Decision Trees as Binary Network

Zhang Jinxiong

November 9, 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, 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 a leaf 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.

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:

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

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 $Sign(0) = -1$. Here n_L is the number of the non-terminals; $D(x)$ is the binarized ReLU function defined by

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

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

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

where $Diag(x)$ 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

$$v[i], \quad i = \arg\max(HP)$$

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 value 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 transform to numerical computation. And we split the decision tree into two phases: the test phase and the traversal phase. 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.

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

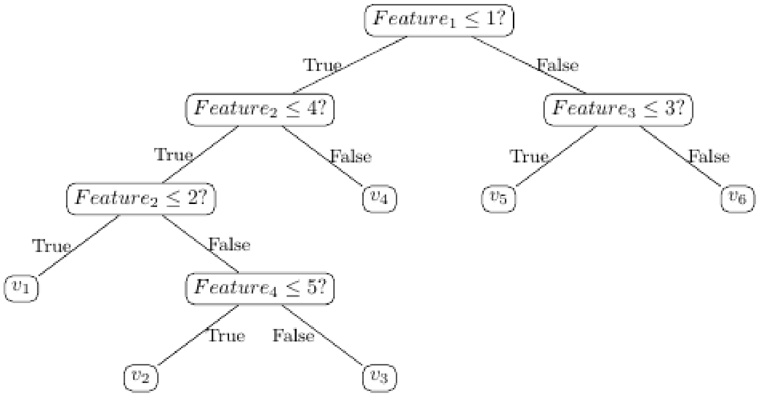
Suppose the input variable x has 4 numerical features, the learned decision tree is as following. The selection matrix S and bit-vector matrix B is

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

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

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

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.



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:

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 ancestors 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.

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. Is this difference always equal to 1 for all the binary decision trees? Given the number of internal nodes and the number of leaves, 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? If the matrix B is not a bitvector matrix of decision tree, does this expression make some sense?

5 New Tree Building Methods

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

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?