Decision Trees in Computational Graph

Zhang Jinxiong

November 4, 2019

Abstract

Decision tree looks like a simple graph without loops, where only the leaf nodes specify the output values and the non-terminals specify their test or computation. Computational graph is regarded as an extension of deep neural network. We will express decision tree in the language of computational graph.

1 Computational Graph

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 transform 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 function 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 its instances' labels.

2 A Unified Framework

Different from building the tree, prediction of decision tree is a 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:

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

where so-called 'selection matrix' $S \in R^{n_L \times p}$ consists of one-hot row vector in R^p representing which feature is tested in a node; the bias vector $t \in R^{n_L}$ 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. 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

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

where $Diag(x)$ maps a vector to a diagonal matrix; $B \in B^{L \times n_L}$ is the bitvector matrix of the decision tree. Every columon 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 exit leaf nodes.

The final stage is to determine the output

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

where $P = (1, 2, \cdots, n_L - 1, n_L)^T \in R^{n_L}$ 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 a compact form, a decision tree is expressed as follows:

$$T(x) = v[\arg\max([(B \ \mathrm{Diag}[\frac{(Sign[Sx - t]) + 1}{2}])]P)]$$

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:

$$\begin{array}{llll} x & \to h \to H & \to v[i] \\ R^p & \to B^{n_L} \to B^L & \to R \end{array} \tag{1}$$

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

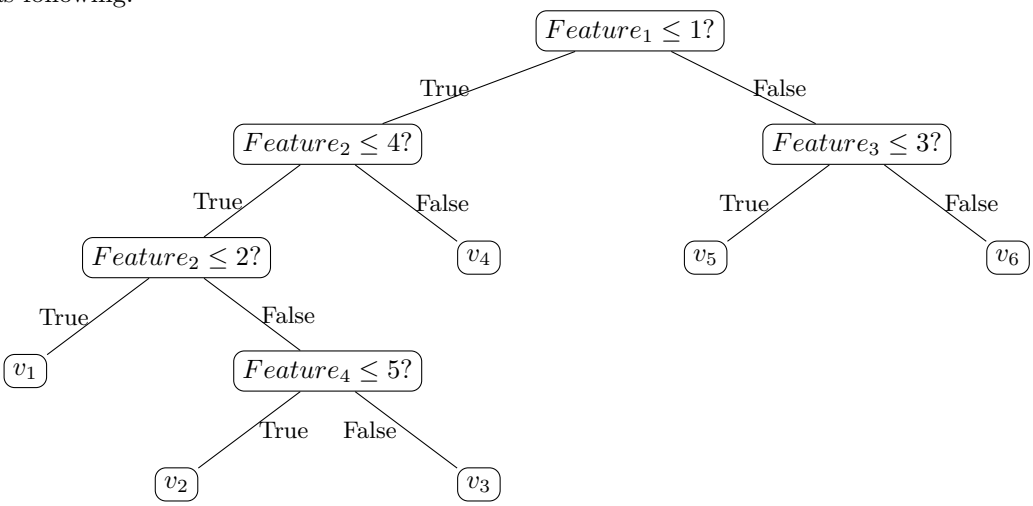
All 'if-then' tests transform to numerical computation.

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.*

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 bias 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

$$T(x) = T(x) = v[\arg\max([(B \ \text{Diag}[\frac{(Sign[Sx - t]) + 1}{2}])]P)] = v_2.$$

4 Beyond Binary Decision Tree

In mathematical consideration, can we generalize the selection matrix S to real matrices? Can we replace the $Sign$ function with some smooth function? Can we generalize the bit-vector matrix B to real matrices? Can we describe the boosted decision tree in computational graph? Can we apply gradient-based methods to train a decision tree?

4.1 Oblique Decision Trees

Oblique Decision Trees extends the selection (binary) matrix S to real matrix in nature.

4.2 Probabilistic Decision Trees

Probabilistic Decision Trees replace the sign function with some smooth function.

4.3 Boosted Decision Tree

Boosted decision tree or multiple additive regression tree is the sum of successive decision tree

$$T(x) = \sum_{n=1}^{N} T_n(x)$$

where T_n relies on the outputs of its 'parent tree' T_{n-1} for $n = 2, 3, \cdots, N$.

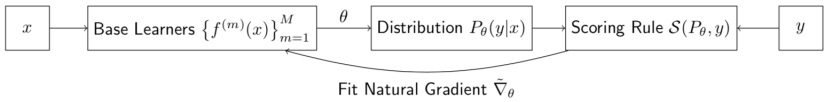
Algorithm 1 Gradient Boost Decision Trees

Input training data set $\{(x_n, y_n) \mid x_i \in \mathrm{X} \subset R^n, y_i \in R, n = 1, 2, \cdots, N\}$
for $t = 1, 2, \ldots, T$ **do**
  for $i = 1, 2, \ldots, n$ **do**
    compute $r_{i,t} = -[\frac{\partial L(\mathrm{y}_i, f(x_i))}{\partial f(x_i)} \mid_{f=f_{t-1}}]$.
  end for
  Fit a regression tree to the targets $r_{i,t}$ giving terminal regions

$$R_{j,m}, j = 1, 2, \ldots, J_m.$$

  for $j = 1, 2, \ldots, J_m$ **do**
    compute $\gamma_{j,t} = \arg\min_\gamma \sum_{x_i \in R_{j,m}} L(\mathrm{y}_i, f_{t-1}(x_i) + \gamma)$.
  end for
  Update $f_t = f_{t-1} + \nu \sum_{j=1}^{J_m} \gamma_{j,t} I(x \in R_{j,m}), \nu \in (0, 1)$
end for
Output $f_T(x)$.

For example, NGBoost has three abstract modular components as following



where the natural gradient as feedback signal is uased to train another base learner.

Compared to deep neural network, boosted decision trees is like a forwards neural network where each decision tree is like a layer in deep neural network and sends the gradients to next layer.

See Gradient Guide Decision Tree for a numerical example.

Algorithm 2 Gradient Guide Decision Tree

Input training data set $\{(x_n, y_n) \mid x_i \in \mathrm{X} \subset R^n, y_i \in R, n = 1, 2, \cdots, N\}$
for $t = 1, 2, \ldots, T$ **do**
  for $i = 1, 2, \ldots, n$ **do**
    compute $\alpha_{i,t} = \arg\min_\alpha L(y_i, f(x_i) - \alpha[\frac{\partial L(\mathrm{y}_i, f(x_i))}{\partial f(x_i)} \mid_{f=f_{t-1}}])$.
  end for
  Fit a regression tree to the targets $r_{i,t} = f(x_i) - \alpha_{i,t}[\frac{\partial L(\mathrm{y}_i, f(x_i))}{\partial f(x_i)} \mid_{f=f_{t-1}]}$ giving terminal regions

$$R_{j,m}, j = 1, 2, \ldots, J_m.$$

  for $j = 1, 2, \ldots, J_m$ **do**
    compute $\gamma_{j,t} = \arg\min_\gamma \sum_{x_i \in R_{j,m}} L(\mathrm{y}_i, \gamma)$.
  end for
  Update $f_t = \sum_{j=1}^{J_m} \gamma_{j,t} I(x \in R_{j,m})$
end for
Output $f_T(x)$.