Computational Intelligence
An International Journal

Computational Intelligence, Volume 26, Number 4, 2010

DECISION TREES DO NOT GENERALIZE TO NEW VARIATIONS

YOSHUA BENGIO, OLIVIER DELALLEAU, AND CLARENCE SIMARD
Department of IRO, Université de Montréal, Montreal, Canada

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 varying functions such as those presumably needed to achieve artificial intelligence, we study some theoretical limitations of decision trees. We demonstrate formally that they can be seriously hurt by the curse of dimensionality in a sense that is a bit different from other nonparametric statistical methods, 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 to make a sensible prediction in that region. A better understanding of the fundamental reasons for this limitation suggests that one should use *forests* or even deeper architectures instead of trees, which provide a form of distributed representation and can generalize to variations not encountered in the training data.

Key words: decision trees, curse of dimensionality, parity function.

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 artificial intelligence (AI) tasks that remain beyond the reach of current algorithms include many tasks involving visual perception, auditory perception, and natural language processing. What we would really like to see are machines that appear to really understand the concepts involved in these tasks. We argue that achieving AI through machine learning involves capturing a good deal of the statistical regularities underlying complex data such as natural text and video. These data objects live in very high-dimensional spaces where the number of possible combinations of values is huge. Yet, we expect these regularities to be representable in a comparatively compact form, simply because they arise from the laws of physics and the organization of our world. Mammals are able to capture a great deal of that structure in a brain that is small in comparison with the set of possible combinations of values of their sensors. Consider as a simple example the variations in pixel intensities one obtains by taking pictures of the same 3D object under different illuminations, in front of different backgrounds, and with different camera geometry, as illustrated in the NORB data set (LeCun, Huang, and Bottou 2004). Changing only slightly one of these factors (that we will call *factors of variation* in the remainder of the article), for example, rotating the object, gives rise to very different images, when an image is looked at as a vector of pixel intensities, associated with Euclidean distance as a metric. Specifically, Bengio, Monperrus, and Larochelle (2006b) illustrate how rotation or translation maps out a manifold in the space of pixel intensities that is highly curved. A function that would be used to really identify an object or estimate density in the space of such data would have to capture most of these variations. If in addition, we consider all the factors that can interact in creating the variations that are observed in natural language text or in video, it becomes clear that modeling such data requires learning *functions with a large number of variations*.

To approach the kind of proficiency that we aim for, it therefore seems plausible to assume that the required learning algorithms will have to learn functions with a large number of variations, which however can be represented compactly (i.e., there exists a

Address correspondence to Olivier Delalleau, University of Montreal, CP 6128, Succ. Centre-Ville Montreal, Quebec H3C 3J7, Canada; e-mail: delallea@iro.umontreal.ca

small number of factors underlying these variations). These variations correspond to many possible combinations of values for the different factors of variation that underlie the unknown generating process of interest. Note that this large set of variations is not arbitrary. Because these variations arise through complex interactions of real-world factors, these variations in the desired function value are structured: one expects that there exists a reasonably simple[1] program (such as the one implicitly computed by human brains) that can predict these variations well. Hence, by the theoretical arguments of Kolmogorov complexity (Solomonoff 1964; Kolmogorov 1965; Li and Vitanyi 1997; Hutter 2005), one would also expect that some learning algorithms could discover the essential elements of this structure, which would be required to truly generalize in such domains.

A better understanding of the limitations of current algorithms can serve as a guide in moving statistical machine learning toward artificial intelligence. If our goal is to achieve AI through machine learning it is important both to identify the limitations of current learning algorithms with respect to learning highly varying (but structured) functions, and to understand these limitations well enough to work around them.

The study of limitations of particular classes of learning algorithms with respect to learning highly varying functions is not new. This article is inspired by previous work that has shown such limitations in the case of kernel methods with a *local kernel* (Bengio, Delalleau, and Le Roux 2006a) as well as in the case of so-called *shallow function classes* (Bengio and LeCun 2007)—which include all fixed-kernel kernel machines such as Support Vector Machines (Boser, Guyon, and Vapnik 1992; Cortes and Vapnik 1995). These papers study in particular the case where the predicted function has the form $f(x) = \sum_i \alpha_i K(x, x_i)$, where x_i are training examples and $K(u, v)$ is a "local" function such as the Gaussian with spread σ. Here, *local* means that a training example x_i has mostly influence on $f(x)$ only for x near x_i. When $\sigma \to 0$ the function is more local and can model more variations (the "bumps" can be distinguished), and when $\sigma \to \infty$ the function becomes quickly very smooth (first a second-order polynomial, then an affine predictor, then a constant predictor). The function can be seen as the addition of local bumps, and it is mostly the training examples in the region around a training example x_i that contribute to the value of the function around x_i. For simple to analyze and simple to express but highly varying functions such as the parity function (also studied here), one can show that an exponential number of training examples is necessary to obtain a given level of generalization error.

Here, we focus on similar limitations, in the case of decision trees. The theorems are specialized to the most common type of decision trees, which has axis-aligned decision nodes and constant leaves, that is, where each node partitions the data getting into it according to whether a particular input variable is greater or not than a threshold, and where for the data associated with a leaf, the predicted output is a constant (chosen by learning). However, we believe that similar proof techniques could be extended to wider classes of decision trees. Previous theoretical and empirical studies have already shown that decision trees can be severely limited in their expressive power, unless one allows the number of leaves to be very large (e.g., exponential in the input size). For example Grigoriev, Karpinski, and Yao (1995) have shown that to compute the MAX function (answering whether the jth input is the maximum over the given n inputs) one would require a tree with a size exponential in n. A related result (which is also closely related to the spirit to this article) is found in Cucker and Grigoriev (1999) and is discussed below. It states a lower bound on the depth of a decision tree when some functions must be approximated with an error less than δ.

[1] Simple in the sense that its size is small compared to the number of possible combinations of these underlying factors of variation.

Decision trees were introduced in Breiman et al. (1984): A decision tree recursively partitions the input space and assigns an output value for each of the input regions in that partition. Each node of the tree corresponds to a region of the input space and the root is associated with the whole input space. The whole tree corresponds to a piece-wise constant function where the pieces are defined by the internal decision nodes, each leaf is associated with one piece, along with a constant to output in the associated region.

In this article, 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 (Pérez and Rendell 1996; Vilalta, Blix, and Rendell 1997) showing that the generalization performance of decision trees degrades when the number of variations in the target function increases. Whereas other nonparametric learning algorithms also suffer from the curse of dimensionality, the way in which the problem appears in the case of decision trees is different and helps to focus on the fundamental difficulty. The general problem is not really dimensionality, nor is it about a predictor that is a sum of purely local terms (like kernel machines). The problem arises from dividing the input space in regions (in a hard way in the case of decision trees) and having separate parameters for each region. Unless the parameters are tied in some way or regularized using strong prior knowledge, the number of available examples thus limits the complexity one can capture, that is, the number of independent regions that can be distinguished.

2. DEFINITIONS

An internal node of the tree is associated with a decision function that splits the region (associated with this node) into subregions. Each subregion corresponds to a child of this internal 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, for example, using cross-validation. A decision tree-learning algorithm is thus nonparametric, constructing a more flexible function when more training examples are available. In many implementations (see Breiman et al. 1984) 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 Loh and Shih 1997), or n-ary splits in the internal nodes.

Definition 1 (n-ary split function). Let n be an internal node of a tree and $n_1, \ldots, n_k$ its k child nodes. The n -ary split function S_n of node n is defined on the region R_n of input space $\mathbb{R}^d$ associated with n, and takes values in $\{n_1, \ldots, n_k\}$. It, thus, defines a region R_{n_i} associated with each child node n_i such that $R_{n_i} = \{x \in R_n | S_n(x) = n_i\}$.

Definition 2 (Decision Tree). A decision tree is the function $T : \mathbb{R}^d \to \mathbb{R}$ resulting from a learning algorithm applied on training data lying in input space $\mathbb{R}^d$, which always has the following form:

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

where $R_i \subset \mathbb{R}^d$ is the region associated with leaf i of the tree, $ancestors(i)$ is the set of ancestors of leaf node i, $c_{a,i}$ is 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 with 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 nonzero (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 training example falling in it (i.e., no R_i is empty).

Definition 3 (Piecewise Constant). We say function $f : R \subseteq \mathbb{R}^d \to \mathbb{R}$ 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 subsets of $\mathbb{R}^d$ such that $\cup_{i=1}^{N} R_i = R$.

Definition 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 5 (Constant-Leaves Decision Tree). A constant-leaves decision tree is a decision tree as in Definition 2 such that for each leaf node i, the decision function $g_i(\cdot)$ 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 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 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

Cucker and Grigoriev (1999) prove a very interesting result on the complexity of function approximation when a round-off error is allowed. We believe that their result is intimately connected to the inability of decision trees to generalize to new variations. Their result, illustrated in Figure 1 , is about trees that define a piecewise-polynomial function T that can approximate another piecewise-polynomial function f. f is associated with regions $V_i \subset \mathbb{R}^d$ that form a partition of the input space $\mathbb{R}^d$, so that f is polynomial in each V_i. The size of these regions is characterized by a quantity $w(\tau)$ that is the number of pieces large enough to contain a d-dimensional cube of side τ. They define a tolerance Γ_τ that depends only on the target function f and the choice of τ, and they prove in their main theorem that if T approximates f with error δ and $\delta < \Gamma_\tau$, then the depth of T cannot be less than $\log_2 w(\tau)$.

The following Lemma can be seen as stating for constant-leaves (piece-wise constant) decision trees a result that has the same flavor as the aforementioned, but in the context of learning and generalization. Instead of characterizing the complexity of the target function by the geometry of regions, we simply count the number of regions N needed to obtain a given accuracy.

Lemma 1. Let $\mathcal{F}$ be the set of piece-wise constant functions. Consider a target function $h : D \subseteq \mathbb{R}^d \to \mathbb{R}$. For a given representation error level ϵ, let N be the minimum number

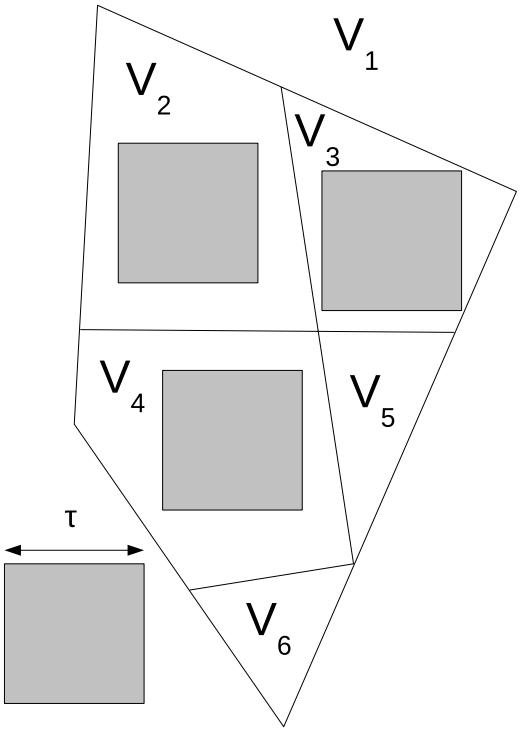

FIGURE 1. Example of a partition of $\mathbb{R}^2$ into six regions V_1 to V_6, with the number of regions able to contain a 2D square of side τ being $w(\tau) = 4$ (V_1 to V_4 can contain such a square, while V_5 and V_6 are too small). A tree approximating a piecewise-polynomial function f defined by these regions must have depth at least $\log_2(w(\tau)) = 2$ when the approximation error is sufficiently small (less than some threshold Γ_τ).

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. This is a direct consequence of the fact that a constant-leaves decision tree with ℓ leaves is piecewise constant with at most ℓ pieces, and each leaf must contain at least one training example. ■

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 exponential growth statements in the special cases of two classes of functions: the parity function and the checkerboard functions (defined later). What is important to note here, is that these functions may otherwise be represented compactly, suggesting that some rather generic learning algorithms could learn them. This is justified by the fact that the Kolmogorov complexity (Kolmogorov 1965; Li and Vitanyi 1997) of these functions could be very low, that is, one would be able to express them with a small program in any current programming language. Functions with low Kolmogorov complexity can theoretically be learned with few examples (Solomonoff 1964; Kolmogorov 1965; Li and Vitanyi 1997; Hutter 2005), but we show in this section that

decision trees are unable to do so, regardless of the learning algorithm being used. Keep in mind that although the following classes of functions may be easy to represent compactly in some standard programming language, it does not necessarily mean it is easy to *learn* this representation, because one needs an efficient way to search in the space of programs. The results in this section do not tell us how to solve this computational complexity issue, but by highlighting some fundamental limitations of decision trees, they also give some ideas as to how one may get around them: this will be discussed in Section 4.

3.1. Curse of Dimensionality on the Parity Task

It was already known from empirical evidence that decision trees were not able to learn the parity function, in the sense of not generalizing to regions of the input space that do not correspond to a training example (see, e.g., Pérez and Rendell 1996). The mathematical results in this section show this formally, connecting the number of training examples, input dimension, and generalization error.

Definition 8 (d-bit Parity Task). The d -bit parity task has as target function the d -bit parity function $p : \{0, 1\}^d \subset \mathbb{R}^d \to \{-1, 1\}$,

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

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 9 (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 2. As illustrated in Figure 2 , 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 subregions 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 with regard to this coordinate (as this would result in empty nodes), and

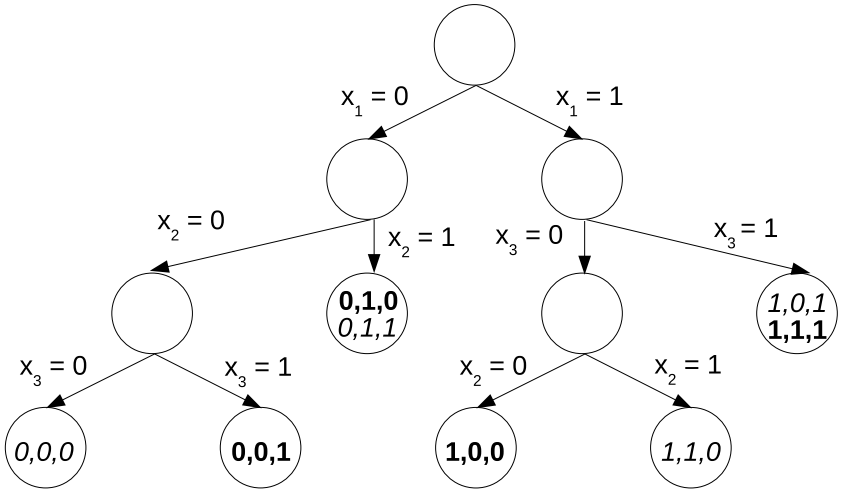

FIGURE 2. Illustration of Lemma 2 on a tree with axis-aligned decision nodes in $\{0, 1\}^3$. Data points are listed inside the leaf node they fall into, in *italics* for points whose parity output would be -1, and in **bold** for those whose parity output would be 1. The two leaf nodes with depth less than 3 must have a classification error of $\frac{1}{2}$, since they contain one of each kind.

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'). ■

Theorem 1. 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 2 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 $\epsilon \leq \frac{1}{2} - \frac{2(n-2^{d-1})}{2^{d+1}} = \frac{1}{2} - \frac{2n}{2^{d+1}} + \frac{2^d}{2^{d+1}} = 1 - \frac{n}{2^d}$. ■

Corollary 1. 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 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 1 we find $n(\epsilon) \geq 2^d(1-2\epsilon)$. ■

3.2. Curse of Dimensionality for the Checkerboard Task

The parity function may look maybe too simple. Can we generalize some of its properties? The checkerboard task defined later is similar in spirit to the parity function, in the sense that it defines a large number of regions such that the target output in each region is different from the target output in neighboring regions, but in such a way that the overall function can be written down with an expression much smaller than the total number of regions, that is, a learning algorithm approximately minimizing Kolmogorov complexity should be able to discover a good solution without requiring an exponential number of examples.

Definition 10 (Checkerboard Task). A Checkerboard task over $[0,1)^d$ *with minimum variation* δ, *minimum mass m per board cell and interval numbers* $\{n_i\}_{i=1}^d$ *defines:*

- *for each dimension* $i \leq d$, *a partition of* $[0,1)$ *into* n_i *intervals* $\{[\alpha_{i,j}, \alpha_{i,j+1})\}_{j=1}^{n_i}$ *with* $\alpha_{i,1} = 0$, $\alpha_{i,n_i+1} = 1$, *and* $\alpha_{i,j} < \alpha_{i,j+1}$
- *a target function f constant over each cell* $C_{j_1,\ldots,j_d} = [\alpha_{1,j_1}, \alpha_{1,j_1+1}) \times \cdots \times [\alpha_{d,j_d}, \alpha_{d,j_d+1})$, *such that the constant values of f on two neighborhing cells differ at least by* δ.

To form a data set, the inputs x are sampled with a probability distribution p such that $P(x \in C_{j_1,j_2,\ldots,j_d}) = \int_{x \in C_{j_1,j_2,\ldots,j_d}} p(x)\,dx \geq m$ *for every cell* $C_{j_1,j_2,\ldots,j_d}$ *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* $E = \int (h(x) - f(x))^2 p(x) dx$.

Proposition 1. 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.*

Before going into the detailed proof, we first give an intuitive explanation for this result. The idea is that if a tree is trained with less than N different examples, then its output must be constant on two neighboring cells. Because the target values in these cells differ by at least δ, and each cell has a probability mass at least m, then the squared error is related to $m\delta^2$. This reasoning is detailed in the last paragraph of the following proof (explaining where the $\frac{1}{2}$ factor comes from), while most of this proof is dedicated to showing first that we may consider only trees using cell boundaries as splitting thresholds.

Proof. We will prove that a tree achieving a generalization error less than $\frac{m\delta^2}{2}$ must have at least N leaf nodes, which by Definition 2 of a decision tree implies it has been trained with at least N examples (since there must be at least one training example falling in each leaf).

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

$$S(x) = \mathbb{1}_{x_i < \alpha_{i,j}}, \tag{2}$$

i.e., whose splitting thresholds on dimension i are constrained to be among the interval boundaries $\{\alpha_{i,j}\}_{j=1}^{n_i}$. 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 (2) 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 (2), that is, is of the form:

$$S(x) = \mathbb{1}_{x_i < \gamma}, \tag{3}$$

where $\gamma \in (0, 1)$ (otherwise some node would contain no example) and such that $\forall j = 1, \ldots, n_i + 1$ 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 $j \in \{1, \ldots, n_i\}$ 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:

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

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:

$$\alpha_{i,j} \leq \lambda < \gamma < \mu \leq \alpha_{i,j+1}. \tag{4}$$

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 with regard to 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 (2), and thus all its child nodes that may split with regard to 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 $D_0 = \{x \in [0, 1)^d | x_i \in [\lambda, \gamma)\}$:

$$E_0 = \frac{1}{\int_{x \in D_0} p(x)\, dx} \int_{x \in D_0} (h(x) - f(x))^2 p(x)\, dx$$
$$\stackrel{def}{=} \frac{1}{p_0} J(D_0).$$

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

$$\begin{aligned} E &= \int_{x \in [0,1)^d} (h(x) - f(x))^2 p(x)\, dx \\ &= J(D_0) + J(D_1) + J([0, 1)^d \setminus (D_0 \cup D_1)) \\ &= p_0 E_0 + p_1 E_1 + J([0, 1)^d \setminus (D_0 \cup D_1)). \end{aligned} \tag{5}$$

Let us first consider the case $E_0 \leq E_1$. Let h' the output function that would result from replacing threshold γ in (3) 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:

$$E' = J'(D_0) + J'(D_1) + J'([0, 1)^d \setminus (D_0 \cup D_1)). \tag{6}$$

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 (5) and (6), the difference between generalization errors E and E' reduces to

$$E - E' = p_1 E_1 - J'(D_1). \tag{7}$$

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

$$\begin{aligned} J'(D_1) &= \int_{x \in D_1} (h'(x) - f(x))^2 p(x)\, dx \\ &= \int_{\tilde{x} \in [0,1)^{d-1}} \left(\int_{x_i \in [\gamma, \mu)} (h'(x) - f(x))^2 p(x)\, dx_i \right) d\tilde{x}. \end{aligned} \tag{8}$$

To simplify (8) we observe that, for a fixed $\tilde{x}$, $h'(x)$ is constant with regard to $x_i \in [\gamma, \mu)$ 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 with regard to the ith 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 ith 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 (4).

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

$$\begin{aligned} J(D_0) &= \int_{x \in D_0} (h(x) - f(x))^2 p(x)\, dx \\ &= \int_{\tilde{x} \in [0,1)^{d-1}} \left(\int_{x_i \in [\lambda, \gamma)} (h(x) - f(x))^2 p(x)\, dx_i \right) d\tilde{x}. \end{aligned} \tag{9}$$

For the same fixed $\tilde{x}$ as above, $h(x)$ is constant with regard to $x_i \in [\lambda, \gamma)$ 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 earlier, that is, with its ith coordinate set to λ). Finally, we observe

that for a fixed $\tilde{x}$, both $p(x)$ and $f(x)$ are also constant with regard to $x_i \in [\lambda, \mu)$. 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 (8) and (9) into

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

$$J(D_0) = \int_{\tilde{x}\in[0,1)^{d-1}} (\gamma - \lambda)(h(y) - \tilde{f}(\tilde{x}))^2 \tilde{p}(\tilde{x})\, d\tilde{x}$$

and consequently

$$J'(D_1) = \frac{\mu - \gamma}{\gamma - \lambda} J(D_0) = \frac{\mu - \gamma}{\gamma - \lambda} p_0 E_0. \tag{10}$$

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

$$p_0 = \int_{\tilde{x}\in[0,1)^{d-1}} \left(\int_{x_i \in [\lambda,\gamma)} \tilde{p}(\tilde{x})\, dx_i \right) d\tilde{x} = \int_{\tilde{x}\in[0,1)^{d-1}} (\gamma - \lambda)\tilde{p}(\tilde{x})\, d\tilde{x}$$

$$p_1 = \int_{\tilde{x}\in[0,1)^{d-1}} \left(\int_{x_i \in [\gamma,\mu)} \tilde{p}(\tilde{x})\, dx_i \right) d\tilde{x} = \int_{\tilde{x}\in[0,1)^{d-1}} (\mu - \gamma)\tilde{p}(\tilde{x})\, d\tilde{x},$$

which shows that $\frac{\mu-\gamma}{\gamma-\lambda} = \frac{p_1}{p_0}$, and thus, using (7) and (10):

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

because 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, 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 (2) 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

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

with m the minimum mass per cell of the checkerboard task. 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. Because from Definition 10, we have $|c - c'| \geq \delta$, we can thus conclude that $E \geq \frac{m\delta^2}{2}$. ■

4. DISCUSSIONS

Decision trees have been used with great success and have generated an important literature in the statistics, machine learning, and data-mining communities. Even though our results suggest that they are insufficient to learn the type of task involved in AI (with a number of ϵ-*variations* much greater than the number of examples one could hope to get), they might still be used as useful components. The above results should also help us gather a better understanding of the limitations of nonparametric learning algorithms by illustrating the differences and common pitfalls of decision trees and other nonparametric learning algorithms.

4.1. Trees versus Local Nonparametric Models

Local nonparametric learning methods such as Gaussian Support Vector Machines (SVMs) and nearest-neighbor classifiers or Parzen windows are hurt by the curse of dimensionality because they associate a separate set of parameters to each region, where each region is a kind of blob centered on a training example (like the radially defined Gaussians in Gaussian kernel machines). Instead, decision trees define regions which can extend for arbitrary distances away from a training example. According to the definition of *local* given by Bengio et al. (2006a), decision trees are nonlocal: they can generalize to a test point arbitrarily far from a training point because they can ignore some dimensions (and do so differently in different parts of the space), and this is true not just for test points that are far from the cloud of training points. Hence the sense in which they are cursed by dimensionality is a bit different from local nonparametric methods such as nearest-neighbor methods, kernel density estimation, or SVMs. However, we believe that there is a way to view these two effects under a common light, by thinking about the notions (that we have tried to highlight in this article) of *variations* and *regions*. Both types of methods construct some kind of soft or hard partition of the input space, and consider a simple parametrization inside each region of this partition: constant model in the case of constant-leaves decision trees and nearest-neighbor classifiers or histograms, and something more powerful (close to a low-degree polynomial) in the case of Gaussian SVMs. What hurts generalization in both cases is the need to have examples in each of these regions to be able to generalize. What is missing is the ability to learn something about the statistical structure in some region of space that could be somehow applied in other regions of space, besides the immediately neighboring regions. See (Bengio, Larochelle, and Vincent 2006c; Bengio et al. 2006b) for discussions of nonlocal generalization and attempts to transform local kernel methods so as to achieve it.

So in spite of the degree of nonlocality that may confer some advantages to decision trees over other nonparametric learning algorithms, decision trees suffer from a very similar

limitation arising not so much because of the dimensionality of the input but because of the degree of variability of the target function to be learned.

4.2. Forests and Boosted Trees

Forests, that is, sums of trees—random forests (Ho 1995; Breiman 2001), error-correcting committees of trees (Kong and Dietterich 1995), and boosted trees (Freund and Schapire 1996)—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 ith 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 2. 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 $k \in \{0, 1, 2, \ldots, 2^n - 1\}$ there exists $x \in \mathbb{R}^d$ such that $f(x) = k$: since $n \leq d$ we can simply use the binary representation $b = b_{n-1} \ldots b_1 b_0$ of k, and set $x_i = b_{i-1}$ for $i \leq n$ (and for instance $x_i = 0$ for $i > n$). ■

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, one per tree in the forest. Clearly, this is a form of distributed representation (Hinton 1986), 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 could buy strong generalization power (for the same reason that a model with a smaller Kolmogorov complexity explaining correctly a much larger data set is likely to generalize well). Note how in error-correcting output coding with one tree for each output bit (Kong and Dietterich 1995), we have a fixed distributed representation (the output code). The work developed by Hinton over the last two decades (e.g., see Hinton 1986; Hinton and Ghahramani 1997; Paccanaro and Hinton 2000; Hinton, Osindero, and Teh 2006) is instead geared toward *learning* internal distributed representations that help capture the main factors of variation in the data.

4.3. Architectural Depth and Distributed Representations

Learning algorithms that learn to represent functions with many levels of composition are said to have a *deep architecture* (we are talking here about *architectural depth*, different from tree depth). Bengio and LeCun (2007) discuss results in computational theory of circuits that strongly suggest that, compared to their shallow counterparts, deep architectures are much more *efficient in terms of representation*, that is, can require a smaller number of computational elements or of parameters to approximate a target function. 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 (Hornik, Stinchcombe, and White 1989, see for example), they may need a huge number of elements and, consequently, of training examples. Generally, a function that can be represented efficiently with a circuit of depth k may require an exponentially larger circuit of depth $k-1$. 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 (Ajtai 1983), for example, in conjunctive or disjunctive normal form. A similar result was proved for Gaussian kernel machines: they require $O(2^d)$ nonzero coefficients (i.e., support vectors in a SVM) to represent such a highly varying function (Bengio et al. 2006a). Note however that parity can be represented efficiently with 2 or 3 levels if the units at each level are slightly more powerful, for example, with an RBF network that has different spreads in each unit, or with a multi-layer neural network with two hidden layers, so it may not be the best illustration of what requires deeper architectures. Another example, discussed by Bengio and LeCun (2007), is that of multiplication of n-bit integers using digital circuits. It can either be achieved with a two-layer architecture that has a number of gates exponential in n, or efficiently with a deep circuit of $O(\log n)$ layers.

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, and of three to decision forests or boosted trees. The top-level disjunction computed by a decision tree 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 nonzero, allowing an exponential number of combinations of leaves (one from each tree) to be added, as discussed above.

It is interesting to pursue the analogy between polynomials and decision trees, in the context of shallow versus deep architectures. For the sake of simplicity, we consider the case of binary inputs $x = (x_1, \ldots, x_d) \in \{0,1\}^d$, and binary functions (i.e., the output is in $\{0,1\}$ as well). We will see how a decision tree corresponds to some kind of (shallow) expansion of a polynomial, while there may exist a kind of (deep) factorization allowing a more compact representation of the same function. In the context of binary inputs and output, equation (1) becomes of the form

$$T(x) = \sum_{(y,\alpha)\in L} \Pi_{i=1}^{d} (y_i x_i + (1-y_i)(1-x_i))^{\alpha_i}, \tag{11}$$

where each $(y,\alpha) \in L$ is associated with a leaf where the output value is 1, and both y and α are d-dimensional binary vectors[2]: the ith term of the product is ensuring that x_i and y_i are equal when α_i is 1 (while being indifferent to their values when α_i is 0). The nonzero values of α thus correspond to the variables that are tested on the path from the root of the tree to a leaf (and their number is the length of this path), while y contains the values of these variables that lead to this leaf (note that the value of y_i does not matter when $\alpha_i = 0$, because the variable is not being tested). The resulting function $T(x)$ thus has a polynomial

[2] Here we use the convention $0^0 = 1$.

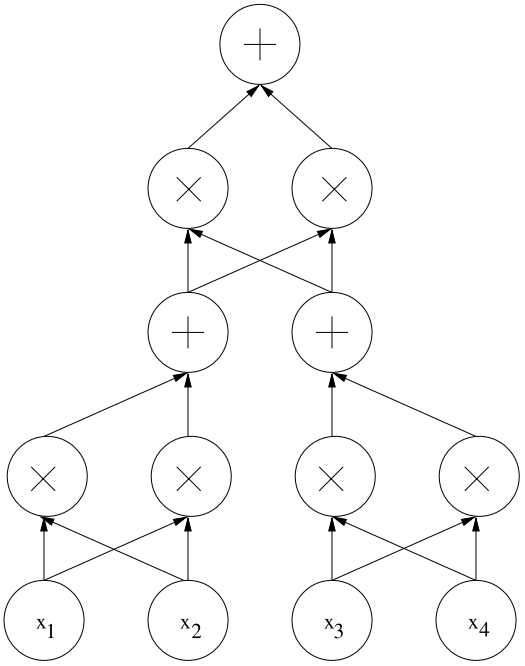

FIGURE 3. Example of a deep architecture obtained when adding extra layers performing a second "sum of products" operation. Here, the + nodes simply sum their children. If the × nodes compute the product of one of their children by the negation of their other child, then this architecture computes the parity function (see text for details).

form, and we call its representation by equation (11) its *expansion* (in the form of sums of products). Note that *any* binary function over $\{0, 1\}^d$ can be written in this form, because for any f we can write

$$f(x) = \sum_{y \in \{0,1\}^d} f(y) \mathbb{1}_{y=x} = \sum_{y \in \{0,1\}^d \ s.t. \ f(y)=1} \Pi_i (y_i x_i + (1 - y_i)(1 - x_i)).$$

This equation shows that we do not need the α's in equation (11) to represent a function. However, they can lead to a more compact representation: for instance the function $x_1 \ OR \ x_2$ can be written $x_1 + (1 - x_1)x_2$ instead of $x_1 x_2 + (1 - x_1)x_2 + x_1(1 - x_2)$. It remains a shallow representation though, corresponding to a 2-level architecture with a hidden layer containing as many nodes as leaves (each node computing a product $\Pi_{i=1}^d (y_i x_i + (1 - y_i)(1 - x_i))^{\alpha_i}$), and a single output unit that simply computes the sum of all hidden nodes. Consider now a deeper architecture obtained first by allowing multiple output units (each performing a sum over a different subset of hidden nodes), then adding two extra layers similar to our initial hidden and output layers (i.e., the first extra layer's nodes compute products of their inputs, possibly negated, while the second extra layer is a unit computing the sum of all these nodes). This process can be repeated such as to obtain an architecture of depth $2k$, that we call *factorized* (as a sum of products of sums of products of ...). Figure 3 shows an example of such an architecture for $k = 2$. When expanding it into an equivalent expansion of the form of equation (11), one may require a number of terms in the sum (i.e., of nodes in the hidden layer of the 2-level corresponding architecture, and of leaves in the corresponding decision

tree) potentially exponential in the depth of the tree, requiring many more parameters to be tuned when learning it from data.

As an example, consider representing the parity function in input dimension $d = 2^k$, with its output defined to be 1 if the number of nonzero bits in the input is odd, and 0 otherwise. It is easy to see that a decision tree of the form of equation (11) requires a sum over 2^{d-1} terms (all possible inputs for which the output is 1) to perfectly model this function, because no variable can be ignored in the decision (i.e., all α_i's must be 1). On another hand, consider a deep architecture of depth $2k$ where the $(2j-1)$th layer computes products of pairs of (possibly negated) nodes in its input layer, more precisely by denoting z_i^ℓ the ith unit in layer ℓ,

$$z_{2i-1}^{2j-1} = z_{2i-1}^{2j-2}\left(1 - z_{2i}^{2j-2}\right)$$
$$z_{2i}^{2j-1} = \left(1 - z_{2i-1}^{2j-2}\right)z_{2i}^{2j-2},$$

and the layer above is made of half the number of nodes, each computing a sum

$$z_i^{2j} = z_{2i-1}^{2j-1} + z_{2i}^{2j-1}.$$

The resulting architecture for $d = 4$ (i.e., $k = 2$) is illustrated in Figure 3. The interpretation is that each node of layer $2j$ represents the output of the parity function over two nodes of layer $2(j-1)$, and thus, by recursion, it is also the output of the parity function over a subset of 2^j bits of the input. Consequently, the top layer is a single unit computing the parity function over the whole input. It is easy to see that such an architecture has a total number of units equal to $3(d-1)$, and thus a number of parameters on the order of $O(d)$, which is exponentially less than in the flat expansion of equation (11). Yet, it defines the same function (that varies a lot in input space). This is an example of how the *factorized* form of a polynomial, as represented by a deep architecture, allows for a more compact representation than its *expanded* form (shallow architecture).

Even though boosted trees and forests have clearly been shown to generalize better than single decision trees in a large number of real world learning tasks (Ho 1995; Freund and Schapire 1996; Breiman 2001), we conjecture that their depth is still too limited to be able to learn highly varying functions like what is needed for the checkerboard task, in the sense of generalizing to variations not seen in the training set. The conjecture is inspired by the circuit complexity results stating that there are functions computable with a polynomial-size logic gates or threshold circuits of depth k that require exponential size when restricted to depth $k-1$ (Håstad 1986; Håstad and Goldmann 1991). In other words, the right depth may be data-dependent.

5. CONCLUSION

Inspired by previous work (Bengio et al. 2006a; Bengio and LeCun 2007) showing the inability of Gaussian kernel machines and more generally of shallow architectures to learn highly varying functions (even when a simple expression for the solution exists), we presented similar negative results for decision trees. We believe that the arguments made in this article can easily be generalized to the case of decision trees with nonconstant 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 ingredients that remain valid from the current analysis are (i) only the examples falling into a leaf are used to produce the estimator associated with this leaf and (ii) the leaves are associated with nonintersecting regions.

This analysis helps to understand the old question of the curse of dimensionality by illustrating its effect in the case of decision trees. It clarifies that the central issue is not one of dimensionality, nor purely one of local predictions (like in the case of Gaussian kernels (Bengio et al. 2006a; Bengio and LeCun 2007)). Instead it is about the limitation of estimators that divide the input space in regions (hard ones in the case of decision trees, soft ones in the case of Gaussian kernel machines), with separate parameters associated with each region. Consider the learning of a highly varying function, that is, which requires many such regions to be properly represented. Barring the injection of additional knowledge to guide the estimation of these parameters, the number of examples required to learn such models thus grows linearly with the number of these regions, that is, the complexity of the functions that can be represented. The analysis also gives an alternative conjecture to explain some of 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 a potentially exponentially more efficient representation 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.

Although previous complexity theory results (Grigoriev et al. 1995; Cucker and Grigoriev 1999), and empirical observations (Pérez and Rendell 1996; Vilalta et al. 1997) have already pointed out limitations of decision trees that originate from the same source that underlies the theorems presented here, we believe that an important contribution of this article is to connect such results with the question of learning, and in particular of learning complex tasks such as those required for AI. This puts closer to the center stage for AI and machine learning research the question of learning efficient representations of highly varying but low Kolmogorov complexity functions, such as those one would expect to need to solve AI tasks. This article adds to previously presented arguments (Bengio and LeCun 2007) suggesting that a necessary condition for solving AI tasks is that the learning algorithm should be able to construct a deep architecture for the learned function. Although these results are related to computational complexity theory results, they point to a *statistical* limitation: the need for a large tree implies the need for a large number of examples. Of course, assuming a deeper architecture does not necessarily provide better generalization because to define a learning algorithm we need in addition to a nice function class a way to search in it. The discussion arising here does not address the computational complexity issue due to the difficulty of searching in the space of deep architectures, for example, optimizing their parameters appears to be a fundamentally difficult challenge. However, new hope has arisen in the form of successful algorithms based on unsupervised learning for particular classes of deep architectures (Hinton et al. 2006; Bengio et al. 2007; Ranzato et al. 2007; Bengio 2009).

ACKNOWLEDGMENT

The authors would like to thank Aaron Courville for precious feedback and discussions. This work was supported by CIFAR, NSERC, MITACS, and Canada Research Chairs.

REFERENCES

AJTAI, M. 1983. $\sum_1^1$-formulae on finite structures. Annals of Pure and Applied Logic, **24**(1):1–48.

BENGIO, Y. 2009. Learning deep architectures for AI. (1995). Foundations and Trends in Machine Learning, **2**(1):1–127.

BENGIO, Y., and Y. LECUN. 2007. Scaling learning algorithms towards AI. *In* Large Scale Kernel Machines. *Edited by* L. Bottou, O. Chapelle, D. DeCoste, and J. Weston. MIT Press, Cambridge, MA.

BENGIO, Y., O. DELALLEAU, and N. LE ROUX. 2006a. The curse of highly variable functions for local kernel machines. *In* Advances in Neural Information Processing Systems 18 (NIPS'05). *Edited by* Y. Weiss, B. Schölkopf, and J. Platt. MIT Press, Cambridge, MA, pp. 107–114.

BENGIO, Y., M. MONPERRUS, and H. LAROCHELLE. 2006b. Non-local estimation of manifold structure. Neural Computation, **18**(10):2509–2528.

BENGIO, Y., H. LAROCHELLE, and P. VINCENT. 2006c. Non-local manifold parzen windows. *In* Advances in Neural Information Processing Systems 18 (NIPS'05). *Edited by* Y. Weiss, B. Schölkopf, and J. Platt. MIT Press, Cambridge, MA, pp. 115–122.

BENGIO, Y., P. LAMBLIN, D. POPOVICI, and H. LAROCHELLE. 2007. Greedy layer-wise training of deep networks. *In* Advances in Neural Information Processing Systems 19 (NIPS'06). *Edited by* B. Schölkopf, J. Platt, and T. Hoffman. MIT Press, Cambridge, MA, pp. 153–160.

BOSER, B. E., I. M. GUYON, and V. N. VAPNIK. 1992. A training algorithm for optimal margin classifiers. *In* Fifth Annual Workshop on Computational Learning Theory. ACM, Pittsburgh, PA, pp. 144–152.

BREIMAN, L. 2001. Random forests. Machine Learning, **45**(1):5–32.

BREIMAN, L., J. H. FRIEDMAN, R. A. OLSHEN, and C. J. STONE. 1984. Classification and Regression Trees. Wadsworth International Group, Belmont, CA.

CORTES, C., and V. VAPNIK. 1995. Support vector networks. Machine Learning, **20**:273–297.

CUCKER, F., and D. GRIGORIEV. 1999. Complexity lower bounds for approximation algebraic computation trees. Journal of Complexity, **15**(4):499–512.

FREUND, Y., and R. E. SCHAPIRE. 1996. Experiments with a new boosting algorithm. *In* Machine Learning: Proceedings of Thirteenth International Conference. ACM, USA, pp. 148–156.

GRIGORIEV, D., M. KARPINSKI, and A. C.-C. YAO. 1995. An exponential lower bound on the size of algebraic decision trees for MAX. Electronic Colloquium on Computational Complexity (ECCC), **2**(057):193–203.

HÅSTAD, J. 1986. Almost optimal lower bounds for small depth circuits. *In* Proceedings of the 18th Annual ACM Symposium on Theory of Computing. ACM Press, Berkeley, California, pp. 6–20.

HÅSTAD, J., and M. GOLDMANN. 1991. On the power of small-depth threshold circuits. Computational Complexity, **1**:113–129.

HINTON, G. E. 1986. Learning distributed representations of concepts. *In* Proceedings of the Eighth Annual Conference of the Cognitive Science Society. Lawrence Erlbaum, Hillsdale, NJ, Amherst, MA, pp. 1–12.

HINTON, G. E., and Z. GHAHRAMANI. 1997. Generative models for discovering sparse distributed representations. Philosophical Transactions of the Royal Society of London, **B**(352):1177–1190.

HINTON, G. E., S. OSINDERO, and Y. TEH. 2006. A fast learning algorithm for deep belief nets. Neural Computation, **18**:1527–1554.

HO, T. K. 1995. Random decision forest. *In* 3rd International Conference on Document Analysis and Recognition (ICDAR'95), Montreal, Canada, pp. 278–282.

HORNIK, K., M. STINCHCOMBE, and H. WHITE. 1989. Multilayer feedforward networks are universal approximators. Neural Networks, **2**:359–366.

HUTTER, M. 2005. Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer, Berlin.

KOLMOGOROV, A. N. 1965. Three approaches to the quantitative definition of information. Problems of Information and Transmission, **1**(1):1–7.

KONG, E. B., and T. G. DIETTERICH. 1995. Error-correcting output coding corrects bias and variance. *In* International Conference on Machine Learning, pp. 313–321.

LECUN, Y., F.-J. HUANG, and L. BOTTOU. 2004. Learning methods for generic object recognition with invariance to pose and lighting. *In* Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'04), Vol. 2. IEEE Computer Society, Los Alamitos, CA, pp. 97–104.

LI, M., and P. VITANYI. 1997. An Introduction to Kolmogorov Complexity and Its Applications (2nd ed.). Springer, New York.

LOH, W.-Y., and Y.-S. SHIH. 1997. Split selection methods for classification trees. Statistica Sinica, **7**:815–840.

PACCANARO, A., and G. E. HINTON. 2000. Extracting distributed representations of concepts and relations from positive and negative propositions. *In* International Joint Conference on Neural Networks (IJCNN), Como, Italy. IEEE, New York.

PÉREZ, E., and L. A. RENDELL. 1996. Learning despite concept variation by finding structure in attribute-based data. *In* Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96). *Edited by* L. Saitta. Morgan Kaufmann, San Francisco, pp. 391–399.

RANZATO, M., C. POULTNEY, S. CHOPRA, and LeCun, Y. 2007. Efficient learning of sparse representations with an energy-based model. *In* Advances in Neural Information Processing Systems 19 (NIPS'06). *Edited by* B. Schölkopf, J. Platt, and T. Hoffman. MIT Press, Cambridge, MA, pp. 1137–1144.

SOLOMONOFF, R. J. 1964. A formal theory of inductive inference. Information and Control, **7**:1–22, 224–254.

VILALTA, R., G. BLIX, and L. RENDELL. 1997. Global data analysis and the fragmentation problem in decision tree induction. *In* Proceedings of the Ninth European Conference on Machine Learning (ECML'97). Springer-Verlag, Berlin, pp. 312–327.