{th} layer. Let σ be any piece-wise linear activation function, and $\mathbf{x}_0$ be the input to the neural network. Then, the output and an intermediate feature of a feed-forward neural network can be represented as in Eq. 1.

$$\begin{aligned} NN(\mathbf{x}_0) &= \mathbf{W}_{n-1}^T\sigma(\mathbf{W}_{n-2}^T\sigma(...\mathbf{W}_1^T\sigma(\mathbf{W}_0^T\mathbf{x}_0))) \\ x_i &= \sigma(\mathbf{W}_{i-1}^T\sigma(...\mathbf{W}_1^T\sigma(\mathbf{W}_0^T\mathbf{x}_0))) \end{aligned} \tag{1}$$

Note that in Eq. 1, we omit any final activation (e.g. softmax) and we ignore the bias term as it can be simply included by concatenating a 1 value to each x_i. The activation function σ acts as an element-wise scalar multiplication, hence the following can be written.

$$\mathbf{W}_i^T\sigma(\mathbf{W}_{i-1}^T\mathbf{x}_{i-1}) = \mathbf{W}_i^T(\mathbf{a}_{i-1} \odot (\mathbf{W}_{i-1}^T\mathbf{x}_{i-1})) \tag{2}$$

In Eq. 2, $\mathbf{a}_{i-1}$ is a vector indicating the slopes of activations in the corresponding linear regions where $\mathbf{W}_{i-1}^T\mathbf{x}_{i-1}$ fall into, $\odot$ denotes element-wise multiplication. Note that, $\mathbf{a}_{i-1}$ can directly be interpreted as a categorization result since it includes indicators (slopes) of linear regions in activation function. The Eq. 2 can be re-organized as follows.

$$\mathbf{W}_i^T\sigma(\mathbf{W}_{i-1}^T\mathbf{x}_{i-1}) = (\mathbf{W}_i \odot \mathbf{a}_{i-1})^T\mathbf{W}_{i-1}^T\mathbf{x}_{i-1} \tag{3}$$

In Eq. 3, we use $\odot$ as a column-wise element-wise multiplication on $\mathbf{W}_i$. This corresponds to element-wise multiplication by a matrix obtained via by repeating $\mathbf{a}_{i-1}$ column-vector to match the size of $\mathbf{W}_i$. Using Eq. 3, Eq. 1 can be rewritten as follows.

$$\begin{aligned} NN(\mathbf{x}_0) = (\mathbf{W}_{n-1} \odot \mathbf{a}_{n-2})^T(\mathbf{W}_{n-2} \odot \mathbf{a}_{n-3})^T \\ ...(\mathbf{W}_1 \odot \mathbf{a}_0)^T\mathbf{W}_0^T\mathbf{x}_0 \end{aligned} \tag{4}$$

From Eq. 4, one can define an effective weight matrix $\hat{\mathbf{W}}_i^T$ of a layer i to be applied directly on input $\mathbf{x}_0$ as follows:

$$\begin{aligned} \mathbf{c}_{i-1}\hat{\mathbf{W}}_i^T &= (\mathbf{W}_i \odot \mathbf{a}_{i-1})^T...(\mathbf{W}_1 \odot \mathbf{a}_0)^T\mathbf{W}_0^T \\ \mathbf{c}_{i-1}\hat{\mathbf{W}}_i^T\mathbf{x}_0 &= \mathbf{W}_i^T\mathbf{x}_i \end{aligned} \tag{5}$$

In Eq. 5, the categorization vector until layer i is defined as follows: $\mathbf{c}_{i-1} = \mathbf{a}_0 \parallel \mathbf{a}_1 \parallel ...\mathbf{a}_{i-1}$, where $\parallel$ is the concatenation operator.

One can directly observe from Eq. 5 that, the effective matrix of layer i is only dependent on the categorization vectors from previous layers. This indicates that in each layer, a new efficient filter is selected -to be applied on the network input- based on the previous categorizations/decisions. This directly shows that a fully connected neural network can be represented as a single decision tree, where effective matrices acts as categorization rules. In each layer i, response of effective matrix $\mathbf{c}_{i-1}\hat{\mathbf{W}}_i^T$ is categorized into $\mathbf{a}_i$ vector, and based on this categorization result, next layer's effective matrix $\mathbf{c}_i\hat{\mathbf{W}}_{i+1}^T$ is determined. A layer i is thus represented as k^{m_i}-way categorization, where m_i is the number filters in each layer i and k is the total number of linear regions in an activation. This categorization in a layer i can thus be represented by a tree of depth m_i, where a node in any depth is branched into k categorizations.

In order to better illustrate the equivalent decision tree of a neural network, in Algorithm 1, we rewrite Eq. 5 for the entire network, as an algorithm. For the sake of simplicity and without loss of generality, we provide the algorithm with the ReLU activation function, where $a \in \{0, 1\}$. It can be clearly observed that, the lines $5-9$ in Algorithm 1 corresponds to a node in the decision tree, where a simple yes/no decision is made.

The decision tree equivalent of a neural network can thus be constructed as in Algorithm 2. Using this algorithm, we share a a tree representation obtained for a neural network with three layers, having 2,1 and 1 filter for layer 1, 2 and 3 respectively. The network has ReLU activation in between layers, and no activation after last layer. It can be observed from Algorithm 2 and Fig. 1 that the depth of a NN-equivalent tree is $d = \sum_{i=0}^{n-2} m_i$, and total number of categories in last branch is 2^d. At first glance, the number of categories seem huge. For example, if first layer of a neural network contains 64 filters, there would exist at least 2^{64} branches in a tree, which is already intractable. But, there

Slopes of activations:
- For ReLU: 1 if neuron (x) > 0 else 0 when neuron < 0
- For Leaky Relu: 1 if neuron (x) > 0 else a small constant when neuron < 0

So a_i-1 is just a vector of [1, 0] or [1, small_constant]

Hmmm, so NNs are an efficient representation of a tree?

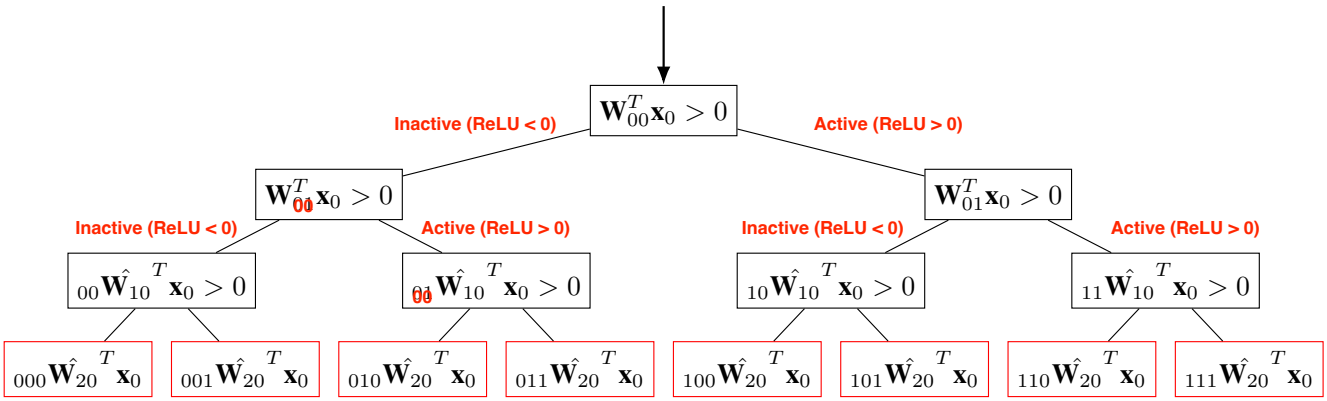


Figure 1. Decision Tree for a 2-layer ReLU Neural Network

may occur violating and redundant rules that would provide lossless pruning of the NN-equivalent tree. Another observation is that, it is highly likely that not all categories will be realized during training due to the possibly much larger number of categories (tree leaves) than training data. These categories can be pruned as well based on the application, and the data falling into these categories can be considered invalid, if the application permits. In the next section, we show that such redundant, violating and unrealized categories indeed exist, by analysing decision trees of some neural networks. But before that, we show that the tree equivalent of a neural network exists for skip connections, normalizations, convolutions, other activation functions and recurrence.

2.1.1 Skip Connections

We analyse a residual neural network of the following type:

$$
\begin{aligned}
{}_r\mathbf{x}_0 &= \mathbf{W}_0^T\mathbf{x}_0 \\
{}_r\mathbf{x}_i &= {}_r\mathbf{x}_{i-1} + \mathbf{W}_i^T\sigma({}_r\mathbf{x}_{i-1})
\end{aligned}
\tag{6}
$$

Using Eq. 6, via a similar analysis in Sec. 2.1, one can rewrite ${}_r\mathbf{x}_i$ as follows.

$$
\begin{aligned}
{}_r\mathbf{x}_i &= {}_{\mathbf{a}_{i-1}}\hat{\mathbf{W}}_i^T\,{}_r\mathbf{x}_{i-1} \\
{}_{\mathbf{a}_{i-1}}\hat{\mathbf{W}}_i^T &= \mathbf{I} + (\mathbf{W}_i \odot \mathbf{a}_{i-1})^T
\end{aligned}
\tag{7}
$$

Finally, using ${}_{\mathbf{a}_{i-1}}\hat{\mathbf{W}}_i^T$ in Eq. 7, one can define effective matrices for residual neural networks as follows.

$$
\begin{aligned}
{}_r\mathbf{x}_i &= {}_r\hat{\mathbf{W}}_i^T\mathbf{x}_0 \\
{}_r\hat{\mathbf{W}}_i^T &= {}_{\mathbf{a}_{i-1}}\hat{\mathbf{W}}_i^T\,{}_{\mathbf{a}_{i-2}}\hat{\mathbf{W}}_{i-1}^T\cdots{}_{\mathbf{a}_0}\hat{\mathbf{W}}_1^T\mathbf{W}_0^T
\end{aligned}
\tag{8}
$$

One can observe from Eq. 8 that, for layer i, the residual effective matrix ${}_r\hat{\mathbf{W}}_i^T$ is defined solely based on categorizations from the previous activations. Similar to the analysis in Sec. 2.1, this enables a tree equivalent of residual neural networks.

Algorithm 1: Algorithm of Eq. 5 for ReLU networks

$\hat{\mathbf{W}} = \mathbf{W}_0$
for $i = 0$ **to** $n-2$ **do**
  $\mathbf{a} = []$
  for $j = 0$ **to** $m_i - 1$ **do**
    if $\hat{\mathbf{W}}_{ij}^T\mathbf{x}_0 > 0$ **then**
      $\mathbf{a}.append(1)$
    else
      $\mathbf{a}.append(0)$
    end
  end
  $\hat{\mathbf{W}} = \hat{\mathbf{W}}(\mathbf{W}_{i+1} \odot \mathbf{a})$
end
return $\hat{\mathbf{W}}^T\mathbf{x}_0$

Hadammard Product

Algorithm 2: Algorithm of converting neural networks to decision trees

Initialize Tree: Set root.
Branch all leafs to k nodes, decision rule is first effective filter.
Branch all nodes to k more nodes, and repeat until all effective filters in a layer is covered.
Calculate effective matrix for each leaf via Eq. 5. Repeat 2,3.
Repeat 4 until all layers are covered.
return Tree

2.1.2 Normalization Layers

A separate analysis is not needed for any normalization layer, as popular normalization layers are linear, and after training, they can be embedded into the linear layer that it comes after or before, in pre-activation or post-activation normalizations respectively.

2.2. Convolutional Neural Networks

Let $\mathbf{K}_i : C_{i+1} \times C_i \times M_i \times N_i$ be the convolution kernel for layer i, applying on an input $\mathbf{F}_i : C_i \times H_i \times W_i$. Note that M_i and N_i denote the spatial size of the convolutional kernel, and H_i and W_i denote the spatial size of the input.

One can write the output of a convolutional neural network $CNN(\mathbf{F}_0)$, and an intermediate feature $\mathbf{F}_i$ as follows.

$$\begin{aligned} CNN(\mathbf{F}_0) &= \mathbf{K}_{n-1} * \sigma(\mathbf{K}_{n-2} * \sigma(...\sigma(\mathbf{K}_0 * \mathbf{F}_0)) \\ \mathbf{F}_i &= \sigma(\mathbf{K}_{i-1} * \sigma(...\sigma(\mathbf{K}_0 * \mathbf{F}_0)) \end{aligned} \tag{9}$$

Similar to the fully connected network analysis, one can write the following, due to element-wise scalar multiplication nature of the activation function.

$$\mathbf{K}_i * \sigma(\mathbf{K}_{i-1} * \mathbf{F}_{i-1}) = (\mathbf{K}_i \odot \mathbf{a}_{i-1}) * (\mathbf{K}_{i-1} * \mathbf{F}_{i-1}) \tag{10}$$

In Eq. 10, $\mathbf{a}_{i-1}$ is of same spatial size as $\mathbf{K}_i$ and consists of the slopes of activation function in corresponding regions in the previous feature $\mathbf{F}_{i-1}$. Note that the above only holds for a specific spatial region, and there exists a separate $\mathbf{a}_{i-1}$ for each spatial region that the convolution $\mathbf{K}_{i-1}$ is applied to. For example, if $\mathbf{K}_{i-1}$ is a 3×3 kernel, there exists a separate $\mathbf{a}_{i-1}$ for all 3×3 regions that the convolution is applied to. An effective convolution ${}_{\mathbf{c}_{i-1}}\hat{\mathbf{K}}_i$ can be written as follows.

$$\begin{aligned} {}_{\mathbf{c}_{i-1}}\hat{\mathbf{K}}_i &= (\mathbf{K}_i \odot \mathbf{a}_{i-1}) * ... * (\mathbf{K}_1 \odot \mathbf{a}_0) * \mathbf{K}_0 \\ {}_{\mathbf{c}_{i-1}}\hat{\mathbf{K}}_i * \mathbf{x}_0 &= \mathbf{K}_i * \mathbf{x}_i \end{aligned} \tag{11}$$

Note that in Eq. 11, ${}_{\mathbf{c}_{i-1}}\hat{\mathbf{K}}_i$ contains specific effective convolutions per region, where a region is defined according to the receptive field of layer i. $\mathbf{c}$ is defined as the concatenated categorization results of all relevant regions from previous layers.

One can observe from Eq. 11 that effecive convolutions are only dependent on categorizations coming from activations, which enables the tree equivalence -similar to the analysis for fully connected network. A difference from fully connected layer case is that many decisions are made on partial input regions rather than entire $\mathbf{x}_0$.

2.3. Continuous Activation Functions

In Eq. 2, for piece-wise linear activations, elements of $\mathbf{a}$ can have a number of values limited by the piece-wise linear regions in the activation function. This number defines the number of child nodes per effective filter. The extension to continuous activation functions is trivial as they can be considered as piece-wise linear functions with infinite regions. Therefore, for continuous activations, the neural network equivalent tree immediately becomes infinite width even for a single filter. This might not be a useful result, but we provide this discussion here for completeness. In order to guarantee finite trees, one may consider using quantized versions of continuous activations which may result in a few piece-wise linear regions, hence few child nodes per activation.

There's the intractability

2.4. Recurrent Networks

As recurrent neural networks (RNNs) can be unrolled to feed-forward representation, RNNs can also be equivalently represented as decision trees. We study following recurrent neural network. Note that we simply omit the bias terms as they can be represented by concatenating a 1 value to input vectors.

$$\begin{aligned} \mathbf{h}^{(t)} &= \sigma(\mathbf{W}^T\mathbf{h}^{(t-1)} + \mathbf{U}^T\mathbf{x}^{(t)}) \\ \mathbf{o}^{(t)} &= \mathbf{V}^T\mathbf{h}^{(t)} \end{aligned} \tag{12}$$

Similar to previous analysis, one can rewrite $\mathbf{h}^{(t)}$ as follows.

$$\mathbf{h}^{(t)} = \mathbf{a}^{(t)} \odot (\mathbf{W}^T\mathbf{h}^{(t-1)} + \mathbf{U}^T\mathbf{x}^{(t)}) \tag{13}$$

Eq. 13 can be rewritten follows.

$$\begin{aligned} \mathbf{h}^{(t)} = \mathbf{a}^{(t)} \odot (\prod_{j=(t-1)}^{1} (\mathbf{W}^T \odot \mathbf{a}^{(j)}))\mathbf{W}^T\mathbf{h}^{(0)} \\ + \mathbf{a}^{(t)} \odot \sum_{i=1}^{t} (\prod_{j=(t-1)}^{i} (\mathbf{W}^T \odot \mathbf{a}^{(j)}))\mathbf{U}^T\mathbf{x}^{(i)} \end{aligned} \tag{14}$$

Note that in Eq. 14, the product operator stands for matrix multiplication, its steps are -1 and we consider the output of product operator to be 1 when $i = t$. One can rewrite Eq. 14 by introducing ${}_{\mathbf{c}_j}\hat{\mathbf{W}}_j$ as follows.

$$\begin{aligned} \mathbf{h}^{(t)} &= \mathbf{a}^{(t)} \odot {}_{\mathbf{c}_1}\hat{\mathbf{W}}_1\mathbf{W}^T\mathbf{h}^{(0)} + \mathbf{a}^{(t)} \odot \sum_{i=1}^{t} {}_{\mathbf{c}_i}\hat{\mathbf{W}}_i\mathbf{U}^T\mathbf{x}^{(i)} \\ {}_{\mathbf{c}_i}\hat{\mathbf{W}}_i^T &= \prod_{j=(t-1)}^{i} (\mathbf{W}^T \odot \mathbf{a}^{(j)}) \end{aligned} \tag{15}$$

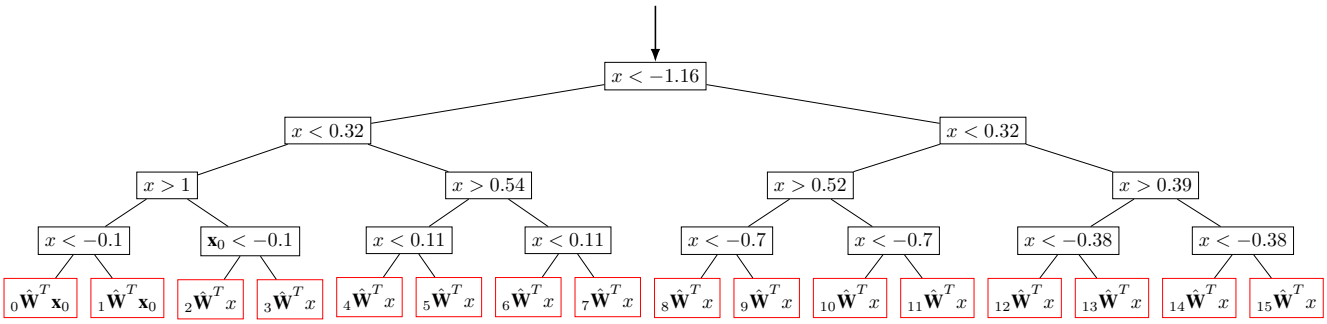
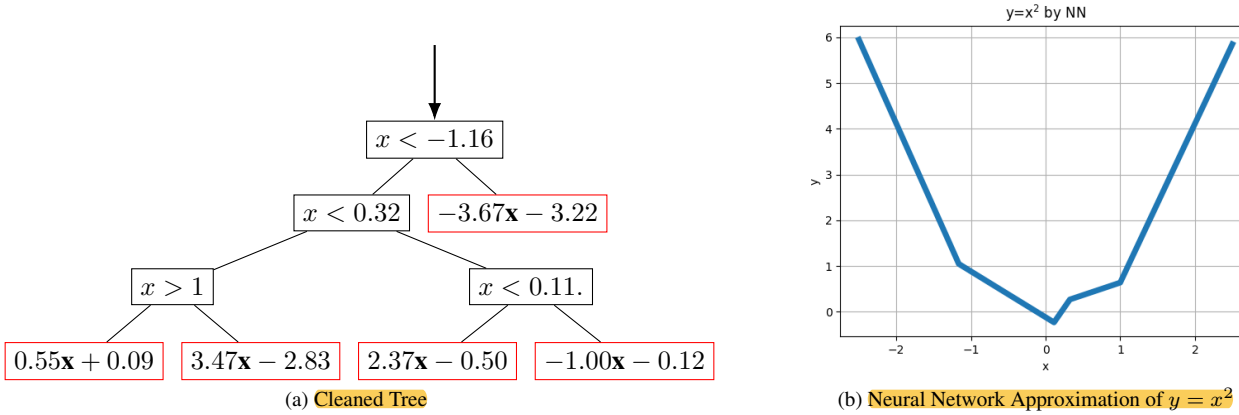


Figure 2. Decision Tree for a $y = x^2$ Regression Neural Network



(a) Cleaned Tree

(b) Neural Network Approximation of $y = x^2$

Figure 3. Cleaned Decision Tree for a $y = x^2$ Regression Neural Network

Combining Eq. 15 and Eq. 12, one can write $\mathbf{o}^{(t)}$ as follows.

$$\mathbf{o}^{(t)} = {}_{\mathbf{a}^{(t)}}\hat{\mathbf{V}}^T{}_{\mathbf{c}_1}\hat{\mathbf{W}}_1\mathbf{W}^T\mathbf{h}^{(0)} + {}_{\mathbf{a}^{(t)}}\hat{\mathbf{V}}^T\sum_{i=1}^{t}{}_{\mathbf{c}_i}\hat{\mathbf{W}}_i\mathbf{U}^T\mathbf{x}^{(i)} \quad (16)$$

Eq. 16 can be further simplified to the following.

$$\mathbf{o}^{(t)} = {}_{\mathbf{c}_1}\hat{\mathbf{Z}}_1^T\mathbf{W}^T\mathbf{h}^{(0)} + \sum_{i=1}^{t}{}_{\mathbf{c}_i}\hat{\mathbf{Z}}_i\mathbf{U}^T\mathbf{x}^{(i)} \quad (17)$$

In Eq. 17, ${}_{\mathbf{c}_i}\hat{\mathbf{Z}}_i^T = {}_{\mathbf{a}^{(t)}}\hat{\mathbf{V}}^T{}_{\mathbf{c}_i}\hat{\mathbf{W}}_i$ .As one can observe from Eq. 17, the RNN output only depends on the categorization vector $\mathbf{c}_i$, which enables the tree equivalence -similar to previous analysis.

Note that for RNNs, a popular choice for σ in Eq. 12 is $tanh$. As mentioned in Section 2.3, in order to provide finite trees, one might consider using a piece-wise linear approximation of $tanh$.

3. Experimental Results

First, we make a toy experiment where we fit a neural network to: $y = x^2$ equation. The neural network has 3 dense layers with 2 filters each, except for last layer which has 1 filter. The network uses leaky-ReLU activations after fully connected layers, except for the last layer which has no post-activation. We have used negative slope of 0.3 for leaky-ReLU which is the default value in Tensorflow [1]. The network was trained with 5000 (x, y) pairs where x was regularly sampled from $[-2.5, 2.5]$ interval. Fig. 2 shows the decision tree corresponding to the neural network. In the tree, every black rectangle box indicates a rule, left child from the box means the rule does not hold, and the right child means the rule holds. For better visualization, the rules are obtained via converting $\mathbf{w}^T x + \beta > 0$ to direct inequalities acting on x. This can be done for the particular regression $y = x^2$, since x is a scalar. In every leaf, the network applies a linear function -indicated by a red rectangle- based on the decisions so far. We have avoided writing these functions explicitly due to limited space. At first glance, the tree representation of a neural network in this example seems large due to the $2^{\sum_i^{n-2} m_i} = 2^4 = 16$ categorizations. However, we notice that a lot of the rules in the decision tree is redundant, and hence some paths in the decision tree becomes invalid. An example to redundant rule is checking $x < 0.32$ after $x < -1.16$ rule holds. This

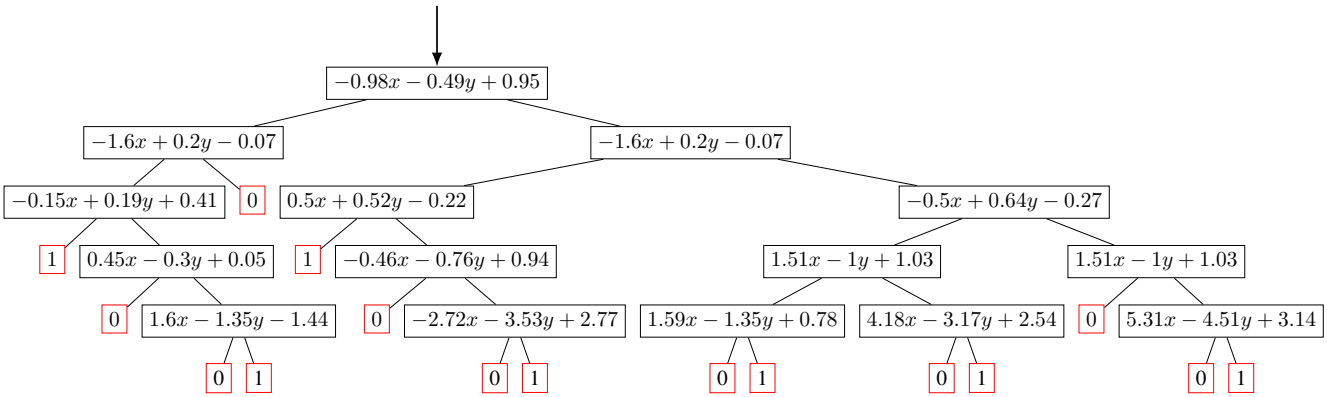


Figure 4. Classification Tree for a Half-Moon Classification Neural Network

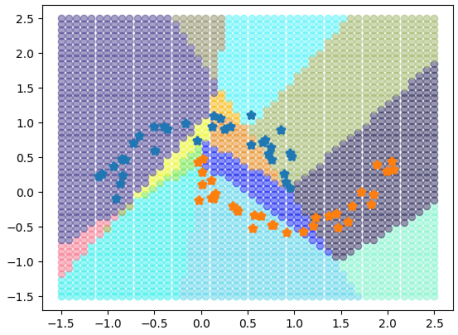


Figure 5. Categorizations made by the decision tree for half-moon dataset

directly creates the invalid left child for this node. Hence, the tree can be cleaned via removing the left child in this case, and merging the categorization rule to the stricter one : $x < -1.16$ in the particular case. Via cleaning the decision tree in Fig. 2, we obtain the simpler tree in Fig. 3a, which only consists of 5 categories instead of 16. The 5 categories are directly visible also from the model response in Fig. 3b. The interpretation of the neural network is thus straightforward: for each region whose boundaries are determined via the decision tree representation, the network approximates the non-linear $y = x^2$ equation by a linear equation. One can clearly interpret and moreover make deduction from the decision tree, some of which are as follows. The neural network is unable to grasp the symmetrical nature of the regression problem which is evident from the fact that the decision boundaries are asymmetrical. The region in below -1.16 and above 1 is unbounded and thus neural decisions lose accuracy as x goes beyond these boundaries.

Next, we investigate another toy problem of classifying half-moons and analyse the decision tree produced by a neural network. We train a fully connected neural network with

	$y = x^2$			Half-Moon		
	Param.	Comp.	Mult./Add.	Param.	Comp.	Mult./Add.
Tree	14	2.6	2	39	4.1	8.2
NN	13	4	16	15	5	25

Table 1. Computation and memory analysis of toy problems.

3 layers with leaky-ReLU activations, except for last layer which has sigmoid activation. Each layer has 2 filters except for the last layer which has 1. The cleaned decision tree induced by the trained network is shown in Fig. 4. The decision tree finds many categories whose boundaries are determined by the rules in the tree, where each category is assigned a single class. In order to better visualize the categories, we illustrate them with different colors in Fig. 5. One can make several deductions from the decision tree such as some regions are very well-defined, bounded and the classifications they make are perfectly in line with the training data, thus these regions are very reliable. There are unbounded categories which help obtaining accurate classification boundaries, yet fail to provide a compact representation of the training data, these may correspond to inaccurate extrapolations made by neural decisions. There are also some categories that emerged although none of the training data falls to them.

Besides the interpretability aspect, the decision tree representation also provides some computational advantages. In Table 1, we compare the number of parameters, float-point comparisons and multiplication or addition operations of the neural network and the tree induced by it. Note that the comparisons, multiplications and additions in the tree representation are given as expected values, since per each category depth of the tree is different. As the induced tree is an unfolding of the neural network, it covers all possible routes and keeps all possible effective filters in memory. Thus, as expected, the number of parameters in the tree representation of a neural network is larger than that of the network. In the induced tree, in every layer i, a maximum

of m_i filters are applied directly on the input, whereas in the neural network always m_i filters are applied on the previous feature, which is usually much larger than the input in the feature dimension. Thus, computation-wise, the tree representation is advantageous compared to the neural network one.

4. Conclusion

In this manuscript, we have shown that neural networks can be equivalently represented as decision trees. The tree equivalence holds for fully connected layers, convolutional layers, residual connections, normalizations, recurrent layers and any activation. We believe that this tree equivalence provides directions to tackle the black-box nature of neural networks.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 5

[2] Karim Ahmed, Mohammad Haris Baig, and Lorenzo Torresani. Network of experts for large-scale image categorization. In *European Conference on Computer Vision*, pages 516–532. Springer, 2016. 1

[3] Randall Balestriero and Richard Baraniuk. Mad max: Affine spline insights into deep learning. *arXiv preprint arXiv:1805.06576*, 2018. 2

[4] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 839–847. IEEE, 2018. 1

[5] Edo Collins, Radhakrishna Achanta, and Sabine Susstrunk. Deep feature factorization for concept discovery. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 336–352, 2018. 1

[6] Rachel Lea Draelos and Lawrence Carin. Use hirescam instead of grad-cam for faithful explanations of convolutional neural networks. *arXiv e-prints*, pages arXiv–2011, 2020. 1

[7] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017. 1

[8] Kelli D Humbird, J Luc Peterson, and Ryan G McClarren. Deep neural network initialization with decision trees. *IEEE transactions on neural networks and learning systems*, 30(5):1286–1295, 2018. 1

[9] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475, 2015. 1

[10] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *International Conference on Machine Learning*, pages 2363–2372. PMLR, 2017. 1

[11] Mohammed Bany Muhammad and Mohammed Yeasin. Eigen-cam: Class activation map using principal components. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020. 1

[12] Ravi Teja Mullapudi, William R Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8080–8089, 2018. 1

[13] Calvin Murdock, Zhen Li, Howard Zhou, and Tom Duerig. Blockout: Dynamic model selection for hierarchical deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2583–2591, 2016. 1

[14] Venkatesh N Murthy, Vivek Singh, Terrence Chen, R Manmatha, and Dorin Comaniciu. Deep decision network for multi-class image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2240–2248, 2016. 1

[15] Duy T Nguyen, Kathryn E Kasmarik, and Hussein A Abbass. Towards interpretable anns: An exact transformation to multi-class multivariate decision trees. *arXiv preprint arXiv:2003.04675*, 2020. 2

[16] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. 1

[17] Anirban Roy and Sinisa Todorovic. Monocular depth estimation using neural regression forest. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5506–5514, 2016. 1

[18] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 1

[19] Ishwar Krishnan Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, 1990. 1

[20] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013. 1

[21] Agus Sudjianto, William Knauth, Rahul Singh, Zebin Yang, and Aijun Zhang. Unwrapping the black box of deep relu networks: interpretability, diagnostics, and simplification. *arXiv preprint arXiv:2011.04041*, 2020. 2

[22] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018. 1

[23] Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E Gonzalez. Nbdt: neural-backed decision trees. *arXiv preprint arXiv:2004.00221*, 2020. 1

[24] Mike Wu, Michael Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 1

[25] Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018. 1

[26] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 1

[27] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *International Journal of Computer Vision*, 126(10):1084–1102, 2018. 1

[28] Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical geometry of deep neural networks. In *International Conference on Machine Learning*, pages 5824–5832. PMLR, 2018. 2

[29] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016. 1