Decision Trees as Binary Network

Zhang Jinxiong

November 11, 2019

Abstract

Decision tree looks like a simple computational graph without cycles, where only the leaf nodes specify the output values and the non-terminals specify their test or computation. We will express decision trees in the language of computational graph. As shown, the decision tree is a shallow binay network.

1 Introduction

Computation Graph is the descriptive language of deep learning models. To create a computational graph, we make each of these operations, along with the input variables, into nodes. When the value of one node is the input to another node, an arrow goes from one to another. These sorts of graphs come up all the time in computer science, especially in talking about functional programs. They are very closely related to the notions of dependency graphs and call graphs. Theyre also the core abstraction behind the popular deep learning framework TensorFlow.

In deep neural network, the operation of non-terminals is a smooth activation function such as $\sigma(x) = \frac{1}{1+\exp(-x)}$ or 'ReLU' and it acts on every input in the elementwise sense. All inputs of the deep neural network share *the same depth and activation function* in computational graph. More complex architecture can include some 'feedback structure'.

In decision tree, the operation of non-terminal is a test function such as typical 'statistical test' and it determines the input next state - for example terminated or not. Different inputs have different depth and test functions in computational graph. The test function depends on the 'splitting criteria' when building a decision tree. In vanilla binary decision tree, the test function does not change the inputs and the final outputs of terminals depend on the labels of its instances.

In the following sections, it is shown that decision tree is a shallow binary network. Thus we extend the field of computational graph beyond deep learning and acceleration technques in deep learning may help to accelerate the tree-based learning algorithms. The classification tree or categorical feature is not discussed.

2 Computational Decision Trees

Different from building a decision tree, prediction of a decision tree is the tree traversal in nature. Inspired by QuickScorer, we split such prediction to the following stages.

The first stage is to find the false nodes in the decision tree with respect to the input x:

$$h = \frac{\text{Sign}(Sx - t) + 1}{2} = D(Sx - t) \tag{1}$$

where so-called 'selection matrix' $S \in \mathbb{R}^{n_L \times p}$ consists of one-hot row vector in $\mathbb{R}^p$ representing which feature is tested in a node; the elemets of threshold vector $t \in \mathbb{R}^{n_L}$ are the optimal points of each node associated with one feature; $Sign(\cdot)$ is the element-wise sign function and $Sign(0) = -1$;$D(x)$ is the binarized ReLU function defined by

$$D(x)_i = \begin{cases} 1, & \text{if } x_i > 0 \\ 0, & \text{otherwise} \end{cases}.$$

Here n_L is the number of the non-terminals and p is the dimension of input space. If the feature of x is greater than the splitting point, the corresponding node is a true node. Otherwise, the node is 'False' node.

The second stage is to find bitvector of false nodes

$$H = B \ \text{Diag}(h) \tag{2}$$

where $Diag(\cdot)$ maps a vector to a diagonal matrix; $B \in \mathbb{B}^{L \times n_L}$ is the bitvector[1]matrix of the decision tree. Every column of B is a bit-vector of node; the matrix B Diag(h) is the matrix multiplication product of matrix B and Diag(h). Here L is the numer of terminal nodes(leaves).

The final stage is to determine the output

$$v[i], \quad i = \arg\max(HP) \tag{3}$$

where $P = (1, 2, \cdots, n_L - 1, n_L)^T \in \mathbb{R}^{n_L}$ and n_L is the number of the non-ternimal leaves; $v[i]$ are the ith elements of vector v;$\arg\max(v)$ returns the index of the first maximum in the vector v. In fact the vector can be any positive vector such as $P = (1, 1, \cdots, 1, 1)^T \in \mathbb{R}^{n_L}$.

In a compact form, a decision tree is expressed as follows:

$$T(x) = v[\arg\max(\{B \ \text{Diag}[(D(Sx - t)]\}P)]$$

where the notation $Diag(x)$ maps a vector to a matrix; B is the bitvector matrix of the decision tree.

The hierarchical structure of decision tree is clear:

$$\begin{array}{llll} x & \to h & \to H & \to v[i] \\ \mathbb{R}^p & \to \mathbb{B}^{n_L} & \to \mathbb{B}^L & \to \mathbb{R}. \end{array}$$

[1]Every internal node n is associated with a node bitvector (of the same length), acting as a bitmask that encodes (with 0s) the set of leaves to be removed from the subtree whenever n is a false node.

It is really a shallow model. And its hidden layers are sparse binary.

Now there is nothing other than expressing the decision tree in the language of computational graph. It looks far from a step function. However, note that

- the Sign or 'binarized ReLU' function is a step function.
- the matrices S and B are binary, i.e., their elements are 0 or 1.
- $\arg\max()$ only selects one element in this case.

All 'if-then' tests are transformed to numerical computation. And the test phase and the traversal phase are split and separated. In test phase, we obtain the hidden state h dependent on the selection matrix S and threshold vector t. In traversal phase, we get the hidden state H dependent on the bitvector matrix B. All the inputs share the same computational procedure.

The tuple (S, t, B, v) really determines a decision tree.

Theorem 1 *The mapping $M : T \mapsto (S, t, B, v)$ is one-one.*

Proof 2.1 *It is only necessary to prove that it is equivalent to the theorem in QuickScorer. The key idea is that the leftmost non-zero element in the result of logical 'AND' of bitvectors is the one which is not zero in every bitvector of its false nodes.*

This also provide some hints why the binary neural networks work if you believe in the reasonability of decision trees.

3 An Example of Binary Decision Tree

Suppose the input variable x has 4 numerical features, the learned decision tree is as shown in the following figure 1. The selection matrix S and bit-vector matrix B is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

respectively. The threshold vector is $t = (1, 4, 3, 2, 5)^T$. The value vector is $v = (v_1, v_2, v_3, v_4, v_5, v_6)^T$. Suppose input vector $x = (2, 1, 2, 2)^T$, then the output is given by

$$T(x) = v[\arg\max(\{B \ \operatorname{Diag}[(D(Sx - t)]\}P)] = v_5.$$

The first element of bitvector always corresponds to the leftmost leaf in the decision tree so that the algorithm return the first value when there is no false node for some inputs such as $(1, 1, 2, 3)^T$. And the order of bitvector is left to right, top to bottom.

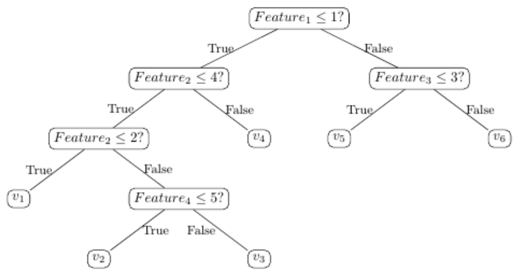


Figure 1: A Binary Decision Tree

4 Structures of Bitvector Matrix

Not all 0-1 matrix is the bitvector matrix of a decision tree. For example, the identity matrix cannot be a bitvector matrix of a decision tree because there are at most one 0 element 0f some columns of the bitvector matrix.

It is obvious that there are some inherent structures in the bitvector matrix B. There are four rules:

1. there are the most 0s in the bitvector of the root node.
2. there is only one 0 in the bitvectors of left-leaf-parent nodes.
3. the left nodes inherit the 1s from their ancestors.
4. the right nodes inherit the 0s from their parents and shift these 0s to 1s.

Here the node n is defined as left-leaf-parent node if its left child node is a terminal node/leaf. And we can infer that there is at least one 0 in the bitvector of the root combining the first rule and the second rule.

If the bitvector matrix is given as following

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

from the first rule, we know the first column corresponds to the root node; from the second rule, we know the third, forth and fifth column correspond to leaf node; from the third rule, we know the second, forth and fifth column correspond to left nodes of root; from the forth rule, we know the third column correspond to right node of root. What is more, we can infer that the second is the parent of the forth and fifth because they share the same last three 1s; the forth is the parent of the fifth because the fifth inhert the last four 1s from the forth.

The number of columns m is less than the number of rows in bitvector matrix n. For above example, $n - m = 1$. In another word, the number of nonterminal nodes is less than the terminal nodes. In fact, this difference is always equal to 1 for all binary decision trees. Additionally, we will know the sizes of S, t, B, v if the input sample diemension p is given.

Given the number of internal nodes, can we find all the possible structure of decision trees? Given a bitvetor matrix B, can we find the selection matrix S and threshold vector t?

If the answers to above questions are all 'Yes', we only need to specify the number of internal nodes and then all the structure of decision tree can be found by a greedy way.

What is the relation of this matrix and the methods to train a decision tree?

5 Optimization of Decision Trees

Even we can find all the structure of the decision trees given the number of internal nodes, there are still selection matrix S, threshold vector t and value vector v to optimize.

XNOR-net shows some methods to training binary-weights-networks. Coarse Gradient Descent Method is designed for learning sparse weight binarized activation neural networks. Blended coarse gradient descent is designed for full quantization of deep neural networks. Can we apply these methods to build decision trees instead of heuristic methods?

Mohammad Norouzi et al formulate a convex-concave upper bound on the trees empirical loss and use stochastic gradient descent to train decision trees. Dimitris Bertsimas and Jack Dunn present Optimal Classification Trees, a novel formulation of the decision tree problem using modern Mixed-Integer Optimization (MIO) techniques. Dimitris Bertsimas and Romy Shioda introduce mixed-integer optimization methods to the classical statistical problems of classification and regression. And these methods can find optimal linear-combination (oblique) splits[2]for decision trees.

Different from our representation of decisin tree, Mohammad Norouzi et al introduce a tree navigation function rather than the bitvector matrix B. And Optimal Classification Trees with Hyperplanes (OCT-H) model encodes the information of structure into MIO formulation.

[2]In our representation, linear-combination (oblique) splits correspond to real rows of the selection matrix S. In another word, the selection matrix S can be any matrix in $\mathbb{R}^{N_L \times p}$ for oblique decision trees.

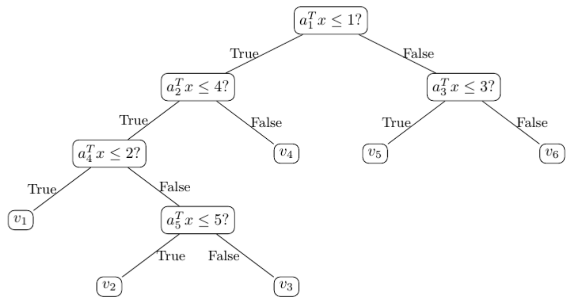
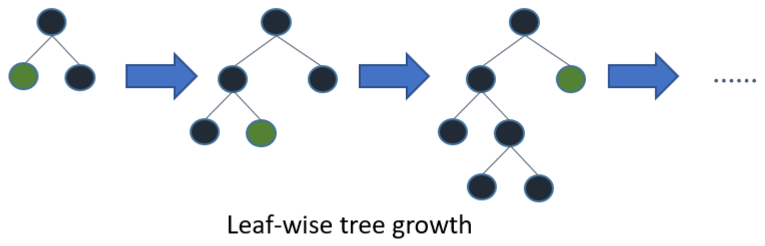


Figure 2: A Toy Example of Oblique Decision Tree

Our aim is to build a decision tree in the case where only the number of internal nodes or leaves is specified. So that the structure and splitting points are learnt from the input samples. In another word, the representation tuple (S, t, B, v) is learnt from the input samples. The difficult is the restriction of bitvector matrix. The bitvector matrix B is not only '0-1' valued but also subject to the four rules in above section. A basic idea is to limit the bitvector matrix B to some special form such as oblivious decision trees. Another idea is to find the whole space that the four rules describe. It is clear that the space governed by the four rules is not a linear space so that it is not possible to find the basises to generate all the bitvector matrices. What is worse, it is not convex.

A leaf-wise tree growth is proposed in LightGBM.



It inspires us to optimize only one node in each step in our framework.

First, we initialize a non-zero bitvector of the root node according to the first rule and the other bitvectors are zero vectors as in the coordinate optimization method. In this step, we only can split the data to the left or the right of the root node. And then we can optimze the first column vector of bitvector matrix B, the first row of selection matrix S and the first element of the threshold vector t by minimizing the 'splitting criteria' as ordinary decision tree growth. For example, if the number of terminal nodes is 6 and input samples have 4 numerical feartures, the first step is given by

$$\arg\min_{B,S,t,v} \sum_{i=1}^{N} \ell(y_i, T(x_i)) \tag{4}$$

where $\ell(\cdot,\cdot)$ is the loss function; $\{(x_i, y_i) \mid i = 1, 2, \cdots, N\}$ is the training dataset;

$$T(x) = v[\arg\max(\{B \ \ \mathrm{Diag}[(D(Sx - t)]\}P)];$$

and

$$S = \begin{pmatrix} s_1 & s_2 & s_3 & s_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ b_2 & 0 & 0 & 0 & 0 \\ b_3 & 0 & 0 & 0 & 0 \\ b_4 & 0 & 0 & 0 & 0 \\ b_5 & 0 & 0 & 0 & 0 \\ b_6 & 0 & 0 & 0 & 0 \end{pmatrix}, t = \begin{pmatrix} t_1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{pmatrix}.$$

Here $s_1, \cdots, s_4, t_1, v_1, \cdots, v_6 \in \mathbb{R}$ and $b_2, \cdots, b_6 \in \{0, 1\}$. Note that the first element of bitvector always corresponds to the leftmost leaf in the decision tree so that there is one 0 in the first column of the matrix B.

Second, we fix the optimal bitvector of root node and initialize the second column of the bivector according to four rules, which correspond to the first left node of the root node. Other bitvectors are set to be zeroes. At the same time, we optimze the second row of selection matrix S and the second element of the threshold vector t by minimizing the 'splitting criteria'. In this step, the data split into the left nodes of root can be split into the right/left nodes of second node. We continue with the example in the first step. It is still given by (4) but we fix the optimal parameters $s_1^*, \cdots, s_4^*, b_2^*, \cdots, b_6^*, t_1^*$ returned in the first step

$$S = \begin{pmatrix} s_1^* & s_2^* & s_3^* & s_4^* \\ s_1 & s_2 & s_3 & s_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & b_1 & 0 & 0 & 0 \\ b_2^* & b_2 & 0 & 0 & 0 \\ b_3^* & b_3 & 0 & 0 & 0 \\ b_4^* & b_4 & 0 & 0 & 0 \\ b_5^* & b_5 & 0 & 0 & 0 \\ b_6^* & b_6 & 0 & 0 & 0 \end{pmatrix}, t = \begin{pmatrix} t_1^* \\ t_2 \\ 0 \\ 0 \\ 0 \end{pmatrix}, v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{pmatrix}.$$

It is a greedy way to test all legal inheritors of the root node $(0, b_2^*, b_3^*, b_4^*, b_5^*, b_6^*)^T$ according to the four rules. We need a more efficient method to solve the problem (4).

Other steps are similar to the second step untill all the parameters are found.

It is mentioned 'as only one-step optimal and not overall optimal'. It is 'block coordinate descent' similar to the layer-wise training. Global sublinear rate of convergence of block coordinate descent type method is established. Such tree growth method is a modification of the top-down growth method.

If the matrix B is not a bitvector matrix of decision tree, does this expression make some sense?