

Decision Trees do not Generalize to New Variations

Yoshua Bengio, Olivier Delalleau and Clarence Simard

Dept. IRO, Université de Montréal

C.P. 6128, Montreal, Qc, H3C 3J7, Canada

{*bengioy,delallea,simardcl*}@*iro.umontreal.ca*

<http://www.iro.umontreal.ca/~lisa>

Technical Report 1304

June 8th, 2007

Abstract

The family of decision tree learning algorithms is among the most widespread and studied. Motivated by the desire to develop learning algorithms that can generalize when learning highly variable functions such as those presumably needed to achieve artificial intelligence, we study some theoretical limitations of decision trees. We demonstrate that they can be seriously hurt by the curse of dimensionality, but most importantly, that they cannot generalize to variations not seen in the training set. This is because a decision tree creates a partition of the input space and needs at least one example in each of the regions associated with a leaf in order to make a sensible prediction in that region. A better understanding of the fundamental reasons for this limitation suggests that one should use *forests* instead of trees, which provide a form of distributed representation and can generalize to variations not encountered in the training data.

1 Introduction

A long-term goal of machine learning research that remains elusive is to produce methods that will enable artificially intelligent agents. Examples of such complex behaviors are found in visual perception, auditory perception, and natural language processing. In all of these cases, it seems plausible to assume that the required learning algorithms will have to learn functions with a large number of variations. These correspond to many possible combinations of values for the different factors of variation that underlie the unknown generating process of interest. A better understanding of the limitations of current algorithms can serve as a guide in moving statistical machine learning towards artificial intelligence.

Inspired by previous work on the limitations of kernel methods with a local kernel [2, 3], this paper focuses on similar limitations of decision trees. Decision trees

were introduced in [5]: they work by recursively partitioning the input space and assigning an appropriate answer for each of the resulting input region. Each node of the tree corresponds to a region of the input space and the root is associated with the whole input space.

In this paper, we study fundamental theoretical limitations of decision trees concerning their inability to generalize to variations not seen in the training set. The basic argument is that we need a separate leaf node to properly model each such variation, and at least one training example for each leaf node. Our theoretical analysis is in line with previous empirical results [16, 17] showing that the generalization performance of decision trees degrades when the amount of variations in the target function increases. Our proofs focus on the most common form of decision tree, with axis-aligned partitions of the input space at internal nodes and a constant prediction associated with each leaf.

2 Background Material

Internal nodes of the tree are associated with a decision function that splits that region into sub-regions, each one associated with a child node. Leaf nodes are associated with a function (usually a constant function) that computes the prediction of the tree when the input example falls in the region associated with the leaf. Because the number of possible decision trees is exponential in the size of the tree, the trees are grown greedily, and the size of the tree is selected based on the data, e.g. using cross-validation. A decision tree learning algorithm is thus non-parametric, constructing a more flexible function when more training examples are available. In many implementations (e.g. [5]) the internal node decision function depends on a single input variable (we call it *axis-aligned*), and for a continuous variable it just splits the space by selecting a threshold value (thus giving rise to a binary tree). It is also possible to use multivariate decision functions, such as a linear classifier, as in [14].

Definition 2.1 (n-ary split function).

Let t_i be an internal node and $\mathcal{T}_i = \{t_{i_0}, \dots, t_{i_{n-1}}\}$ the set of t_i 's children nodes. The n -ary split function S_i of node t_i is a function defined on $R_i \subseteq \mathbb{R}^d$, the region associated with the node t_i , which splits R_i in n nonempty subsets $\{R_{i_0}, \dots, R_{i_{n-1}}\}$, the regions associated with children nodes t_{i_k} . The subsets must be disjoint and $R_i = \cup_{k=0}^{n-1} R_{i_k}$. Formally, S_i is a n -ary split if it can be written $S_i(x) = \sum_{k=0}^{n-1} k \mathbb{1}_{x \in R_{i_k}}$, i.e. it gives the index of the child node containing x , in $\{0, 1, \dots, n-1\}$.

Definition 2.2 (Decision Tree).

A decision tree is the function $T : \mathbb{R}^d \rightarrow \mathbb{R}$ resulting from a learning algorithm applied on training data, which always has the following form:

$$T(x) = \sum_{i \in \text{leaves}} g_i(x) \mathbb{1}_{x \in R_i} = \sum_{i \in \text{leaves}} g_i(x) \prod_{a \in \text{ancestors}(i)} \mathbb{1}_{S_a(x) = c_{i,a}}$$

where $R_i \subset \mathbb{R}^d$ is the region associated with leaf i of the tree, $\text{ancestors}(i)$ is the set of ancestors of leaf node i , $c_{i,a}$ is the index of the child of node a on the path from

a to leaf i , and S_a is the n -ary split function at node a . $g_i(\cdot)$ is the decision function associated to leaf i and is learned only from training examples in R_i . Note that exactly one term of the sum in $T(x)$ can be non-zero (associated with the leaf in which x falls). Learning algorithms for decision trees grow the tree by adding and removing nodes, in such a way that every node has at least one example falling in it (i.e. no R_i is empty).

Definition 2.3 (Piecewise Constant).

We say function $f : R \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$ is piecewise constant if it is of the form $f(x) = \sum_{i=1}^N g_i \mathbb{1}_{x \in R_i}$ for some finite N , where $g_i \in \mathbb{R}$ and the R_i 's are disjoint connected subsets of \mathbb{R}^d .

Definition 2.4 (Piecewise Constant with N Pieces).

We say that function f is piecewise constant with N pieces if it is piecewise constant and it cannot be represented with less than N pieces.

Definition 2.5 (Constant-Leaves Decision Tree).

A constant-leaves decision tree is a decision tree as in definition 2.2 such that for each leaf node i , the decision function $g_i(x)$ is constant, i.e. $\forall x \in R_i, g_i(x) = g_i \in \mathbb{R}$. If it has N leaf nodes, it can thus be written $T_N(x) = \sum_{i=1}^N g_i \mathbb{1}_{x \in R_i}$ and it is a piecewise constant function with at most N pieces.

Definition 2.6 (Approximation and error).

We say a function f approximates a function g with error ϵ if $\sup_x |f(x) - g(x)|$ is smaller than ϵ .

Definition 2.7 (ϵ -variation).

We say that a function f has n ϵ -variations if it takes at least n constant pieces for a piecewise constant function to approximate f with an error at most ϵ over the domain of f .

3 Inability to Generalize to New Variations

Lemma 3.1. Let \mathcal{F} be the set of piece-wise constant functions. Consider a target function $h : D \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$. For a given representation error level ϵ , let N be the minimum number of constant pieces required to approximate with a function in \mathcal{F} the target function h with an error less than ϵ . Then to train a constant-leaves decision tree with error less than ϵ one requires at least N training examples.

Proof. Let us consider a constant-leaves decision tree approximating the target function with an error less than ϵ , and ℓ be the number of leaves in the tree. Since it is piecewise constant with at most ℓ pieces, and any piecewise constant function with an error less than ϵ must be piecewise constant with at least N pieces, then $\ell \geq N$. From definition 2.2, each leaf node contains at least one example, and thus in order to have ℓ leaves, the tree requires at least $\ell \geq N$ examples. \square

Since one can easily form an exponential number of distinct regions in \mathbb{R}^d by taking cross-products of one-dimensional partitions, it should now appear clearly that the

number of examples required to train a constant-leaves decision tree can grow exponentially with the dimension of the input space \mathbb{R}^d . To illustrate this phenomenon concretely, we prove such statements in the special cases of two classes of functions: the parity function and the checkerboard functions (defined below). What is important to note here, is that these functions can otherwise be represented compactly, suggesting that some rather generic learning algorithms could learn them. This is justified by the fact that the Kolmogorov complexity [13] of these functions would be very low in any current human language or programming language. Functions with low Kolmogorov complexity can theoretically be learned with few examples (although computational complexity is another question, since we need an efficient way to search in that space).

3.1 Curse of Dimensionality on the Parity Task

Definition 3.2 (d-bit Parity Task). *The d-bit parity task has as target function the d-bit parity function $p : \{0, 1\}^d \subset \mathbb{R}^d \rightarrow \{-1, 1\}$,*

$$p(x) = \begin{cases} -1 & \text{if } \sum_{i=1}^d x_i \text{ is even} \\ 1 & \text{if } \sum_{i=1}^d x_i \text{ is odd.} \end{cases}$$

As a learning task it has a total of 2^d possible examples, each sampled with equal probability. On this task, we thus define the number of generalization errors of a predictor g as $|\{x \in \{0, 1\}^d : g(x) \neq p(x)\}|$ and its generalization error (or error rate) as $|\{x \in \{0, 1\}^d : g(x) \neq p(x)\}|/2^d$. Similarly we can talk of the generalization error of a node of a decision tree as the average error among the examples falling into that node.

Definition 3.3 (Constant-leaves decision tree with axis-aligned decision nodes). *A constant-leaves decision tree with axis-aligned decision nodes is a constant-leaves decision tree whose decision split at internal node i is of the form $S_i(x) = \mathbb{1}_{x_j < \alpha_i}$, i.e. it splits the current region in two, based on the comparison between the j -th coordinate and a single threshold α_i .*

Lemma 3.4. *On the task of learning the d-bit parity function, a constant-leaves decision tree with axis-aligned decision nodes and output in $\{-1, 1\}$ will have a generalization error of $\frac{1}{2}$ on leaf nodes of depth less than d .*

Proof. Let us prove the result by induction on $d \geq 1$, for both the tasks of learning the parity function and its opposite $-p(\cdot)$. For $d = 1$, it is obvious since the only leaf node of depth less than d can be the root node, which contains all 2 possible examples. Let us suppose the result is true for $d = k \geq 1$, and let us consider the case $d = k + 1$. Since no node can be empty, the split function at the root node r must have a threshold $\alpha_r \in (0, 1)$. Without loss of generality, suppose the split is performed on the first input coordinate. The two sub-regions thus defined are $R = \{x \in \{0, 1\}^{k+1} | x_1 = 0\}$ and $R' = \{x \in \{0, 1\}^{k+1} | x_1 = 1\}$. Since the first coordinate is constant in both R and R' , the corresponding subtrees cannot perform additional splitting w.r.t. this coordinate (as this would result in empty nodes), and x_1 can be ignored. If the original task was to learn the parity task this implies the subtree trained on R tries to learn the parity task

in dimension k , while the subtree trained on R' tries to learn the opposite of the parity task in dimension k (because the target is switched on R' , due to $x_1 = 1$). From the induction hypothesis, all leaf nodes of depth less than k in these subtrees (i.e. of depth less than $k + 1 = d$ in the full tree) have a generalization error of $\frac{1}{2}$. If the original task was to learn the opposite of the parity task, the same reasoning applies (switching the roles of R and R'). \square

Theorem 3.5. *On the d -bit parity task, a constant-leaves decision tree with axis-aligned decision nodes trained on n different examples has a generalization error in $[\frac{1}{2} - \frac{n}{2^{d+1}}, 1 - \frac{n}{2^d}]$.*

Proof. Let k the number of leaf nodes of depth d in a tree trained on n different examples. We will first show that the generalization error is $\epsilon = \frac{1}{2} - \frac{k}{2^{d+1}}$. On a leaf of depth d , there can be only one training example (because every ancestor of the leaf splits on a different input and divides the input space in 2, and there are only 2^d possible examples). Hence training and generalization error on the k depth d leaves is 0. On the other hand, Lemma 3.4 shows that the generalization error is $1/2$ on the other leaves. Since there are k examples falling in depth d leaves, and $2^d - k$ examples falling in the others, the total number of generalization errors is $\frac{1}{2}(2^d - k)$ and the error rate is $\epsilon = \frac{1}{2} - \frac{k}{2^{d+1}}$.

To prove our theorem we now have to find lower and upper bounds for k . Clearly, $k \leq n$, otherwise we could have more leaf nodes of depth d than examples. This proves the first inequality: $\epsilon \geq \frac{1}{2} - \frac{n}{2^{d+1}}$. Moreover, the worst we can do – in terms of generalization error – before inserting a first leaf node of depth d is to create all the leaf nodes of depth $d - 1$ and this requires at least 2^{d-1} examples. Then each additional example will lead to a node of depth $d - 1$ being split, creating two nodes of depth d , so that $k \geq 2(n - 2^{d-1})$. Hence $\epsilon \leq \frac{1}{2} - \frac{2(n - 2^{d-1})}{2^{d+1}} = \frac{1}{2} - \frac{2n}{2^{d+1}} + \frac{2^d}{2^{d+1}} = 1 - \frac{n}{2^d}$. \square

Corollary 3.6. *On the task of learning the d -bit parity function, a constant-leaves decision tree with axis-aligned decision nodes will require at least $2^d(1 - 2\epsilon)$ examples in order to achieve a generalization error less than or equal to ϵ .*

Proof. Let $n(\epsilon)$ be the number of examples to get a generalization error ϵ . Directly from the lower bound of theorem 3.5 we find $n(\epsilon) \geq 2^d(1 - 2\epsilon)$. \square

3.2 Curse of Dimensionality for the Checkerboard Task

Definition 3.7 (Checkerboard Task). *A Checkerboard task over $[0, 1]^d$ with minimum variation δ , minimum mass m per board cell and interval numbers $\{n_i\}$ has a target function $f : [0, 1]^d \rightarrow \mathbb{R}$ of the form $f(x) = g(v_1(x_1), v_2(x_2), \dots, v_d(x_d))$ where each $v_i : \mathbb{R} \rightarrow \mathbb{R}$ is based on n_i intervals $R_{i,1}, \dots, R_{i,n_i}$ such that $R_{i,j} = [\alpha_{i,j}, \alpha_{i,j+1})$ with $\alpha_{i,1} = 0$, $\alpha_{i,n_i+1} = 1$, and $\alpha_{i,j} < \alpha_{i,j+1}$. These intervals thus form a contiguous partition of $[0, 1)$, and v_i is such that it associates a particular value $a_{i,j}$ to each interval:*

$$v_i(x) = \sum_{j=1}^{n_i} a_{i,j} \mathbb{1}_{x \in R_{i,j}}.$$

Hence for each cell $C_{i_1, \dots, i_d} \stackrel{\text{def}}{=} R_{1, i_1} \times \dots \times R_{d, i_d}$ the function f assigns to every $x \in C_{i_1, \dots, i_d}$ the constant value $f(x) = g(v_1(x_1), \dots, v_d(x_d)) = g(a_{1, i_1}, \dots, a_{d, i_d}) \stackrel{\text{def}}{=} w_{i_1, \dots, i_d}$.

Furthermore, to ensure that each rectangle of the board is sufficiently different from any of its neighbors, the following constraint is satisfied:

$$|w_{i_1, i_2, \dots, i_d} - w_{k_1, k_2, \dots, k_d}| \geq \delta$$

for every (i_1, i_2, \dots, i_d) and (k_1, k_2, \dots, k_d) such that cells C_{i_1, i_2, \dots, i_d} and C_{k_1, k_2, \dots, k_d} share a common boundary. To form a data set, the inputs x are sampled with a probability distribution p such that $P(x \in C_{i_1, i_2, \dots, i_d}) = \int_{x \in C_{i_1, i_2, \dots, i_d}} p(x) dx \geq m$ for every cell C_{i_1, i_2, \dots, i_d} and p is uniform within each cell. The generalization error of a predictor h on a checkerboard task is measured as the average squared error $\int (h(x) - f(x))^2 p(x) dx$.

Proposition 3.8. *To obtain an average generalization error less than $\frac{m\delta^2}{2}$ on a checkerboard task over $[0, 1]^d$ with minimum mass m per cell, minimum variation δ and interval numbers $\{n_i\}_{i=1}^d$, using a constant-leaves decision tree with axis-aligned decision nodes, the tree must be trained with at least $N = \prod_{i=1}^d n_i$ different examples.*

Proof. We will prove that such a tree must have at least N leaf nodes, which by definition 2.2 of a decision tree implies it has been trained with at least N examples.

We first prove that we can restrict ourselves to decision trees whose splitting functions at each node are of the form

$$S(x) = \mathbb{1}_{x_i < \alpha_{i,j}} \tag{1}$$

i.e. whose splitting thresholds on dimension i are constrained to be among the interval boundaries $\alpha_{i,1}, \dots, \alpha_{i, n_i+1}$. To show this, let us consider any constant-leaves decision tree with axis-aligned decision nodes, and let us show its thresholds can be modified to verify constraint (1) without adding nodes nor increasing its generalization error on the checkerboard task. Let S be the splitting function of the highest depth node that does not verify (1), i.e. is of the form:

$$S(x) = \mathbb{1}_{x_i < \gamma} \tag{2}$$

where $\gamma \in (0, 1)$ (otherwise some node would contain no example) and such that $\forall j = 1, \dots, n_i + 1$ we have $\gamma \neq \alpha_{i,j}$. Since $\alpha_{i,1} = 0$, $\alpha_{i, n_i+1} = 1$ and the $\alpha_{i,j}$ are increasing with j , there must exist $j \in \{1, \dots, n_i\}$ such that $\gamma \in (\alpha_{i,j}, \alpha_{i, j+1})$. Now consider, among all nodes in the path from the root of the tree to the parent of the node we are considering, those that also make a split based on the same variable x_i , and define \mathcal{T} the set of all their split thresholds. We will focus on the interval defined by the following two real numbers:

$$\begin{aligned} \lambda &= \max(\alpha_{i,j}, \max(\beta \in \mathcal{T} | \beta < \gamma)) \\ \mu &= \min(\alpha_{i, j+1}, \min(\beta \in \mathcal{T} | \beta > \gamma)) \end{aligned}$$

where we take the minimum of an empty set to be $+\infty$ and its maximum to be $-\infty$ (note also that $\gamma \notin \mathcal{T}$ because a tree does not split twice on the same variable with the same threshold in the same branch, otherwise one would get 0 training examples in a node). From their definition, λ and μ verify the following inequality:

$$\alpha_{i,j} \leq \lambda < \gamma < \mu \leq \alpha_{i,j+1}. \quad (3)$$

Moreover, for fixed $x_j, j \neq i$, when x_i varies in $[\lambda, \gamma)$ or in $[\gamma, \mu)$ the output of the decision tree does not change, since there exists no split w.r.t. coordinate x_i with a threshold in the interior of these intervals (remember that the node we are considering is the deepest one not verifying (1), and thus all its child nodes that may split w.r.t. x_i have a threshold in $(0, \alpha_{i,j}]$ or $[\alpha_{i,j+1}, 1)$).

Let h be the output function of the tree, and E_0 its average error on $D_0 = \{x \in [0, 1]^d | x_i \in [\lambda, \gamma)\}$:

$$E_0 = \frac{1}{\int_{x \in D_0} p(x) dx} \int_{x \in D_0} (h(x) - f(x))^2 p(x) dx$$

$$\stackrel{\text{def}}{=} \frac{1}{p_0} J(D_0)$$

Similarly, define $E_1 = \frac{1}{p_1} J(D_1)$ its average error on $D_1 = \{x \in [0, 1]^d | x_i \in [\gamma, \mu)\}$. Since D_0 and D_1 are disjoint, the overall generalization error of h can be written

$$E = J(D_0) + J(D_1) + J([0, 1]^d \setminus (D_0 \cup D_1))$$

$$= p_0 E_0 + p_1 E_1 + J([0, 1]^d \setminus (D_0 \cup D_1)). \quad (4)$$

Let us first consider the case $E_0 \leq E_1$. Let h' the output function that would result from replacing threshold γ in (2) by μ , and $D' = \{x \in [0, 1]^d | x_i \in [\lambda, \mu)\}$ (note that $D' = D_0 \cup D_1$). Denoting by $J'(A)$ the error $\int_{x \in A} (h'(x) - f(x))^2 p(x) dx$, the generalization error E' of h' can be written:

$$E' = J'(D_0) + J'(D_1) + J'([0, 1]^d \setminus (D_0 \cup D_1)). \quad (5)$$

For $x \in [0, 1]^d \setminus D_1$, we have $h'(x) = h(x)$ since for such a x , $x_i < \gamma \Leftrightarrow x_i < \mu$ and $x_i \geq \gamma \Leftrightarrow x_i \geq \mu$. Thus, using (4) and (5), the difference between generalization errors E and E' reduces to

$$E - E' = p_1 E_1 - J'(D_1). \quad (6)$$

Defining $\tilde{x} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d) \in [0, 1]^{d-1}$, $J'(D_1)$ can be written

$$J'(D_1) = \int_{x \in D_1} (h'(x) - f(x))^2 p(x) dx$$

$$= \int_{\tilde{x} \in [0, 1]^{d-1}} \left(\int_{x_i \in [\gamma, \mu)} (h'(x) - f(x))^2 p(x) dx_i \right) d\tilde{x}. \quad (7)$$

In order to simplify (7) we observe that, for a fixed \tilde{x} , $h'(x)$ is constant w.r.t. $x_i \in [\gamma, \mu)$ because there is no node in the tree performing a split on x_i with a threshold within

this interval. Let $y \in [0, 1]^d$ the point defined by $y_j = x_j$ for all $j \neq i$, and $y_i = \lambda$. Then $h'(x) = h(y)$, because:

- on the path from the root to the parent of the node we are considering, all splits w.r.t. the i -th coordinate return the same result for all values in $[\lambda, \mu]$ (this is a direct consequence of the definition of λ and μ)
- the split for the node we are considering returns 0 when evaluating $h'(x)$ because $x_i < \mu$, and 0 when evaluating $h(y)$ because $y_i < \gamma$
- the splits on the i -th coordinate for its child nodes can only involve thresholds equal to some $\alpha_{i,j}$, and thus return the same results for all values in $[\lambda, \mu]$ due to (3).

Similarly, we can compute $J(D_0)$ by

$$\begin{aligned} J(D_0) &= \int_{x \in D_0} (h(x) - f(x))^2 p(x) dx \\ &= \int_{\tilde{x} \in [0,1]^{d-1}} \left(\int_{x_i \in [\lambda, \gamma]} (h(x) - f(x))^2 p(x) dx_i \right) d\tilde{x}. \end{aligned} \quad (8)$$

For the same fixed \tilde{x} as above, $h(x)$ is constant w.r.t. $x_i \in [\lambda, \gamma]$ since there is no node in the tree performing a split on x_i with a threshold within this interval. Thus $h(x) = h(y)$ (with y defined as above, i.e. with its i -th coordinate set to λ). Finally, we observe that for a fixed \tilde{x} , both $p(x)$ and $f(x)$ are also constant w.r.t. $x_i \in [\lambda, \mu]$. Indeed, let C be the cell containing x when $x_i = \lambda \geq \alpha_{i,j}$: as x_i increases, x stays in the same cell as long as $x_i < \alpha_{i,j+1}$, which is true since $\mu < \alpha_{i,j+1}$. The probability distribution being uniform within cell C , $p(x)$ is thus constant and we denote it by $\tilde{p}(\tilde{x})$. Moreover, $f(x)$ is also constant since x stays in the same cell. We denote its value by $\tilde{f}(\tilde{x})$. All these observations allow us to rewrite (7) and (8) into

$$\begin{aligned} J'(D_1) &= \int_{\tilde{x} \in [0,1]^{d-1}} (\mu - \gamma)(h(y) - \tilde{f}(\tilde{x}))^2 \tilde{p}(\tilde{x}) d\tilde{x} \\ J(D_0) &= \int_{\tilde{x} \in [0,1]^{d-1}} (\gamma - \lambda)(h(y) - \tilde{f}(\tilde{x}))^2 \tilde{p}(\tilde{x}) d\tilde{x} \end{aligned}$$

and consequently

$$J'(D_1) = \frac{\mu - \gamma}{\gamma - \lambda} J(D_0) = \frac{\mu - \gamma}{\gamma - \lambda} p_0 E_0. \quad (9)$$

To conclude, we need to express the above ratio in terms of p_0 and p_1 . Using the same notations:

$$\begin{aligned} p_0 &= \int_{\tilde{x} \in [0,1]^{d-1}} \left(\int_{x_i \in [\lambda, \gamma]} \tilde{p}(\tilde{x}) dx_i \right) d\tilde{x} = \int_{\tilde{x} \in [0,1]^{d-1}} (\gamma - \lambda) \tilde{p}(\tilde{x}) d\tilde{x} \\ p_1 &= \int_{\tilde{x} \in [0,1]^{d-1}} \left(\int_{x_i \in [\gamma, \mu]} \tilde{p}(\tilde{x}) dx_i \right) d\tilde{x} = \int_{\tilde{x} \in [0,1]^{d-1}} (\mu - \gamma) \tilde{p}(\tilde{x}) d\tilde{x} \end{aligned}$$

which shows that $\frac{\mu - \gamma}{\gamma - \lambda} = \frac{p_1}{p_0}$, and thus, using (6) and (9):

$$E - E' = p_1 E_1 - \frac{p_1}{p_0} p_0 E_0 = p_1 (E_1 - E_0) \geq 0$$

since we are in the situation where $E_0 \leq E_1$. This means the new tree h' does not degrade the generalization error compared to h . In the case where $E_0 > E_1$, the same reasoning can be applied when replacing threshold γ with λ instead of μ , in order to obtain a tree h' with a similar (or lower) generalization error. After this step, one of the two following situations can occur:

- Either the new threshold is equal to a threshold of a parent node splitting on the same coordinate. In this case, it is useless and the current node can be deleted (along with one of its subtrees), leading to a smaller tree. The above procedure can then be iterated.
 - Or the new threshold is equal to $\alpha_{i,j}$ or $\alpha_{i,j+1}$, and the above procedure can also be iterated (note that if this threshold is equal to 0 or 1, then the tree can also be pruned).
- Since at each step we either remove a node or set its threshold to an interval boundary $\alpha_{i,j}$, in the end we obtain a tree that (i) does not contain more nodes than h , (ii) does not have a higher generalization error than h , and (iii) has only thresholds among interval boundaries $\alpha_{i,j}$.

We can now study the case where (1) is verified at each node. A direct consequence is that the output function h is constant on all cells of the target function f . Let M the number of pieces in the piece-wise constant function h (i.e. the number of leaf nodes in the tree). If $M < N$ pieces, there must be two neighbor cells C and C' on which h assigns the same value t , and on which function f takes values respectively c and c' . The generalization error E of h is then at least $E_{C \cup C'}$, with

$$\begin{aligned} E_{C \cup C'} &= \int_{x \in C \cup C'} (h(x) - f(x))^2 p(x) dx = \int_{x \in C} (t - c)^2 p(x) dx + \int_{x \in C'} (t - c')^2 p(x) dx \\ &\geq (t - c)^2 m + (t - c')^2 m. \end{aligned}$$

This quadratic function of t is minimized for $t = \frac{c+c'}{2}$ and is equal to $\frac{m}{2}(c - c')^2$ for this value of t . Since from definition 3.7 we have $|c - c'| \geq \delta$, we can thus conclude that $E \geq \frac{m\delta^2}{2}$. \square

4 Discussion

Decision trees have been used with great success and have generated an important literature in the statistics, machine learning, and data-mining communities. Here we initiate a discussion about how they could still be used as useful components in the development of AI, with the requirement of learning highly complicated functions, i.e. with an exponential number of ϵ -variations, much larger than the number of examples one could hope to obtain.

Forests, i.e. sums of trees – random forests [10, 4], error-correcting committees of trees [12], and boosted trees [6] – are known to perform generally better than decision trees. Many empirical results support this statement, and several explanations have already been proposed, such as variance reduction in forests and margin bounds for boosting. Boosted trees and other forests have the form $f(x) = \sum_{i=1}^n \alpha_i T_i(x)$ where $T_i(x)$ is the prediction of the i -th tree. It has been reported often that boosted trees and random forests generalized better than single trees. The theoretical results presented here suggest yet another reason for the better performance of forests and boosted trees

over single trees. Indeed, our negative theorems do not apply to sums of trees. In fact we have that:

Proposition 4.1. *Let $f(x) = \sum_{i=1}^n \alpha_i T_i(x)$ be a forest of constant-leaves decision tree with axis-aligned decision nodes defined on an input space of dimension d , with $n \leq d$. Then the number of different values f can take can grow exponentially with n , even in situations where the number of different values each T_i can take is bounded by a fixed constant.*

Proof. Let $T_i(x) = \mathbb{1}_{x_i < \frac{1}{2}}$ and $\alpha_i = 2^{i-1}$. For any $k \in \{0, 1, 2, \dots, 2^n - 1\}$ there exists $x \in \mathbb{R}^d$ such that $f(x) = k$: since $n \leq d$ we can simply use the binary representation $b = b_{n-1} \dots b_1 b_0$ of k , and set $x_i = b_{i-1}$ for $i \leq n$ (and for instance $x_i = 0$ for $i > n$). \square

This proof suggests a different way to combine trees. If we consider the output $T(x)$ of a tree to be a discrete variable specifying in which leaf x falls, then we can consider the output of a forest as the encoding of a vector whose elements are these discrete variables. Clearly, this is a form of distributed representation [7], which can express a number of configurations possibly exponential in the number of trees (even though the model is expressed with a set of numbers of size linear in the number of trees). This expressive power (a small set of numbers saying things about a large set of distinct regions in input space) is also what can buy us strong generalization power. Note how in error-correcting output coding with one tree for each output bit [12], we have a fixed distributed representation (the output code). The work developed by Hinton over the last two decades (e.g. see [7, 8, 15, 9]) is instead geared towards *learning* internal representations that help to capture the main factors of variation in the data.

Learning algorithms that learn to represent functions with many levels of composition are said to have a *deep architecture* (we talk here of *architectural depth*, different from tree depth). [3] discusses results in computational theory of circuits that strongly suggest that deep architectures are much more efficient in terms of representation (number of computational elements, number of parameters) than their shallow counterparts. In spite of the fact that 2-level architectures (e.g., a one-hidden layer neural network, a kernel machine, or a 2-level digital circuit) are able to represent any function (see for example [11]), they may need a huge number of elements and, consequently, of training examples. For example, the parity function on d bits can be implemented by a digital circuit of *architectural depth* $\log(d)$ with $O(d)$ elements but requires $O(2^d)$ elements to be represented by a 2-level digital circuit [1], e.g., in conjunctive or disjunctive normal form. A similar result was proved for Gaussian kernel machines: they require $O(2^d)$ non-zero coefficients (i.e., support vectors in a Support Vector Machine) to represent such highly varying functions [2].

What is the architectural depth of decision trees and decision forests? It depends on what elementary units of computation are allowed on each level. By analogy with the disjunctive normal form (which is usually assigned an architectural depth of two) one would assign an architectural depth of two to a decision tree. The top level disjunction is a sum over the terms associated with each leaf. A first level conjunctive term is a product of the indicator functions associated with each internal node and with the predicted constant associated with the leaf. With this interpretation, a decision forest

has an architectural depth of three. An extra summation layer is added. Note how this summation layer is very different from the top layer of the decision tree architecture. Although both perform a summation, the decision tree top layer sums over mutually exclusive terms, whereas the decision forest sums over terms which are generally non-zero, allowing an exponential number of combinations of leaves (one from each tree) to be added, as discussed above.

5 Conclusion

Inspired by previous work [2] showing limitations of Gaussian kernel machines to learn highly varying functions (even when a simple expression for the solution exists), we showed similar negative results for decision trees. We believe that the arguments made in this paper can easily be generalized to the case of decision trees with non-constant leaf predictions (such as linear predictions) and decision nodes that are not axis-aligned. Formal proofs for these more general cases remain to be established, but the crucial (killing) ingredients that remain valid from the current analysis are: (i) only the examples falling into the leaf are used to produce the estimator associated with the leaf, and (ii) the leaves are associated with non-intersecting regions.

This analysis gives an alternative explanation for the better generalization abilities of forests and boosted trees. The latter actually exploit a distributed representation of the input space in order to generalize to regions not covered by the training set, giving them exponentially more efficient power than single decision trees. One question raised by this work and inspired by results on complexity theory of circuits is whether even forests and boosted trees could be significantly improved upon by considering yet deeper architectures.

References

- [1] M. Ajtai. \sum_1^1 -formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):48, 1983.
- [2] Y. Bengio, O. Delalleau, and N. Le Roux. The curse of highly variable functions for local kernel machines. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 107–114. MIT Press, Cambridge, MA, 2006.
- [3] Y. Bengio and Y. Le Cun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, 2007.
- [4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.
- [6] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of Thirteenth International Conference*, pages 148–156, 1996.
- [7] G. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, Amherst 1986, 1986. Lawrence Erlbaum, Hillsdale.

- [8] G. Hinton and Z. Ghahramani. Generative models for discovering sparse distributed representations. *Philosophical Transactions of the Royal Society of London*, B(352):1177–1190, 1997.
- [9] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.
- [10] T. K. Ho. Random decision forest. In *3rd International Conference on Document Analysis and Recognition*, pages 278–282, Montreal, Canada, 1995.
- [11] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [12] E. B. Kong and T. G. Dietterich. Error-correcting output coding corrects bias and variance. In *International Conference on Machine Learning*, pages 313–321, 1995.
- [13] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Second edition, Springer, New York, NY, 1997.
- [14] W.-Y. Loh and Y.-S. Shih. Split selection methods for classification trees. *Statistica Sinica*, 7:815–840, 1997.
- [15] A. Paccanaro and G. Hinton. Extracting distributed representations of concepts and relations from positive and negative propositions. In *Proceedings of the International Joint Conference on Neural Network, IJCNN'2000*, Como, Italy, 2000. IEEE, New York.
- [16] E. Pérez and L. A. Rendell. Learning despite concept variation by finding structure in attribute-based data. In *Proceedings of the 13th International Conference on Machine Learning*, pages 391–399, 1996.
- [17] R. Vilalta, G. Blix, and L. Rendell. Global data analysis and the fragmentation problem in decision tree induction. In *Proceedings of the 9th European Conference on Machine Learning*, pages 312–327. Springer-Verlag, 1997.