Neural Random Forests

Gérard Biau
Sorbonne Université, CNRS, LPSM, Paris, France
gerard.biau@upmc.fr

Erwan Scornet
Centre de Mathématiques Appliquées, Ecole Polytechnique, CNRS, Palaiseau, France
erwan.scornet@polytechnique.edu

Johannes Welbl
University College London, London, England
J.Welbl@cs.ucl.ac.uk

Abstract

Given an ensemble of randomized regression trees, it is possible to restructure them as a collection of multilayered neural networks with particular connection weights. Following this principle, we reformulate the random forest method of Breiman (2001) into a neural network setting, and in turn propose two new hybrid procedures that we call neural random forests. Both predictors exploit prior knowledge of regression trees for their architecture, have less parameters to tune than standard networks, and less restrictions on the geometry of the decision boundaries than trees. Consistency results are proved, and substantial numerical evidence is provided on both synthetic and real data sets to assess the excellent performance of our methods in a large variety of prediction problems.

Index Terms — Random forests, neural networks, ensemble methods, randomization, sparse networks.

2010 Mathematics Subject Classification: 62G08, 62G20, 68T05.

1 Introduction

Decision tree learning is a popular data-modeling technique that has been around for over fifty years in the fields of statistics, artificial intelligence, and machine learning. The approach and its innumerable variants have been

successfully involved in many challenges requiring classification and regression tasks, and it is no exaggeration to say that many modern predictive algorithms rely directly or indirectly on tree principles. What has greatly contributed to this success is the simplicity and transparency of trees, together with their ability to explain complex data sets. The monographs by Breiman et al. (1984), Devroye et al. (1996), Rokach and Maimon (2008), and Hastie et al. (2009) will provide the reader with introductions to the general subject area, both from a practical and theoretical perspective.

The history of trees goes on today with random forests (Breiman, 2001), which are on the list of the most successful machine learning algorithms currently available to handle large-scale and high-dimensional data sets. This method works according to the simple but effective *bagging* principle: sample fractions of the data, grow a predictor (a decision tree in the case of forests) on each small piece, and then paste the results together. Although the theory is still incomplete, random forests have been shown to give state-of-the-art performance on a number of practical problems and in different contexts (e.g., Meinshausen, 2006; Ishwaran et al., 2011). They work fast, generally exhibit a substantial improvement over single tree learners, and yield generalization error rates that often rank among the best (see, e.g., Fernández-Delgado et al., 2014). The surveys by Boulesteix et al. (2012) and Biau and Scornet (2016), which include practical guidelines and updated references, are a good starting point for understanding the method.

It is sometimes alluded to that forests have the flavor of deep network architectures (e.g., Bengio, 2009), insofar as ensemble of trees allow to discriminate between a very large number of regions. The richness of forest partitioning results from the fact that the number of intersections of the leaf regions can be exponential in the number of trees. That being said, the connection between random forests and neural networks is largely unexamined. On the one hand, the many parameters of neural networks make them a versatile and expressively rich tool for complex data modeling. However, their expressive power comes with the downside of increased overfitting risk, especially on small data sets. Conversely, random forests have fewer parameters to tune, but the greedy feature space separation by orthogonal hyperplanes results in typical stair or box-like decision surfaces, which may be advantageous for some data but suboptimal for other, particularly for colinear data with correlated features (Menze et al., 2011). In this context, the empirical studies by Welbl (2014) and Richmond et al. (2015) have highlighted the advantage of casting random forests into a neural network framework, with the intention to exploit the benefits of both approaches to overcome their respective shortcomings.

In view of the above, the objective of this article is to reformulate the random forest method into a neural network setting, and in turn propose two new hybrid procedures that we call *neural random forests*. In a nutshell, given an ensemble of random trees, it is possible to restructure them as a collection of (random) multilayered neural networks, which have sparse connections and less restrictions on the geometry of the decision boundaries. Their activation functions are soft nonlinear and differentiable, thus trainable with a gradient-based optimization algorithm and expected to exhibit better generalization performance. The idea of constructing a decision tree and using this tree to obtain a neural network is by no means new—see for example Sethi (1990, 1991), Brent (1991), and Devroye et al. (1996, Chapter 30). Similar work in the intersection between decision trees and neural networks has been undertaken by Kontschieder et al. (2015), who learn differentiable split functions to guide inputs through a tree. The *conditional networks* from Ioannou et al. (2016) also use trainable routing functions to perform conditional transformations on the inputs—they are thus capable of transferring computational efficiency benefits of decision trees into the domain of convolutional networks. However, to our best knowledge, no theoretical study has yet been reported regarding the connection between random forests and networks.

This paper makes several important contributions. First, in Section 2, we show that any regression tree can be seen as a particular neural network. In Section 3, we exploit this equivalence to combine neural networks in a random forest style and define the two neural random forest predictors. Both predictors exploit prior knowledge of regression trees for their initialization. This provides a major advantage in terms of interpretability (compared to more "black-box-like" neural network models) and effectively offers the networks a warm start at the prediction level of traditional forests. Section 4 is devoted to the derivation of theoretical consistency guarantees of the two methods. We illustrate in Section 5 the excellent performance of our approach both on simulated and real data sets, and show that it outperforms random forests in most situations. For clarity, the most technical proofs are gathered in Section 6.

2 Trees, forests, and networks

The general framework is nonparametric regression estimation, in which an input random vector $\mathbf{X} \in [0,1]^d$ is observed and the goal is to predict the square integrable random response $Y \in \mathbb{R}$ by estimating the regression function $r(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$. With this aim in mind, we assume we are given

a training sample $\mathscr{D}_n = ((\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n))$, $n \geq 2$, of independent random variables distributed the same as the independent prototype pair $(\mathbf{X}, Y)$. The data set $\mathscr{D}_n$ is used to construct an estimate $r(\cdot\,; \mathscr{D}_n) : [0,1]^d \to \mathbb{R}$ of the function r. We abbreviate $r(\mathbf{x}\,; \mathscr{D}_n)$ to $r_n(\mathbf{x})$ and say that the regression function estimate r_n is (mean squared error) consistent if $\mathbb{E}|r_n(\mathbf{X}) - r(\mathbf{X})|^2 \to 0$ as $n \to \infty$ (the expectation is evaluated over $\mathbf{X}$ and the sample $\mathscr{D}_n$).

2.1 From one tree to a neural network

A regression tree is a regression function estimate that uses a hierarchical segmentation of the input space, where each tree node corresponds to one of the segmentation subsets in $[0,1]^d$. In the following, we consider ordinary binary regression trees. In this model, a node has exactly either zero children (in which case it is called a terminal node or leaf) or two children. If a node u represents the set $A \subseteq [0,1]^d$ and its children u_L, u_R (L and R for *Left* and *Right*) represent $A_L \subseteq [0,1]^d$ and $A_R \subseteq [0,1]^d$, then we require that $A = A_L \cup A_R$ and $A_L \cap A_R = \emptyset$. The root represents the entire space $[0,1]^d$ and the leaves, taken together, form a partition of $[0,1]^d$. In an ordinary tree, we pass from A to A_L and A_R by answering a question on $\mathbf{x} = (x^{(1)}, \ldots, x^{(d)})$ of the form: "Is $x^{(j)} \geq \alpha$", for some dimension $j \in \{1, \ldots, d\}$ and some $\alpha \in [0,1]$. Thus, the feature space $[0,1]^d$ is partitioned into hyperrectangles whose sides are parallel to the coordinate axes. During prediction, the input is first passed into the tree root node. It is then iteratively transmitted to the child node that belongs to the subspace in which the input is located; this is repeated until a leaf node is reached. If a leaf represents region A, then the natural regression function estimate takes the simple form $t_n(\mathbf{x}) = (\sum_{i=1}^n Y_i \mathbb{1}_{\mathbf{X}_i \in A})/N_n(A)$, $\mathbf{x} \in A$, where $N_n(A)$ is the number of observations in cell A (where, by convention, $0/0 = 0$). In other words, the prediction for a query point $\mathbf{x}$ in leaf node A is the average of the Y_i of all training instances that fall into this region A. An example in dimension $d = 2$ is depicted in Figure 1.

The tree structure is usually data-dependent and indeed, it is in the construction itself that different trees differ. Of interest in the present paper is the CART program of Breiman et al. (1984), which will be described later on. Let us assume for now that we have at hand a regression tree t_n (whose construction eventually depends upon the data $\mathscr{D}_n$), which takes constant values on each of $K \geq 2$ terminal nodes. It turns out that this estimate may be reinterpreted as a three-layer neural network estimate with two hidden layers and one output layer, as summarized in the following. Let $\mathscr{H} = \{H_1, \ldots, H_{K-1}\}$

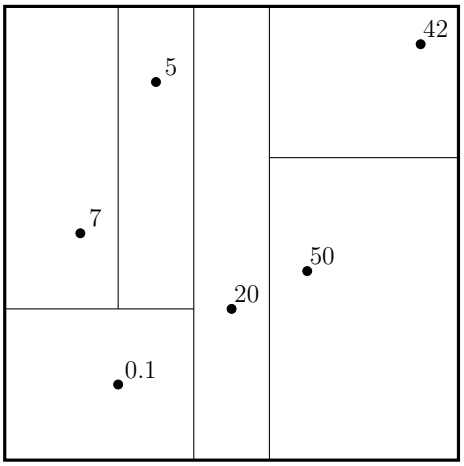

Figure 1: Tree partitioning in dimension $d = 2$, with $n = 6$ data points.

be the collection of all hyperplanes participating in the construction of t_n. We note that each $H_k \in \mathscr{H}$ is of the form $H_k = \{\mathbf{x} \in [0,1]^d : h_k(\mathbf{x}) = 0\}$, where $h_k(\mathbf{x}) = x^{(j_k)} - \alpha_{j_k}$ for some (eventually data-dependent) $j_k \in \{1, \ldots, d\}$ and $\alpha_{j_k} \in [0,1]$. To reach the leaf of the query point $\mathbf{x}$, we find, for each hyperplane H_k, the side on which $\mathbf{x}$ falls ($+1$ codes for right and -1 for left). With this notation, the tree estimate t_n is identical to the neural network described below.

First hidden layer. The first hidden layer of neurons corresponds to $K-1$ perceptrons (one for each inner tree node), whose activation is defined as

$$\tau(h_k(\mathbf{x})) = \tau(x^{(j_k)} - \alpha_{j_k}),$$

where $\tau(u) = 2\mathbb{1}_{u \geq 0} - 1$ is a threshold activation function. The weight vector is merely a single one-hot vector for feature j_k, and $-\alpha_{j_k}$ is the bias value. So, for each split in the tree, there is a neuron in layer 1 whose activity encodes the relative position of an input $\mathbf{x}$ with respect to the concerned split. In total, the first layer outputs the ± 1-vector $(\tau(h_1(\mathbf{x})), \ldots, \tau(h_{K-1}(\mathbf{x})))$, which describes all decisions of the inner tree nodes (including nodes off the tree path of $\mathbf{x}$). The quantity $\tau(h_k(\mathbf{x}))$ is $+1$ if $\mathbf{x}$ is on one side of the hyperplane H_k, -1 if $\mathbf{x}$ is on the other side of H_k, and by convention $+1$ if $\mathbf{x} \in H_k$. We again stress that each neuron k of this layer is connected to one, and only one, input $x^{(j_k)}$, and that this connection has weight 1 and offset $-\alpha_{j_k}$. An example is presented in Figure 2. Given these particular activations of layer 1, layer 2 can then easily reconstruct the precise tree leaf membership (i.e., the terminal cell) of $\mathbf{x}$.

Second hidden layer. Layer 1 outputs a $(K-1)$-dimensional vector of

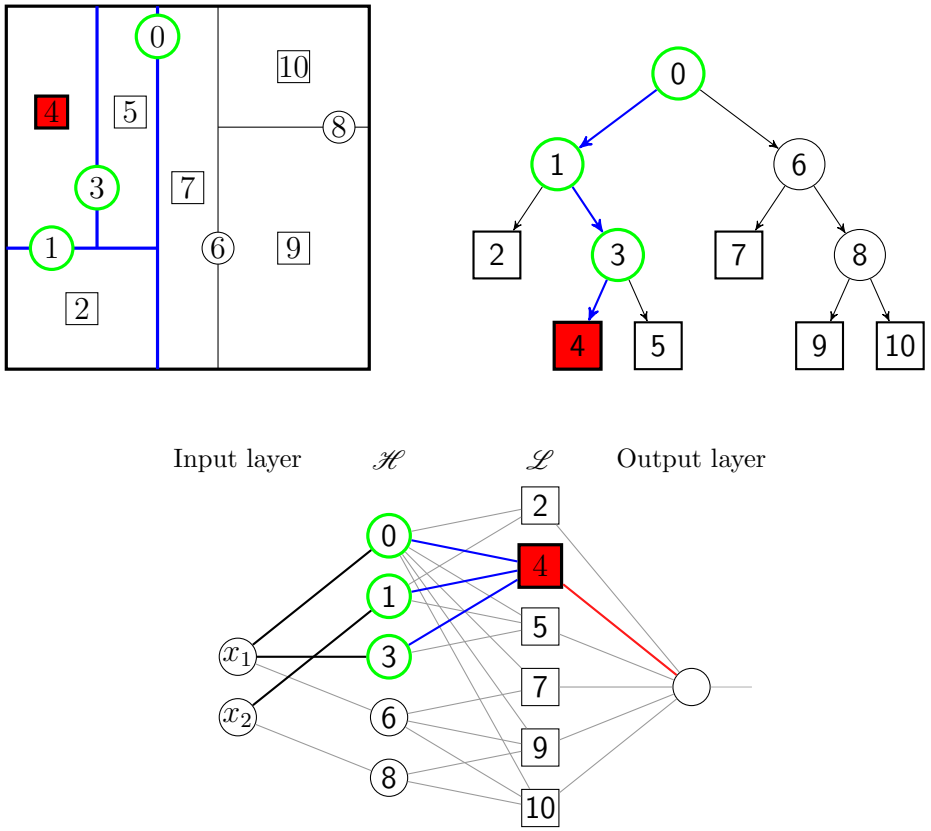

Figure 2: An example of regression tree (**top**) and the corresponding neural network (**down**).

± 1-bits that encodes for the precise location of $\mathbf{x}$ in the leaves of the tree. The leaf node identity of $\mathbf{x}$ can now be extracted from this vector using a weighted combination of the bits, together with an appropriate thresholding.

Let $\mathscr{L} = \{L_1, \ldots, L_K\}$ be the collection of all tree leaves, and let $L(\mathbf{x})$ be the leaf containing $\mathbf{x}$. The second hidden layer has K neurons, one for each leaf, and assigns a terminal cell to $\mathbf{x}$ as explained below. We connect a unit k from layer 1 to a unit k' from layer 2 if and only if the hyperplane H_k is involved in the sequence of splits forming the path from the root to the leaf $L_{k'}$. The connection has weight $+1$ if, in that path, the split by H_k is from a node to a right child, and -1 otherwise. So, if $(u_1(\mathbf{x}), \ldots, u_{K-1}(\mathbf{x}))$ is the vector of ± 1-bits seen at the output of layer 1, the output $v_{k'}(\mathbf{x}) \in \{-1, 1\}$ of neuron k' is $\tau(\sum_{k \to k'} b_{k,k'} u_k(\mathbf{x}) + b^0_{k'})$, where notation $k \to k'$ means that k is connected to k' and $b_{k,k'} = \pm 1$ is the corresponding weight. The offset

$b^0_{k'}$ is set to

$$b^0_{k'} = -\ell(k') + \frac{1}{2}, \tag{2.1}$$

where $\ell(k')$ is the length of the path from the root to $L_{k'}$. To understand the rationale behind the choice (2.1), observe that there are exactly $\ell(k')$ connections starting from the first layer and pointing to k', and that

$$\begin{cases} \sum_{k\to k'} b_{k,k'} u_k(\mathbf{x}) - \ell(k') + \frac{1}{2} = \frac{1}{2} & \text{if } \mathbf{x} \in L_{k'} \\ \sum_{k\to k'} b_{k,k'} u_k(\mathbf{x}) - \ell(k') + \frac{1}{2} \leq -\frac{1}{2} & \text{otherwise.} \end{cases} \tag{2.2}$$

Thus, with the choice (2.1), the argument of the threshold function is $1/2$ if $\mathbf{x} \in L_{k'}$ and is smaller than $-1/2$ otherwise. Hence $v_{k'}(\mathbf{x}) = 1$ if and only if the terminal cell of $\mathbf{x}$ is $L_{k'}$. To summarize, the second hidden layer outputs a vector of ± 1-bits $(v_1(\mathbf{x}), \ldots, v_K(\mathbf{x}))$ whose components equal -1 except the one corresponding to the leaf $L(\mathbf{x})$, which is $+1$.

Output layer. Let $(v_1(\mathbf{x}), \ldots, v_K(\mathbf{x}))$ be the output of the second hidden layer. If $v_{k'}(\mathbf{x}) = 1$, then the output layer computes the average $\bar{Y}_{k'}$ of the Y_i corresponding to $\mathbf{X}_i$ falling in $L_{k'}$. This is equivalent to take

$$t_n(\mathbf{x}) = \sum_{k'=1}^{K} w_{k'} v_{k'}(\mathbf{x}) + b_{\text{out}}, \tag{2.3}$$

where $w_{k'} = \frac{1}{2}\bar{Y}_{k'}$ for all $k' \in \{1, \ldots, K\}$, and $b_{\text{out}} = \frac{1}{2}\sum_{k'=1}^{K} \bar{Y}_{k'}$.

2.2 The CART program

We know from the preceding section that every regression tree may be seen as a neural network estimate with two hidden layers and threshold activation functions. Let us now be more precise about the details of the CART program of Breiman et al. (1984) to induce a regression tree. The core of their approach is a tree with K_n leaf regions defined by a partition of the space based on the n data points. When constructing the tree, the so-called CART-split criterion is applied recursively. This criterion determines which input direction should be used for the split and where the cut should be made. Let A be a generic cell and denote by $N_n(A)$ the number of examples falling in A. Formally, a cut in A is a pair (j, α), where j is a dimension from $\{1, \ldots, d\}$ and $\alpha \in [0, 1]$ is the position of the cut along the j-th coordinate, within the limits of A. Let $\mathscr{C}_A$ be the set of all such possible cuts in A. Then,

using the notation $\mathbf{X}_i = (\mathbf{X}_i^{(1)}, \ldots, \mathbf{X}_i^{(d)})$, the CART-split criterion takes the form, for any $(j, \alpha) \in \mathscr{C}_A$,

$$L_n(j,\alpha) = \frac{1}{N_n(A)} \sum_{i=1}^{n} (Y_i - \bar{Y}_A)^2 \mathbb{1}_{\mathbf{X}_i \in A} - \frac{1}{N_n(A)} \sum_{i=1}^{n} (Y_i - \bar{Y}_{A_L} \mathbb{1}_{\mathbf{X}_i^{(j)} < \alpha} - \bar{Y}_{A_R} \mathbb{1}_{\mathbf{X}_i^{(j)} \geq \alpha})^2 \mathbb{1}_{\mathbf{X}_i \in A}, \tag{2.4}$$

where $A_L = \{\mathbf{x} \in A : \mathbf{x}^{(j)} < \alpha\}$, $A_R = \{\mathbf{x} \in A : \mathbf{x}^{(j)} \geq \alpha\}$, and $\bar{Y}_A$ (resp., $\bar{Y}_{A_L}$, $\bar{Y}_{A_R}$) is the average of the Y_i belonging to A (resp., A_L, A_R), with the convention $0/0 = 0$. The quantity (2.4) measures the (renormalized) difference between the empirical variance in the node before and after a cut is performed. For each cell A, the best cut $(j_n^\star, \alpha_n^\star)$ is selected by maximizing $L_n(j, \alpha)$ over $\mathscr{C}_A$; that is,

$$(j_n^\star, \alpha_n^\star) \in \operatorname*{arg\,max}_{(j,\alpha) \in \mathscr{C}_A} L_n(j, \alpha). \tag{2.5}$$

(To remove some of the ties in the argmax, the best cut is always performed in the middle of two consecutive data points.) So, at each cell, the algorithm evaluates criterion (2.4) over all possible cuts in the d directions and returns the best one. This process is applied recursively down the tree, and stops when the tree contains exactly K_n terminal nodes, where $K_n \geq 2$ is an integer eventually depending on n.

2.3 Random forests

A random forest is a predictor consisting of a collection of M (large) randomized CART-type regression trees. For the m-th tree in the family, the predicted value at the query point $\mathbf{x}$ is denoted by $t(\mathbf{x}\,; \Theta_m, \mathscr{D}_n)$, where $\Theta_1, \ldots, \Theta_M$ are random variables, distributed the same as a generic random variable Θ. It is assumed that $\Theta_1, \ldots, \Theta_M$, Θ, and $\mathscr{D}_n$ are mutually independent. The variable Θ, which models the extra randomness introduced in each tree construction, is used to (i) resample the training set prior to the growing of individual trees, and (ii) select the successive directions for splitting via a randomized version of the CART criterion—see below. Lastly, the trees are combined to form the forest estimate

$$t(\mathbf{x}\,; \Theta_1, \ldots, \Theta_M, \mathscr{D}_n) = \frac{1}{M} \sum_{m=1}^{M} t(\mathbf{x}\,; \Theta_m, \mathscr{D}_n).$$

To lighten notation we write $t_{M,n}(\mathbf{x})$ instead of $t(\mathbf{x}\,;\Theta_1,\ldots,\Theta_M,\mathscr{D}_n)$.

The method works by growing the M randomized trees as follows. Let $a_n \geq 2$ be an integer smaller than or equal to n. Prior to the construction of each tree, a_n observations are drawn at random without replacement from the original data set; then, at each cell of the current tree, a split is performed by choosing uniformly at random, without replacement, a subset $\mathscr{M}_{\text{try}} \subseteq \{1,\ldots,d\}$ of cardinality $m_{\text{try}} := |\mathscr{M}_{\text{try}}|$, by evaluating criterion (2.4) over all possible cuts in the m_{try} directions, and by returning the best one. In other words, for each cell A, the best cut $(j_n^\star, \alpha_n^\star)$ is selected by maximizing $L_n(j,\alpha)$ over $\mathscr{M}_{\text{try}}$ and $\mathscr{C}_A$; that is,

$$(j_n^\star, \alpha_n^\star) \in \underset{\substack{j\in\mathscr{M}_{\text{try}}\\(j,\alpha)\in\mathscr{C}_A}}{\arg\max}\, L_n(j,\alpha). \tag{2.6}$$

The essential difference between (2.5) and (2.6) is that (2.6) is evaluated over a subset $\mathscr{M}_{\text{try}}$ of randomly selected coordinates, *not* over the whole range $\{1,\ldots,d\}$. The parameter m_{try}, which aims at reducing the computational burden and creating some diversity between the trees, is independent of n and often set to $d/3$. Also, because of the without-replacement sampling, each tree is constructed on a subset of a_n examples picked within the initial sample, *not* on the whole sample $\mathscr{D}_n$. In accordance with the CART program, the construction of individual trees is stopped when each tree reaches exactly K_n terminal nodes (recall that $K_n \in \{2,\ldots,a_n\}$ is a parameter of the algorithm). We insist on the fact that the number of leaves in each tree equals K_n. There are variants of the approach in terms of parameter choices and resampling mode that will not be explored here—see, e.g., the discussion in Biau and Scornet (2016) and the comments after Theorem 4.1.

Finally, by following the principles of Subsection 2.1, each random tree estimate $t(\cdot\,;\Theta_m,\mathscr{D}_n)$, $1 \leq m \leq M$, of the forest can be reinterpreted in the setting of neural networks. The M resulting networks are different because they correspond to different random trees. We see in particular that the m-th network has exactly $K_n - 1$ neurons in the first hidden layer and K_n in the second one. We also note that the network architecture (i.e., the way neurons are connected) and the associated coefficients depend on both Θ_m *and* $\mathscr{D}_n$.

3 Neural forests

Consider the m-th tree estimate $t(\cdot\,; \Theta_m, \mathscr{D}_n)$ of the forest, seen as a neural network estimate. Conditional on Θ_m and $\mathscr{D}_n$, the architecture of this network is fixed, and so are the weights and offsets of the three layers. A natural idea is then to keep the structure of the network intact and let the parameters vary in a subsequent network training procedure with backpropagation training. In other words, once the connections between the neurons have been designed by the tree-to-network mapping, we could then learn even better network parameters by minimizing some empirical mean squared error for this network over the sample $\mathscr{D}_n$. This additional training can potentially improve the predictions of the original random forest, and we will see more about this later in the experiments.

To allow for training based on gradient backpropagation, the activation functions must be differentiable. A natural idea is to replace the original relay-type activation function $\tau(u) = 2\mathbb{1}_{u \geq 0} - 1$ with a smooth approximation of it; for this the hyperbolic tangent activation function

$$\sigma(u) := \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}} = \frac{e^{2u} - 1}{e^{2u} + 1},$$

which has a range from -1 to 1 is chosen. More precisely, we use $\sigma_1(u) = \sigma(\gamma_1 u)$ at every neuron of the first hidden layer and $\sigma_2(u) = \sigma(\gamma_2 u)$ at every neuron of the second one. Here, γ_1 and γ_2 are positive hyperparameters that determine the contrast of the hyperbolic tangent activation: the larger γ_1 and γ_2, the sharper the transition from -1 to 1. Of course, as γ_1 and γ_2 approach infinity, the continuous functions σ_1 and σ_2 converge to the threshold function. Besides eventually providing better generalization, the hyperbolic tangent activation functions favor smoother decision boundaries and permit a relaxation of crisp tree node membership. Lastly, they allow to operate with a smooth approximation of the discontinuous step activation function. This makes the network loss function differentiable with respect to the parameters everywhere, and gradients can be backpropagated to train the network. Similar ideas are developed in the so-called soft tree models (Jordan and Jacobs, 1994; Olaru and Wehenkel, 2003; Geurts and Wehenkel, 2005; Yildiz and Alpaydin, 2013).

In this sparse setting, the number of parameters is much smaller than in a fully connected feed-forward network with the same number of neurons. Recall that the two hidden layers have respectively $K_n - 1$ and K_n hidden nodes. Without any information regarding the connections between neurons

(fully connected network), fitting such a network would require optimizing a total of $(d+1)(K_n-1)+K_n^2+K_n+1$ parameters, which is $\mathscr{O}(dK_n+K_n^2)$. On the other hand, assuming that the tree generated by the CART algorithm is roughly balanced, then the average depth of the tree is $\mathscr{O}(\log K_n)$. This gives, on average, $2(K_n-1)+\mathscr{O}(K_n \log K_n)+K_n+1$ parameters to fit, which is $\mathscr{O}(K_n \log K_n)$. For large K_n, this quantity can be much smaller than $\mathscr{O}(dK_n+K_n^2)$ and, in any case, it is independent of the dimension d—a major computational advantage in high-dimensional settings. If an unbalanced tree does occur, then it has at most $\mathscr{O}(K_n)$ levels and the total work required is $\mathscr{O}(K_n^2)$. Thus, even in this extreme case, our algorithm is independent of d and should be competitive in high dimensions, when $d \gg K_n$.

Remark 3.1. *Training a network with sparse connectivity retains some degree of interpretability of its internal representation and may be seen as a relaxation of original tree structures. But besides these sparse networks, the relaxation of tree structures can go even further when allowing for full connectivity between layers—in this case, the tree structure is merely used as initialization for a fully connected network of the same size. In this setting, all weights belonging to tree structures have a nonzero initialization value, while the other weights start with 0. During training then, all weights can be modified so that arbitrary connections between the layers (i.e., also across trees) can be learned. The initial tree-type parametrization provides a strong inductive bias, compared to a random initialization, which contains valuable information and mimics the regression function of a CART-type tree already before backpropagation training. Compared to a random initialization, this gives the network an effective warm start. The effects of using random forests as warm starts for neural networks will be further explored in the experimental Section 5.*

Thus, the above mechanisms allow to convert the M CART-type trees into M tree-type neural networks that have different sparse architectures and smooth activation functions at the nodes. Interestingly, this link opens up a principled way for inferring the structure of a neural network, by viewing a regression tree as a specially structured network and putting some kind of prior information about the weights.

To complete the presentation, it remains to explain how to combine the M individual networks. This can be done in at least two different ways, which are described below. We call the two resulting estimates *neural forests*.

Method 1: Independent training. The parameters of each tree-type network are fitted network by network, independently of each other. With

this approach, we end up with an ensemble of M "small" neural network estimates $r(\cdot\,;\Theta_m, \mathscr{D}_n)$, $1 \leq m \leq M$, which are finally averaged to form the estimate

$$r(\mathbf{x}\,;\Theta_1, \ldots, \Theta_M, \mathscr{D}_n) = \frac{1}{M}\sum_{m=1}^{M} r(\mathbf{x}\,;\Theta_m, \mathscr{D}_n) \tag{3.1}$$

(see the illustration in Figure 3). To lighten notation we write $r_{M,n}(\mathbf{x})$ instead of the more complicated $r(\mathbf{x}\,;\Theta_1, \ldots, \Theta_M, \mathscr{D}_n)$, keeping in mind that $r_{M,n}(\mathbf{x})$ depends on both $\Theta_1, \ldots, \Theta_M$ and the sample $\mathscr{D}_n$.

The minimization program implemented at each "small" network is described in Section 4 below, together with the statistical properties of $r_{M,n}(\mathbf{x})$. This predictor has the flavor of a randomized ensemble (forest) of neural networks.

Method 2: Joint training. In this approach, the individual tree networks are first concatenated into one single "big" network, as shown in Figure 4. The parameters of the resulting "big" network are then fitted jointly in one optimization procedure over the whole network.

Although the hidden layers of the "small" networks are not connected across the sections belonging to each tree, this algorithm has two main differences with the previous one:

(i) The output layer, which is shared by all "small" networks, computes a combination of *all* outputs of the second-layer neurons.

(ii) The optimization is performed in one single run over the whole "big" network, and not network by network. Assuming that the trees are balanced, the first method performs, on average, M different optimization programs in a space of $\mathscr{O}(K_n \log K_n)$ parameters, whereas the second one accomplishes only one minimization in a space of average dimension $\mathscr{O}(MK_n \log K_n)$.

In the sequel, we let $s_{M,n}(\mathbf{x})$ be the regression function estimate corresponding to the second method. We note that this estimate still depends upon $\Theta_1, \ldots, \Theta_M$ and $\mathscr{D}_n$, but is *not* of the averaging form (3.1). It will be more formally described in the next section.

Remark 3.2. *The approach presented in this paper is based on the interpretation of trees as two-layer neural networks. This link does not immediately extend to more complex networks, especially those with more than two layers. One idea would be to artificially increase the depth of the networks by adding a collection of layers such that each newly added layer leaves the previous one intact. Nevertheless, such an architecture would probably be very difficult to optimize because of the numerous connections playing the same role.*

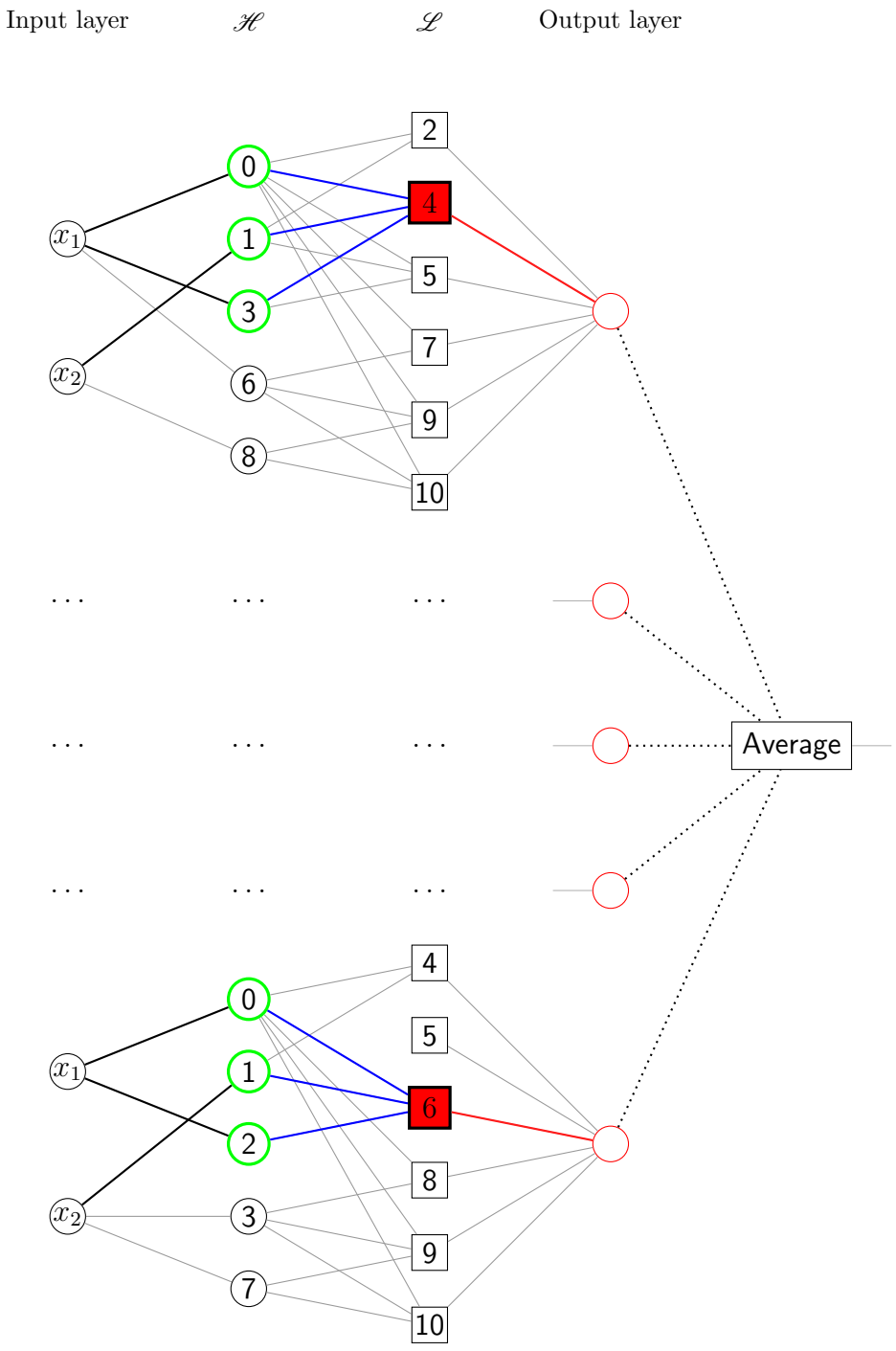

Figure 3: Method 1: Independent training.

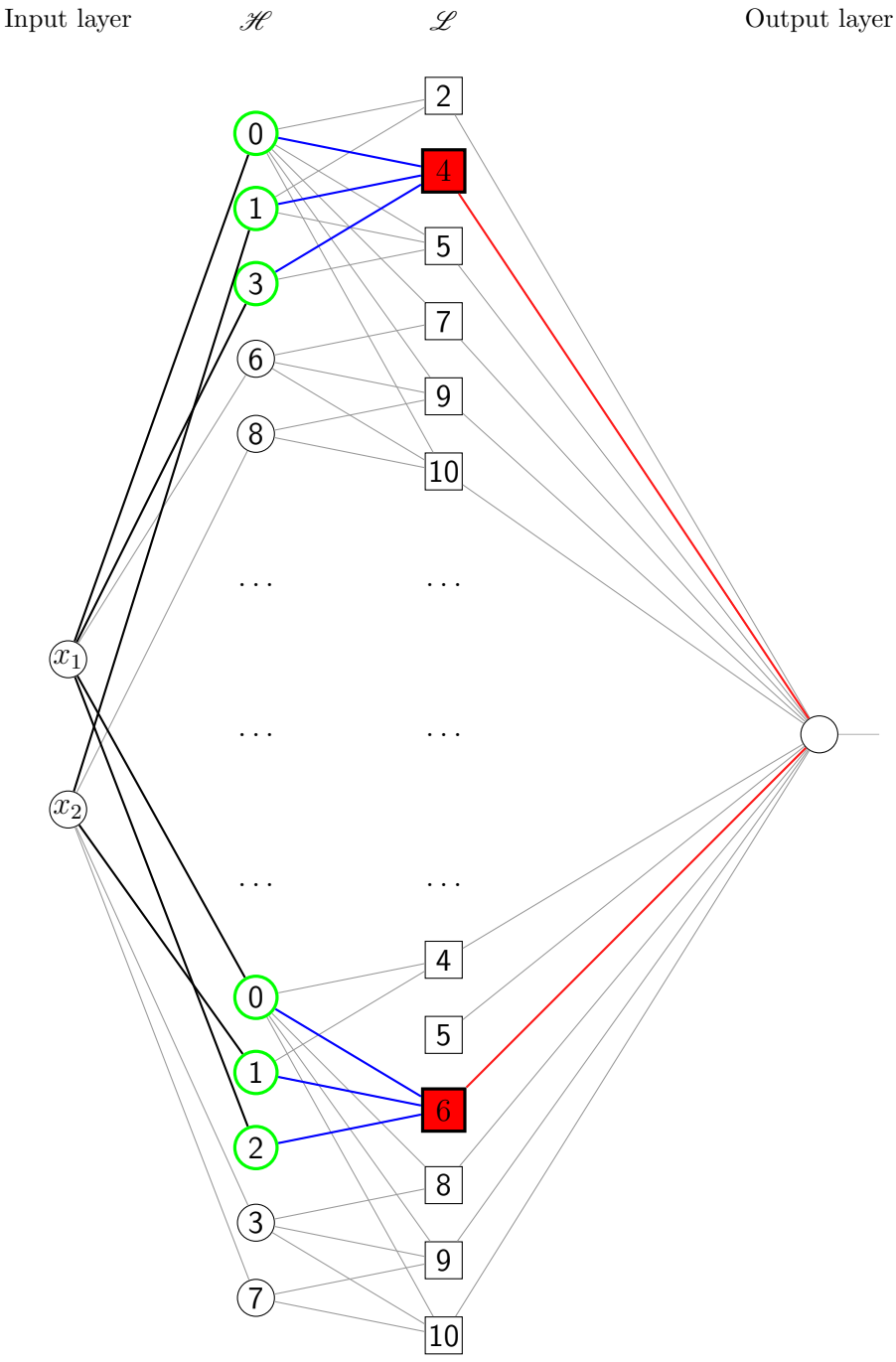

Figure 4: Method 2: Joint training.

4 Some theory

4.1 Empirical risk minimization

We now describe in full detail the construction of the two regression function estimates $r_{M,n}$ (**independent training**) and $s_{M,n}$ (**joint training**).

Method 1: Independent training. Consider the m-th random tree in the ensemble and denote by $\mathscr{G}_1 \equiv \mathscr{G}_1(\Theta_m, \mathscr{D}_n)$ the bipartite graph modeling the connections between the vector of inputs $\mathbf{x} = (x^{(1)}, \ldots, x^{(d)})$ and the $K_n - 1$ hidden neurons of the first layer. Similarly, let $\mathscr{G}_2 \equiv \mathscr{G}_2(\Theta_m, \mathscr{D}_n)$ be the bipartite graph representing the connections between the first layer and the K_n hidden neurons of the second layer.

Let $\mathbb{M}(\mathscr{G}_1)$ be the set of $d \times (K_n - 1)$ matrices $\mathbf{W}_1 = (a_{ij})$ such that $a_{ij} = 0$ if $(i, j) \notin \mathscr{G}_1$, and let $\mathbb{M}(\mathscr{G}_2)$ be the $(K_n - 1) \times K_n$ matrices $\mathbf{W}_2 = (b_{ij})$ such that $b_{ij} = 0$ if $(i, j) \notin \mathscr{G}_2$. The parameters that specify the first hidden units are encapsulated in a matrix $\mathbf{W}_1$ of $\mathbb{M}(\mathscr{G}_1)$ of weights over the edges of $\mathscr{G}_1$ and by a column vector of biases $\mathbf{b}_1$, of size $K_n - 1$. Similarly, the parameters of the second hidden units are represented by a matrix $\mathbf{W}_2$ of $\mathbb{M}(\mathscr{G}_2)$ of weights over $\mathscr{G}_2$ and by a column vector $\mathbf{b}_2$ of offsets, of size K_n. Finally, we let the output weights and offset be $\mathbf{W}_{\text{out}} = (w_1, \ldots, w_{K_n})^\top \in \mathbb{R}^{K_n}$ and $b_{\text{out}} \in \mathbb{R}$, respectively ($\top$ denotes transposition and vectors are in column format).

Thus, the parameters that specify the m-th network are represented by a "vector"

$$\begin{aligned}\boldsymbol{\lambda} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_{\text{out}}, b_{\text{out}}) \\ \in \mathbb{M}(\mathscr{G}_1) \times \mathbb{R}^{K_n - 1} \times \mathbb{M}(\mathscr{G}_2) \times \mathbb{R}^{K_n} \times \mathbb{R}^{K_n} \times \mathbb{R}.\end{aligned}$$

However, in order to obtain consistency we have to restrict the range of variation for these parameters. For a given matrix M, the notation $|M|$ means the matrix of absolute values of the entries of M. We assume that there exists a positive constant C_1 such that

$$\|\mathbf{W}_2\|_\infty + \|\mathbf{b}_2\|_\infty + \|\mathbf{W}_{\text{out}}\|_1 + |b_{\text{out}}| \leq C_1 K_n, \tag{4.1}$$

where $\|\cdot\|_\infty$ denotes the supremum norm of matrices and $\|\cdot\|_1$ is the L_1-norm of vectors. In other words, it is basically assumed that the weights and offsets (resp., the sum of absolute values of the weights and the offset) absorbed by the computation units of the second layer (resp., the output layer) are at most of the magnitude of K_n. We emphasize that this requirement is mild

and that it leaves a lot of freedom for optimizing the parameters. We note in particular that it is satisfied by the original random tree estimates as soon as Y is almost surely bounded, with the choice $C_1 = (\frac{3}{2} + \|Y\|_\infty)$ ($\|Y\|_\infty$ is the essential supremum of Y).

Therefore, letting

$$\Lambda(\Theta_m, \mathscr{D}_n) = \big\{\boldsymbol{\lambda} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_{\text{out}}, b_{\text{out}}) : (4.1) \text{ is satisfied}\big\},$$

we see that the m-th neural network implements functions of the form

$$f_{\boldsymbol{\lambda}}(\mathbf{x}) = \mathbf{W}_{\text{out}}^\top \sigma_2\Big(\mathbf{W}_2^\top \sigma_1(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2\Big) + b_{\text{out}}, \quad \mathbf{x} \in \mathbb{R}^d,$$

where $\lambda \in \Lambda(\Theta_m, \mathscr{D}_n)$, and σ_1 and σ_2 are applied element-wise. Our aim is to adjust the parameter $\boldsymbol{\lambda}$ using the data $\mathscr{D}_n$ such that the function realized by the obtained network is a good estimate of r. Let

$$\mathscr{F}(\Theta_m, \mathscr{D}_n) = \big\{f_{\boldsymbol{\lambda}} : \boldsymbol{\lambda} \in \Lambda(\Theta_m, \mathscr{D}_n)\big\}.$$

For each $m \in \{1, \dots, M\}$, our algorithm constructs a regression function estimate $r(\cdot\,; \Theta_m, \mathscr{D}_n)$ by minimizing the empirical error

$$J_n(f) = \frac{1}{n} \sum_{i=1}^{n} \big|Y_i - f(\mathbf{X}_i)\big|^2$$

over functions f in $\mathscr{F}(\Theta_m, \mathscr{D}_n)$, that is

$$J_n\big(r(\cdot\,; \Theta_m, \mathscr{D}_n)\big) \le J_n(f) \quad \text{for all } f \in \mathscr{F}(\Theta_m, \mathscr{D}_n).$$

Remark 4.1. *Here we assumed the existence of a minimum, though not necessarily its uniqueness. In cases where a minimum does not exist, the same analysis can be carried out with functions whose error is arbitrarily close to the infimum, but for the sake of simplicity we stay with the assumption of existence throughout the paper. Note also that we do not investigate the properties of the gradient descent algorithm used in Section 5, and assume instead that the global minimum (if it exists) can be computed.*

By repeating this minimization process for each $m \in \{1, \dots, M\}$, we obtain a collection of (randomized) estimates $r(\cdot\,; \Theta_1, \mathscr{D}_n), \dots, r(\cdot\,; \Theta_M, \mathscr{D}_n)$, which are aggregated to form the estimate

$$r_{M,n}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} r(\mathbf{x}\,; \Theta_m, \mathscr{D}_n).$$

The estimate $r_{M,n}$ is but a generalization of the random forest estimate $t_{M,n}$ to the neural network framework, with an additional relaxation of crisp to fuzzy tree node membership due to the hyperbolic tangent activation functions: samples not merely fall into one direction per split and one final leaf but simultaneously into several tree branches and leaves.

Method 2: Joint training. The notation needed to describe the second approach is a bit burdensome, but the ideas are simple. Following the above, we denote by $\mathscr{G}_{1,1}, \ldots, \mathscr{G}_{1,M}$ and $\mathscr{G}_{2,1}, \ldots, \mathscr{G}_{2,M}$ the bipartite graphs associated with the M "small" original random trees. We also let $\mathbf{W}_{1,1}, \ldots, \mathbf{W}_{1,M}$ and $\mathbf{b}_{1,1}, \ldots, \mathbf{b}_{1,M}$ be the respective weight matrices and offset vectors of the first hidden layers of the M "small" networks, with $\mathbf{W}_{1,m} \in \mathbb{M}(\mathscr{G}_{1,m})$ and $\mathbf{b}_{1,m} \in \mathbb{R}^{K_n - 1}$, $1 \leq m \leq M$. Similarly, we denote by $\mathbf{W}_{2,1}, \ldots, \mathbf{W}_{2,M}$ and $\mathbf{b}_{2,1}, \ldots, \mathbf{b}_{2,M}$ the respective weight matrices and offset vectors of the second layer, with $\mathbf{W}_{2,m} \in \mathbb{M}(\mathscr{G}_{2,m})$ and $\mathbf{b}_{2,m} \in \mathbb{R}^{K_n}$, $1 \leq m \leq M$.

Next, we form the concatenated matrices $[\mathbf{W}_1]$, $[\mathbf{b}_1]$, $[\mathbf{W}_2]$, and $[\mathbf{b}_2]$, defined by

$$[\mathbf{W}_1] = \begin{pmatrix} \mathbf{W}_{1,1} & \cdots & \mathbf{W}_{1,M} \end{pmatrix}, \quad [\mathbf{b}_1] = \begin{pmatrix} \mathbf{b}_{1,1} \\ \vdots \\ \mathbf{b}_{1,M} \end{pmatrix},$$

and

$$[\mathbf{W}_2] = \begin{pmatrix} \mathbf{W}_{2,1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{2,2} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{W}_{2,M-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{W}_{2,M} \end{pmatrix}, \quad [\mathbf{b}_2] = \begin{pmatrix} \mathbf{b}_{2,1} \\ \vdots \\ \mathbf{b}_{2,M} \end{pmatrix}.$$

Notice that $[\mathbf{W}_1]$, $[\mathbf{b}_1]$, $[\mathbf{W}_2]$, and $[\mathbf{b}_2]$ are of size $d \times M(K_n - 1)$, $M(K_n - 1) \times 1$, $M(K_n - 1) \times MK_n$, and $MK_n \times 1$, respectively. Let us finally denote by $\mathbf{W}_{\text{out}} \in \mathbb{R}^{MK_n}$ and $b_{\text{out}} \in \mathbb{R}$ the output weights and offset of the concatenated network. All in all, the parameters of the network are represented by a "vector"

$$[\boldsymbol{\lambda}] = \big([\mathbf{W}_1], [\mathbf{b}_1], [\mathbf{W}_2], [\mathbf{b}_2], \mathbf{W}_{\text{out}}, b_{\text{out}}\big),$$

where $[\mathbf{W}_1]$, $[\mathbf{b}_1]$, $[\mathbf{W}_2]$, and $[\mathbf{b}_2]$ are defined above. As in the first method, we restrict the range of variation of these parameters and assume that there exists a positive constant C_2 such that

$$\big\|[\mathbf{W}_2]\big\|_\infty + \big\|[\mathbf{b}_2]\big\|_\infty + \|\mathbf{W}_{\text{out}}\|_1 + |b_{\text{out}}| \leq C_2 K_n. \tag{4.2}$$

Therefore, letting

$$\begin{aligned}&\Lambda(\Theta_1,\dots,\Theta_M,\mathscr{D}_n)\\&\qquad=\big\{[\boldsymbol{\lambda}]=\big([\mathbf{W}_1],[\mathbf{b}_1],[\mathbf{W}_2],[\mathbf{b}_2],\mathbf{W}_{\text{out}},b_{\text{out}}\big):(4.2)\text{ is satisfied}\big\},\end{aligned}$$

the "big" network implements functions of the form

$$f_{[\boldsymbol{\lambda}]}(\mathbf{x})=\mathbf{W}_{\text{out}}^{\top}\sigma_2\Big([\mathbf{W}_2]^{\top}\sigma_1\big([\mathbf{W}_1]^{\top}\mathbf{x}+[\mathbf{b}_1]\big)+[\mathbf{b}_2]\Big)+b_{\text{out}},\quad \mathbf{x}\in\mathbb{R}^d,$$

where $[\boldsymbol{\lambda}]\in\Lambda(\Theta_1,\dots,\Theta_M,\mathscr{D}_n)$, and σ_1 and σ_2 are applied element-wise. Next, let

$$\mathscr{F}(\Theta_1,\dots,\Theta_M,\mathscr{D}_n)=\big\{f_{[\boldsymbol{\lambda}]}:[\boldsymbol{\lambda}]\in\Lambda(\Theta_1,\dots,\Theta_M,\mathscr{D}_n)\big\}.$$

Then the final estimate $s_{M,n}$ is obtained by minimizing the empirical error

$$J_n(f)=\frac{1}{n}\sum_{i=1}^{n}\big|Y_i-f(\mathbf{X}_i)\big|^2$$

over functions f in $\mathscr{F}(\Theta_1,\dots,\Theta_M,\mathscr{D}_n)$, that is

$$J_n(s_{M,n})\le J_n(f)\quad\text{for all } f\in\mathscr{F}(\Theta_1,\dots,\Theta_M,\mathscr{D}_n).$$

4.2 Consistency

In order to analyze the consistency properties of the regression function estimates $r_{M,n}$ and $s_{M,n}$, we first need to consider some specific class $\mathscr{F}$ of functions over $[0,1]^d$. It is defined as follows.

For a hyperrectangle $A=[a_1,b_1]\times\cdots\times[a_d,b_d]\subseteq[0,1]^d$, we let $A^{\backslash j}=\prod_{i\neq j}[a_i,b_i]$ and $\mathrm{d}\mathbf{x}^{\backslash j}=\mathrm{d}x_1\dots\mathrm{d}x_{j-1}\mathrm{d}x_{j+1}\dots\mathrm{d}x_d$. Assume we are given a measurable function $f:[0,1]^d\to\mathbb{R}$ together with $A=[a_1,b_1]\times\cdots\times[a_d,b_d]\subseteq[0,1]^d$, and consider the following two statements:

(i) For any $j\in\{1,\dots,d\}$, the function

$$x_j\mapsto\int_{A^{\backslash j}}f(\mathbf{x})\mathrm{d}\mathbf{x}^{\backslash j}$$

is constant on $[a_j,b_j]$;

(ii) The function f is constant on A.

Definition 4.1. *We let $\mathscr{F}$ be the class of continuous real functions on $[0,1]^d$ such that, for any $A = [a_1, b_1] \times \cdots \times [a_d, b_d] \subseteq [0,1]^d$, (i) implies (ii).*

Although the membership requirement for $\mathscr{F}$ seems at first glance a bit restrictive, it turns out that $\mathscr{F}$ is in fact a rich class of functions. For example, additive functions of the form

$$f(\mathbf{x}) = \sum_{j=1}^{d} f_j(x^{(j)}),$$

where each f_j is continuous, do belong to $\mathscr{F}$. This is also true for polynomial functions whose coefficients have the same sign. Also, product of continuous functions of the form

$$f(\mathbf{x}) = \prod_{j=1}^{d} f_j(x^{(j)}),$$

where, for all $j \in \{1, \ldots, d\}$, $[f_j > 0 \text{ or } f_j < 0]$, are included in $\mathscr{F}$.

Our main theorem states that the neural forest estimates $r_{M,n}$ and $s_{M,n}$ are consistent, provided the number K_n of terminal nodes and the parameters γ_1 and γ_2 are properly regulated as functions of n.

Theorem 4.1 (Consistency of $r_{M,n}$ and $s_{M,n}$). *Assume that $\mathbf{X}$ is uniformly distributed in $[0,1]^d$, $\|Y\|_\infty < \infty$, and $r \in \mathscr{F}$. Assume, in addition, that $K_n, \gamma_1, \gamma_2 \to \infty$ such that, as n tends to infinity,*

$$\frac{K_n^6 \log(\gamma_2 K_n^5)}{n} \to 0, \quad K_n^2 e^{-2\gamma_2} \to 0, \quad \text{and} \quad \frac{K_n^4 \gamma_2^2 \log(\gamma_1)}{\gamma_1} \to 0.$$

Then, as n tends to infinity,

$$\mathbb{E}\big|r_{M,n}(\mathbf{X}) - r(\mathbf{X})\big|^2 \to 0 \quad \text{and} \quad \mathbb{E}\big|s_{M,n}(\mathbf{X}) - r(\mathbf{X})\big|^2 \to 0.$$

It is interesting to note that Theorem 4.1 still holds when the individual trees are subsampled and fully grown (that is, when $K_n = a_n$, i.e., one single observation in each leaf) as soon as the assumptions are satisfied with a_n instead of K_n. Put differently, we require that the trees of the forest are either pruned (restriction on K_n) or subsampled (restriction on a_n). If not, the assumptions of Theorem 4.1 are violated. So, to obtain a consistent prediction, it is therefore mandatory to keep the depth of the tree or the size of subsamples under control. We also note that Theorem 4.1 can be

adapted to deal with networks based on bootstrapped and fully grown trees. In this case, n observations are chosen in $\mathscr{D}_n$ *with* replacement prior to each tree construction, and only one distinct example is left in the leaves. In this setting, care must be taken in the analysis to consider only the trees that use at least K_n distinct data points—we leave the adaptation as a small exercise.

Let us finally point out that the proof of Theorem 4.1 relies on the consistency of the individual "small" networks. Therefore, the proposed analysis does not really highlight the benefits of aggregating individual trees in terms of finite-sample analysis or asymptotic behavior. Undoubtedly, there is room for further improvement.

4.3 Proof of Theorem 4.1

Consistency of $r_{M,n}$. Denote by μ the distribution of $\mathbf{X}$. The consistency proof of $r_{M,n}$ starts with the observation that

$$\begin{aligned}
\mathbb{E}\big|r_{M,n}(\mathbf{X}) - r(\mathbf{X})\big|^2 &= \mathbb{E}\left|\frac{1}{M}\sum_{m=1}^{M} r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n) - r(\mathbf{X})\right|^2 \\
&\le \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}\big|r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n) - r(\mathbf{X})\big|^2 \\
&\quad \text{(by Jensen's inequality)} \\
&= \mathbb{E}\big|r(\mathbf{X}\,;\Theta,\mathscr{D}_n) - r(\mathbf{X})\big|^2.
\end{aligned}$$

Therefore, we only need to show that, under the conditions of the theorem, $\mathbb{E}|r(\mathbf{X}\,;\Theta,\mathscr{D}_n) - r(\mathbf{X})|^2 \to 0$, i.e., that a single random network estimate is consistent (note that the expectation is taken with respect to $\mathbf{X}$, Θ, and $\mathscr{D}_n$).

We have (see, e.g., Lugosi and Zeger, 1995, or Györfi et al., 2002, Lemma 10.1)

$$\begin{aligned}
&\mathbb{E}\big|r(\mathbf{X}\,;\Theta,\mathscr{D}_n) - r(\mathbf{X})\big|^2 \\
&\quad\le 2\mathbb{E}\sup_{f\in\mathscr{F}(\Theta,\mathscr{D}_n)}\left|\frac{1}{n}\sum_{i=1}^{n}\big|Y_i - f(\mathbf{X}_i)\big|^2 - \mathbb{E}\big|Y - f(\mathbf{X})\big|^2\right| \\
&\qquad + \mathbb{E}\inf_{f\in\mathscr{F}(\Theta,\mathscr{D}_n)}\int_{[0,1]^d}\big|f(\mathbf{x}) - r(\mathbf{x})\big|^2\mu(\mathrm{d}\mathbf{x}). \qquad (4.3)
\end{aligned}$$

The first term—the estimation error—is handled in Proposition 4.1 below by using nonasymptotic uniform deviation inequalities and covering numbers corresponding to $\mathscr{F}(\Theta,\mathscr{D}_n)$ (proofs are in Section 6).

Proposition 4.1. *Assume that* $K_n, \gamma_2 \to \infty$ *such that* $K_n^6 \log(\gamma_2 K_n^5)/n \to 0$. *Then*

$$\mathbb{E} \sup_{f \in \mathscr{F}(\Theta, \mathscr{D}_n)} \left| \frac{1}{n} \sum_{i=1}^{n} \left|Y_i - f(\mathbf{X}_i)\right|^2 - \mathbb{E}\left|Y - f(\mathbf{X})\right|^2 \right| \to 0 \quad \text{as } n \to \infty.$$

To deal with the second term of the right-hand side of (4.3)—the approximation error—, we consider a piecewise constant function (pseudo-estimate) similar to the original CART-tree $t_n(.\,; \Theta, \mathscr{D}_n)$, with only one difference: the function computes the true conditional expectation $\mathbb{E}[Y|\mathbf{X} \in L_{k'}]$ in each leaf $L_{k'}$, not the empirical one $\bar{Y}_{k'}$. Put differently, we take $(\mathbf{W}_{\text{out}}^\star)_{k'} = \mathbb{E}[Y|\mathbf{X} \in L_{k'}]/2$ and $b_{\text{out}}^\star = \sum_{k'=1}^{K_n} \mathbb{E}[Y|\mathbf{X} \in L_{k'}]/2$ in (2.3). This tree-type pseudo-estimate has the form

$$t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) = \mathbf{W}_{\text{out}}^{\star\top} \tau\Big(\mathbf{W}_2^{\star\top} \tau(\mathbf{W}_1^{\star\top} \mathbf{x} + \mathbf{b}_1^\star) + \mathbf{b}_2^\star\Big) + b_{\text{out}}^\star, \quad \mathbf{x} \in \mathbb{R}^d,$$

(recall that $\tau(u) = 2\mathbb{1}_{u \geq 0} - 1$), for some $\boldsymbol{\lambda}^\star = (\mathbf{W}_1^\star, \mathbf{b}_1^\star, \mathbf{W}_2^\star, \mathbf{b}_2^\star, \mathbf{W}_{\text{out}}^\star, b_{\text{out}}^\star)$. We have

$$\begin{aligned}
&\inf_{f \in \mathscr{F}(\Theta, \mathscr{D}_n)} \int_{[0,1]^d} \left|f(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}) \\
&\quad = \inf_{f \in \mathscr{F}(\Theta, \mathscr{D}_n)} \int_{[0,1]^d} \left|f(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}) - 2 \int_{[0,1]^d} \left|t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}) \\
&\qquad + 2 \int_{[0,1]^d} \left|t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}).
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\inf_{f \in \mathscr{F}(\Theta, \mathscr{D}_n)} \int_{[0,1]^d} \left|f(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}) \\
&\quad \leq \int_{[0,1]^d} \left|f_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}) - 2 \int_{[0,1]^d} \left|t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}) \\
&\qquad + 2 \int_{[0,1]^d} \left|t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}) \\
&\quad \leq 2 \int_{[0,1]^d} \left|f_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - t_{\boldsymbol{\lambda}^\star}(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}) + 2 \int_{[0,1]^d} \left|t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - r(\mathbf{x})\right|^2 \mu(\mathrm{d}\mathbf{x}).
\end{aligned}$$

We prove in Proposition 4.2 that the expectation of the first of the two terms above tends to zero under appropriate conditions on γ_1 and γ_2. The second term is less standard and requires a careful analysis of the asymptotic geometric behavior of the cells of the tree pseudo-estimate $t_{\boldsymbol{\lambda}^\star}$. This is the topic of Proposition 4.3. Taken together, inequality (4.3) and Proposition 4.1-4.3 prove the result.

Proposition 4.2. *Assume that* $\mathbf{X}$ *is uniformly distributed in* $[0,1]^d$ *and* $\|Y\|_\infty < \infty$. *Assume, in addition, that* $K_n, \gamma_1, \gamma_2 \to \infty$ *such that*

$$K_n^2 e^{-2\gamma_2} \to 0 \quad \textit{and} \quad K_n^4 \gamma_2^2 \log(\gamma_1)/\gamma_1 \to 0.$$

Then

$$\mathbb{E} \int_{[0,1]^d} \left| f_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) \right|^2 \mu(\mathrm{d}\mathbf{x}) \to 0 \quad \textit{as } n \to \infty.$$

Proposition 4.3. *Assume that* $\mathbf{X}$ *is uniformly distributed in* $[0,1]^d$ *and* $\|Y\|_\infty < \infty$. *If* $r \in \mathscr{F}$, *then*

$$\mathbb{E} \int_{[0,1]^d} \left| t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - r(\mathbf{x}) \right|^2 \mu(\mathrm{d}\mathbf{x}) \to 0 \quad \textit{as } n \to \infty.$$

Consistency of $s_{M,n}$. The consistency of $s_{M,n}$ is a consequence of the first statement of Theorem 4.1. Denote by $\widehat{\mathbf{W}}_{1,1}, \ldots, \widehat{\mathbf{W}}_{1,M}$, $\widehat{\mathbf{b}}_{1,1}, \ldots, \widehat{\mathbf{b}}_{1,M}$, $\widehat{\mathbf{W}}_{2,1}, \ldots, \widehat{\mathbf{W}}_{2,M}$, $\widehat{\mathbf{b}}_{2,1}, \ldots, \widehat{\mathbf{b}}_{2,M}$, $\widehat{\mathbf{W}}_{\mathrm{out},1}, \ldots, \widehat{\mathbf{W}}_{\mathrm{out},M}$, and $\widehat{b}_{\mathrm{out},1}, \ldots, \widehat{b}_{\mathrm{out},M}$ the set of parameters (weights and offsets) output by the minimization programs performed at each tree-type network of **Method 1**. It is then easy to see that the "big" network fitted with the parameters

$$[\widehat{\boldsymbol{\lambda}}] = \big([\widehat{\mathbf{W}}_1], [\widehat{\mathbf{b}}_1], [\widehat{\mathbf{W}}_2], [\widehat{\mathbf{b}}_2], [\widehat{\mathbf{W}}_{\mathrm{out}}], \widehat{b}_{\mathrm{out}}\big),$$

where

$$[\widehat{\mathbf{W}}_{\mathrm{out}}] = \frac{1}{M} \begin{pmatrix} \widehat{\mathbf{W}}_{\mathrm{out},1} \\ \vdots \\ \widehat{\mathbf{W}}_{\mathrm{out},M} \end{pmatrix}$$

and $\widehat{b}_{\mathrm{out}} = \frac{1}{M} \sum_{m=1}^M \widehat{b}_{\mathrm{out},m}$, exactly computes the function $r_{M,n}$. This implies

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n \left| Y_i - s_{M,n}(\mathbf{X}_i) \right|^2 &\leq \frac{1}{n} \sum_{i=1}^n \left| Y_i - r_{M,n}(\mathbf{X}_i) \right|^2 \\ &\leq \frac{1}{M} \sum_{m=1}^M \Big(\frac{1}{n} \sum_{i=1}^n \left| Y_i - r(\mathbf{X}_i \,; \Theta_m, \mathscr{D}_n) \right|^2 \Big), \end{aligned}$$

by Jensen's inequality. Denote by $\mathbb{E}_{\mathbf{X},Y}$ the expectation with respect to $\mathbf{X}$ and Y only (that is, all the other random variables are kept fixed). Thus,

simple calculations show that

$$
\begin{aligned}
&\mathbb{E}\big|s_{M,n}(\mathbf{X}) - r(\mathbf{X})\big|^2 \\
&\le \mathbb{E}\left|\mathbb{E}\big|Y - s_{M,n}(\mathbf{X})\big|^2 - \frac{1}{n}\sum_{i=1}^{n}\big|Y_i - s_{M,n}(\mathbf{X}_i)\big|^2\right| \\
&\quad + \mathbb{E}\left[\frac{1}{M}\sum_{m=1}^{M}\left|\frac{1}{n}\sum_{i=1}^{n}\big|Y_i - r(\mathbf{X}_i\,;\Theta_m,\mathscr{D}_n)\big|^2 - \mathbb{E}_{\mathbf{X},Y}\big|Y - r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n)\big|^2\right|\right] \\
&\quad + \mathbb{E}\left[\frac{1}{M}\sum_{m=1}^{M}\Big(\mathbb{E}_{\mathbf{X},Y}\big|Y - r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n)\big|^2 - \mathbb{E}\big|Y - r(\mathbf{X})\big|^2\Big)\right]. \qquad (4.4)
\end{aligned}
$$

Regarding the third term on the right-hand side and under the assumptions of Theorem 4.1, we have

$$
\begin{aligned}
&\mathbb{E}\left[\frac{1}{M}\sum_{m=1}^{M}\Big(\mathbb{E}_{\mathbf{X},Y}\big|Y - r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n)\big|^2 - \mathbb{E}\big|Y - r(\mathbf{X})\big|^2\Big)\right] \\
&= \mathbb{E}\left[\frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{\mathbf{X},Y}\big|r(\mathbf{X}) - r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n)\big|^2\right] \\
&= \mathbb{E}\big|r(\mathbf{X}) - r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n)\big|^2 \\
&\quad \to 0 \quad \text{as } n \to \infty, \qquad (4.5)
\end{aligned}
$$

since $r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n)$ is consistent by the first statement of Theorem 4.1. Regarding the second term in (4.4), according to the proof of Proposition 4.1,

$$
\begin{aligned}
&\mathbb{E}\left[\frac{1}{M}\sum_{m=1}^{M}\left|\frac{1}{n}\sum_{i=1}^{n}\big|Y_i - r(\mathbf{X}_i\,;\Theta_m,\mathscr{D}_n)\big|^2 - \mathbb{E}_{\mathbf{X},Y}\big|Y - r(\mathbf{X}\,;\Theta_m,\mathscr{D}_n)\big|^2\right|\right] \\
&\le \mathbb{E}\sup_{f\in\mathscr{F}_n}\left|\frac{1}{n}\sum_{i=1}^{n}\big|Y_i - f(\mathbf{X}_i)\big|^2 - \mathbb{E}\big|Y - f(\mathbf{X})\big|^2\right| \\
&\quad \to 0 \quad \text{as } n \to \infty, \qquad (4.6)
\end{aligned}
$$

where the set $\mathscr{F}_n$ contains all neural networks, constrained by (4.1). A similar result can be obtained for the "big" network $s_{M,n}$, just by replacing K_n (an upper bound over the number of neurons in the two layers of the "small" networks) by MK_n. Thus, since M is fixed, still under the assumptions of

Theorem 4.1, we may write

$$\begin{aligned}
&\mathbb{E}\left|\frac{1}{n}\sum_{i=1}^{n}\left|Y_i - s_{M,n}(\mathbf{X}_i)\right|^2 - \mathbb{E}\left|Y - s_{M,n}(\mathbf{X})\right|^2\right| \\
&\leq \mathbb{E}\sup_{f\in\mathscr{F}(\Theta_1,\ldots,\Theta_M,\mathscr{D}_n)}\left|\frac{1}{n}\sum_{i=1}^{n}\left|Y_i - f(\mathbf{X}_i)\right|^2 - \mathbb{E}\left|Y - f(\mathbf{X})\right|^2\right| \\
&\quad\to 0 \quad \text{as } n\to\infty. \qquad (4.7)
\end{aligned}$$

Therefore, taking expectation on both sides of (4.4), and assembling inequalities (4.5), (4.6), and (4.7), we have

$$\mathbb{E}\left|s_{M,n}(\mathbf{X}) - r(\mathbf{X})\right|^2 \to 0 \quad \text{as } n\to\infty,$$

which concludes the proof.

5 Experiments

In this section we validate the Neural Random Forest (NRF) models experimentally (Method 1 and Method 2). We compare them with standard Random Forests (RF), Neural Networks (NN) with one, two, and three hidden layers, as well as Bayesian Additive Regression Trees (BART, Chipman et al., 2010).

5.1 Training procedure

Overall, the training procedure for the NRF models goes as follows. A random forest is first learned using the *scikit-learn* implementation (Pedregosa et al., 2011) for random forests. Based on this, the set of all split directions and split positions are extracted and used to build the neural network initialization parameters. The NRF models are then trained using the *TensorFlow* framework (Abadi et al., 2015).

Network optimization. The optimization objective when learning either of the network models (NRF or standard NN) is to minimize the mean squared error (MSE) on some training set. In neural network training, this is typically achieved by employing an iterative gradient-based optimization algorithm. The algorithm iterates over the training set, generates predictions,

and then propagates the gradient of the resulting error signal with respect to all individual network parameters back through the network. The network parameters are updated such that this error is decreased, and slowly the model learns to generate the correct predictions. In practice, we found that *Adam* (Kingma and Ba, 2015), which also includes an adaptive momentum term, performed well as optimization algorithm, though any other iterative gradient-based optimization algorithm could be used instead. The results reported here were obtained with *Adam* using minibatches of size 32, default hyperparameter values ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e-08$), and initial learning rate 0.001 for which a stable optimization behavior could be observed. In preliminary experiments, minibatch size did not have a big impact on performance and was thus not tuned. At the beginning of every new epoch, the training set was shuffled and minibatches assigned anew so as to avoid overfitting to a specific order or choice of particular minibatches.

Each neural network (including the NRF networks) was trained for 100 epochs. During training, both training loss and validation loss were monitored every time an epoch was completed. The final parameters chosen are the ones that gave minimum validation error across all 100 epochs.

Neural random forests. The individual networks in Method 1 are trained just like the larger network in Method 2 for 100 epochs, and the best parameters with minimum loss on the validation set during training were picked. Computed sequentially, training Method 1 takes longer since each "small" model has to be fitted individually.

We generally found that using a lower value for γ_2 than for γ_1 (the initial contrast parameters of the activation functions in the second and first hidden layer, respectively) is helpful. This is because with a relatively small contrast γ_2, the transition in the activation function from -1 to $+1$ is smoother and a stronger gradient signal reaches the first hidden layer in backpropagation training. Concretely, for our experiments we used $\gamma_1 = 100$ and $\gamma_2 = 1$.

In some rare cases, the NRF just overfitted during optimization. This is, even though the training error went down, validation error only increased, i.e., the model's ability to generalize actually suffered from network optimization. Wherever NRF validation loss was actually worse than the RF validation loss during *all* epochs, we kept the original RF model predictions. As a consequence, we can expect the NRF predictions to be at least as good as the RF predictions, since cases where further optimization is likely to have lead to overfitting are directly filtered out.

Sparse vs. fully connected optimization. All NRF networks described in the previous sections have sparse connectivity architecture between the layers due to the nature of the translated tree structures. However, besides training sparse NRF networks, a natural modification is to not limit network training to optimizing a small set of weights, but instead to relax the sparsity constraint and train a fully connected feed-forward network. With initial connection weights of 0 between neurons where no connections were described in the sparse framework, this model still gives the same predictions as the initial sparsely connected network. However, it can adapt a larger set of weights for optimizing the objective. During the experiments, we will refer to this relaxed NRF version as *fully connected*, in contrast to the *sparse* setting, where fewer weights are subject to optimization. Both algorithms fight overfitting in their own way: the sparse network takes advantage of the forest structure during the whole optimization process, whereas the fully connected network uses the smart initialization provided by the forest structure as a warm start. Indeed, the tendency of fully connected networks to overfit on small data sets can potentially be reduced with the inductive bias derived from the trees.

We also stress that the fully connected approach differs from the sparse one only through the relaxation of the sparsity constraint during the optimization process. Apart from that, both approaches are declined with the same architectures: Method 1 (disconnected networks) or Method 2 (a "big" network). In fact, the fully connected approach should rather be understood as an algorithmic trick to initialize the network weights, as part of a traditional optimization procedure. In practice, without a fully differentiable implementation of sparse models, the fully connected models can be faster to optimize than their sparse counterpart, especially when training on a GPU. We used a dense matrix multiplication, forcing the entries of non-existing connections to 0. We acknowledge that this is not the most elegant solution and a fully differentiable implementation of sparse matrix multiplication could accelerate training here, though we did not pursue this way further.

5.2 Benchmark comparison experiments

We now compare the NRF models with standard RF, standard NN, and BART on regression data sets from UCI Machine Learning Repository (Lichman, 2013). Concretely, we use the *Auto MPG*, *Housing*, *Communities and Crime* (Redmond and Baveja, 2002), *Forest Fires* (Cortez and Morais, 2007), *Breast Cancer Wisconsin (Prognostic)*, *Concrete Compressive Strength* (Yeh,

1998), and *Protein Tertiary Structure* data sets as testing ground for the models.

To showcase the abilities of the NRF, we picked diverse, but mostly small regression data sets with few samples. These are the sets where RF often perform better than NN, since the latter typically require plentiful training data to perform reasonably well. It is on these small data sets where the benefits of neural optimization can usually not be exploited, but with the specific inductive bias of the NRF it becomes possible. We also study NRF performance on a larger data set, *Protein Tertiary Structure*, which contains 45000 observations.

Random forests are trained with 30 trees and maximum depth restriction of 6. The neural networks with one, two, or three hidden layers (NN1, NN2, NN3) trained for comparison are fully connected and have the same number of neurons in each layer as the NRF networks of Method 2 (for the third layer of NN3, the number of neurons is the same as in the second layer of NRF2). The initial parameters of standard NN were drawn randomly from a standard Gaussian distribution, and the above training procedure was followed.

For comparison we also evaluate Bayesian Additive Regression Trees (BART, Chipman et al., 2010). BART is trained also with 30 trees, 1000 MCMC steps (after burn-in of 100 steps), and otherwise default hyperparameters from the *BayesTree* R implementation.

Data preparation. We shuffled the observations in each data set randomly and split it into training, validation, and test part in a ratio of 50/25/25. Each experiment is repeated 10 times with different randomly assigned training, validation, and test set.

Data set	Number of samples	Number of features
Auto MPG	398	7
Housing	506	13
Communities and Crime	1994	101
Forest Fires	517	10
Wisconsin	194	32
Concrete	1030	8
Protein	45730	9

Table 1: Data set characteristics: number of samples and number of features, after removing observations with missing information or nonnumerical input features.

Nonnumerical features and samples with missing entries were systematically

removed. Table 1 summarizes the characteristics of the resulting data sets. In preliminary experiments, data normalisation or rescaling to the unit cube did not have a big impact on the models, so we kept the original values given in each of the sets.

Results. Table 2 summarizes the results in terms of RMSE (root mean squared error) for the different models. The first important comment is that all NRF models improve over the original RF, consistently across all data sets. So the NRF are indeed more competitive than the original random forests which they were derived from. These consistent improvements over the original RF can be observed both for sparse and fully connected NRF networks.

Data set	NN1	NN2	NN3	RF
Auto MPG	4.56 (0.83)	3.95 (0.39)	5.02 (0.72)	3.44 (0.38)
Housing	9.06 (0.85)	7.81 (0.71)	12.52 (1.08)	4.78 (0.88)
Crime	5.39 (0.42)	6.78 (0.47)	6.64 (0.31)	0.17 (0.01)
Forest Fires	96.7 (0.20)	*54.87 (34.33)*	97.9 (0.90)	95.47 (43.52)
Wisconsin	36.91 (0.88)	*34.71 (2.36)*	38.43 (2.88)	45.63 (3.53)
Concrete	10.18 (0.49)	10.21 (0.68)	11.52 (0.88)	8.39 (0.62)
Protein	6.12 (0.02)	6.11 (0.02)	6.12 (0.02)	5.06 (0.03)

NRF1 full	NRF2 full	NRF1 sparse	NRF2 sparse	BART
3.20(0.39)	3.35 (0.46)	3.28 (0.41)	3.28 (0.42)	**2.90 (0.33)**
4.34 (0.85)	4.68 (0.88)	4.59 (0.91)	4.62 (0.88)	**3.78 (0.51)**
0.16 (0.01)	0.16 (0.01)	*0.16 (0.01)*	0.16 (0.01)	**0.14 (0.01)**
54.47 (34.64)	78.60 (28.17)	68.51 (35.56)	82.80 (32.07)	55.04 (16.40)
37.12 (2.89)	41.22 (3.05)	40.70 (2.51)	38.03 (3.95)	**33.50 (2.26)**
6.28 (0.40)	6.44 (0.37)	7.42 (0.56)	7.78 (0.56)	**5.20 (0.34)**
4.82 (0.04)	*4.77 (0.05)*	4.82 (0.04)	4.77 (0.05)	**4.57 (0.04)**

Table 2: RMSE test set results (and their standard deviation) for each of the models across the different data sets. "Sparse" stands for the sparsely connected NRF model, "full" for the fully connected. Best results are displayed in bold font, second best results in italic.

For most data sets, standard NN (regardless of the number of layers) do not achieve performance that is on par with the other models, though in some cases they give competitive RMSE score. Interestingly, NRF Method 1 seems to mostly outperform Method 2 with few exceptions. In fact, BART aside, the best overall performance is achieved by the fully connected NRF with Method 1, i.e., the averaging of individually trained per-tree networks. So one typically obtains bigger benefits from averaging several independently

optimized single-tree networks than from jointly training one large network for all trees. This means that ensemble averaging (Method 1) is more advantageous for the NRF model than the co-adaptation of parameters across trees (Method 2). We conjecture that by separating the optimization process for the individual networks, some implicit regularization occurs that reduces the vulnerability to overfitting when using Method 1. Additionally, it is important to note the excellent performance of BART, which is ranked first in 6 out of the 7 experiments. Although in this context BART should be seen as a benchmark, it remains that a good idea for future research is to use BART to design neural networks, and extend the approach presented in this article.

Training evolution. To illustrate training behavior, the training and validation error for NRF (Method 2, fully connected) as well as for RF and NN from one of the experiments on the *Concrete Compressive Strength* data set are summarized in Figure 5.

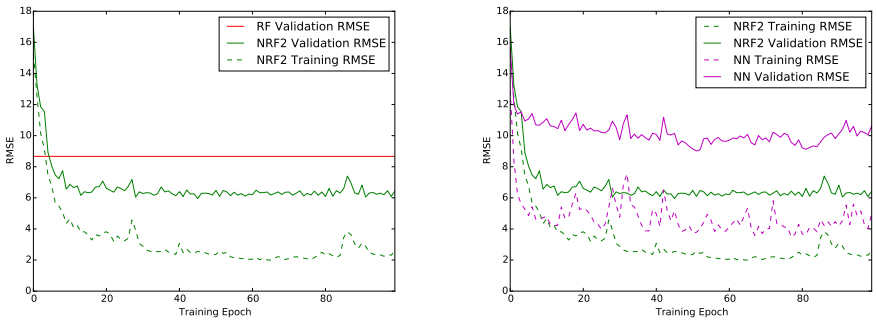

Figure 5: Evolution of training set and validation RMSE for different models. **Left**: Validation RMSE for RF (red), along with validation and training set RMSE for NRF Method 2 (fully connected, green) across training epochs. **Right**: Validation and training RMSE (solid and dotted lines, respectively) for NRF Method 2 (green) and a NN (purple) with same number of layers and neurons.

Firstly, one can observe in the left plot that validation RMSE clearly drops below the level of RF RMSE during training. Note that the difference to the constant red line (RF validation error) represents the relative improvement over the RF on the validation set. So, in fact, network optimization helps the NRF to generate better predictions than the RF and this is achieved already very early in training. The initial RMSE value at epoch 0 is not exactly

the same as for the RF, which is due to using smooth activation functions instead of sharp step functions (cf. the contrast hyperparameters γ_1 and γ_2). In the plot on the right, the same NRF model behavior is shown again, but contrasted with a standard feed-forward NN of exactly the same size and number of parameters. Clearly, the NRF reaches much lower RMSE than the NN, both on training and validation set.

5.3 Asymptotic behavior

We also investigate the asymptotic behavior of the NRF model on an artificial data set created by sampling inputs $\mathbf{x}$ uniformly from the d-dimensional hypercube $[0,1]^d$ and computing outputs y as

$$y(x) = \sum_{j=1}^{d} \sin(20x^{(j)} - 10) + \varepsilon,$$

where ε is a zero mean Gaussian noise with variance σ^2, which corrupts the deterministic signal. We choose $d = 2$ and $\sigma = 0.01$, and investigate the asymptotic behavior as the number of training samples increases. Figure 6 illustrates the RMSE for an increasing number of training samples and shows that the NRF (Method 2, fully connected) error decreases much faster than the RF error as sample size increases.

6 Some technical results

6.1 Proof of Proposition 4.1

An easy adaptation of Györfi et al. (2002, Theorem 10.2) shows that we can always assume that $\|Y\|_\infty < \infty$. Let $\mathbb{M}_{d,K_n-1}$ (resp., $\mathbb{M}_{K_n-1,K_n}$) be the vector space of $d \times (K_n - 1)$ (resp., $(K_n - 1) \times K_n$) matrices. For a generic parameter

$$\begin{aligned}\boldsymbol{\lambda} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_{\text{out}}, b_{\text{out}}) \\ \in \mathbb{M}_{d,K_n-1} \times \mathbb{R}^{K_n-1} \times \mathbb{M}_{K_n-1,K_n} \times \mathbb{R}^{K_n} \times \mathbb{R}^{K_n} \times \mathbb{R},\end{aligned}$$

we consider the constraint

$$\|\mathbf{W}_2\|_\infty + \|\mathbf{b}_2\|_\infty + \|\mathbf{W}_{\text{out}}\|_1 + |b_{\text{out}}| \leq C_1 K_n, \tag{6.1}$$

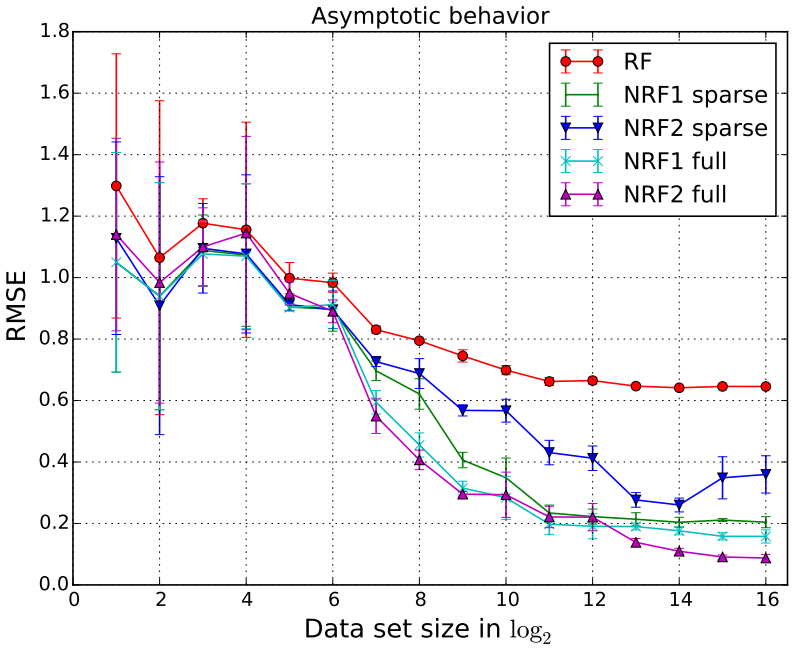

Figure 6: This figure shows the test RMSE for synthetic data with exponentially increasing training set size (x-axis). Solid lines connect the mean RMSE values obtained across 3 randomly drawn datasets for each dataset size, whereas error bars show the empirical standard deviation; 30 trees, maximum depth 6, $\gamma_1 = 100$, $\gamma_2 = 1$.

and let

$$\Lambda = \big\{\boldsymbol{\lambda} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_{\text{out}}, b_{\text{out}}) : (6.1) \text{ is satisfied}\big\}.$$

We finally define

$$\mathscr{F}_n = \{f_{\boldsymbol{\lambda}} : \boldsymbol{\lambda} \in \Lambda\},$$

where

$$f_{\boldsymbol{\lambda}}(\mathbf{x}) = \mathbf{W}_{\text{out}}^\top \sigma_2\Big(\mathbf{W}_2^\top \sigma_1(\mathbf{W}_1^\top \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2\Big) + b_{\text{out}}, \quad \mathbf{x} \in \mathbb{R}^d.$$

The set $\mathscr{F}_n$ contains all neural networks—constrained by (6.1)—with inputs in $\mathbb{R}^d$, two hidden layers of respective size $K_n - 1$ and K_n, and one output unit. We note that $\mathscr{F}(\Theta, \mathscr{D}_n) \subseteq \mathscr{F}_n$ and that $\mathscr{F}_n$ is deterministic, in the sense that it does not depend neither on Θ nor on $\mathscr{D}_n$, but only on the size n of the original data set.

Clearly,

$$\mathbb{E} \sup_{f \in \mathscr{F}(\Theta, \mathscr{D}_n)} \left| \frac{1}{n} \sum_{i=1}^{n} \left|Y_i - f(\mathbf{X}_i)\right|^2 - \mathbb{E}\left|Y - f(\mathbf{X})\right|^2 \right|$$
$$\leq \mathbb{E} \sup_{f \in \mathscr{F}_n} \left| \frac{1}{n} \sum_{i=1}^{n} \left|Y_i - f(\mathbf{X}_i)\right|^2 - \mathbb{E}\left|Y - f(\mathbf{X})\right|^2 \right|.$$

According to (6.1), each $f \in \mathscr{F}_n$ satisfies $\|f\|_\infty \leq C_1 K_n$. Thus, since Y is assumed to be bounded, using uniformly bounded classes of functions we will be able to derive a useful exponential inequality.

Let $z_1^n = (z_1, \ldots, z_n)$ be a vector of n fixed points in $\mathbb{R}^d$ and let $\mathscr{H}$ be a set of functions from $\mathbb{R}^d \to \mathbb{R}$. For every $\varepsilon > 0$, we let $\mathscr{N}_1(\varepsilon, \mathscr{H}, z_1^n)$ be the L_1 ε-covering number of $\mathscr{H}$ with respect to $z_1, \ldots, z_n$. Recall that $\mathscr{N}_1(\varepsilon, \mathscr{H}, z_1^n)$ is defined as the smallest integer N such that there exist functions $h_1, \ldots, h_N : \mathbb{R}^d \to \mathbb{R}$ with the property that for every $h \in \mathscr{H}$ there is a $j = j(h) \in \{1, \ldots, N\}$ such that

$$\frac{1}{n} \sum_{i=1}^{n} \left|h(z_i) - h_j(z_i)\right| < \varepsilon.$$

Note that if $Z_1^n = (Z_1, \ldots, Z_n)$ is a sequence of i.i.d. random variables, then $\mathscr{N}_1(\varepsilon, \mathscr{H}, Z_1^n)$ is a random variable as well.

Now, let $Z = (\mathbf{X}, Y)$, $Z_1 = (\mathbf{X}_1, Y_1), \ldots, Z_n = (\mathbf{X}_n, Y_n)$, and

$$\mathscr{H}_n = \Big\{ h(\mathbf{x}, y) := \left|y - f(\mathbf{x})\right|^2 : (\mathbf{x}, y) \in [0,1]^d \times [-\|Y\|_\infty, \|Y\|_\infty] \text{ and } f \in \mathscr{F}_n \Big\}.$$

Note that the functions in $\mathscr{H}_n$ satisfy

$$0 \leq h(\mathbf{x}, y) \leq 2C_1^2 K_n^2 + 2\|Y\|_\infty^2.$$

In particular,

$$0 \leq h(\mathbf{x}, y) \leq 4C_1^2 K_n^2$$

if n is large enough such that $C_1 K_n \geq \|Y\|_\infty$ is satisfied, which we assume.

According to an inequality of Pollard (1984) (see also Györfi et al., 2002,

Theorem 9.1), we have, for arbitrary $\varepsilon > 0$,

$$\begin{aligned}
&\mathbb{P}\Big\{ \sup_{f\in\mathscr{F}_n} \Big|\frac{1}{n}\sum_{i=1}^{n} \big|Y_i - f(\mathbf{X}_i)\big|^2 - \mathbb{E}\big|Y - f(\mathbf{X})\big|^2\Big| > \varepsilon \Big\} \\
&\quad = \mathbb{P}\Big\{ \sup_{h\in\mathscr{H}_n} \Big|\frac{1}{n}\sum_{i=1}^{n} h(Z_i) - \mathbb{E}h(Z)\Big| > \varepsilon \Big\} \\
&\quad \le 8\mathbb{E}\mathscr{N}_1\big(\frac{\varepsilon}{8}, \mathscr{H}_n, Z_1^n\big) e^{-\frac{n\varepsilon^2}{2048C_1^4K_n^4}}. \qquad (6.2)
\end{aligned}$$

So, we have to upper bound $\mathbb{E}\mathscr{N}_1\big(\frac{\varepsilon}{8}, \mathscr{H}_n, Z_1^n\big)$. To begin with, consider two functions $h(\mathbf{x}, y) = |y - f(\mathbf{x})|^2$ and $h_1(\mathbf{x}, y) = |y - f_1(\mathbf{x})|^2$ of $\mathscr{H}_n$. Clearly,

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n} \big|h(Z_i) - h_1(Z_i)\big| \\
&\quad = \frac{1}{n}\sum_{i=1}^{n} \Big|\big|Y_i - f(\mathbf{X}_i)\big|^2 - \big|Y_i - f_1(\mathbf{X}_i)\big|^2\Big| \\
&\quad = \frac{1}{n}\sum_{i=1}^{n} \big|f(\mathbf{X}_i) - f_1(\mathbf{X}_i)\big| \times \big|f(\mathbf{X}_i) + f_1(\mathbf{X}_i) - 2Y_i\big| \\
&\quad \le \frac{4C_1K_n}{n}\sum_{i=1}^{n} \big|f(\mathbf{X}_i) - f_1(\mathbf{X}_i)\big|.
\end{aligned}$$

Thus,

$$\mathscr{N}_1\Big(\frac{\varepsilon}{8}, \mathscr{H}_n, Z_1^n\Big) \le \mathscr{N}_1\Big(\frac{\varepsilon}{64C_1K_n}, \mathscr{F}_n, \mathbf{X}_1^n\Big). \qquad (6.3)$$

The covering number $\mathscr{N}_1(\frac{\varepsilon}{64C_1K_n}, \mathscr{F}_n, \mathbf{X}_1^n)$ can be upper bounded independently of $\mathbf{X}_1^n$ by extending the arguments of Lugosi and Zeger (1995) from a network with one hidden layer to a network with two hidden layers. In the arguments below, we repeatedly apply Györfi et al. (2002, Theorem 9.4, Lemma 16.4, and Lemma 16.5). The neurons of the first hidden layer output functions that belong to the class

$$\mathscr{G}_1 = \{\sigma_1(a^\top\mathbf{x} + a_0) : a \in \mathbb{R}^d, a_0 \in \mathbb{R}\},$$

and it is easy to show that, for $\varepsilon \in (0, 1/4)$,

$$\mathscr{N}_1(\varepsilon, \mathscr{G}_1, \mathbf{X}_1^n) \le 9\Big(\frac{6e}{\varepsilon}\Big)^{4d+8}.$$

Next, letting

$$\mathscr{G}_2 = \big\{bg : g \in \mathscr{G}_1, |b| \le C_1K_n\big\},$$

we get

$$\mathscr{N}_1(\varepsilon, \mathscr{G}_2, \mathbf{X}_1^n) \le \frac{4C_1K_n}{\varepsilon}\mathscr{N}_1\Big(\frac{\varepsilon}{2C_1K_n}, \mathscr{G}_1, \mathbf{X}_1^n\Big) \le \Big(\frac{36eC_1K_n}{\varepsilon}\Big)^{4d+9}.$$

The second units compute functions of the collection

$$\mathscr{G}_3 = \Big\{\sigma_2\Big(\sum_{i=1}^{K_n-1} g_i + b_0\Big) : g_i \in \mathscr{G}_2, |b_0| \le C_1K_n\Big\}.$$

Note that σ_2 satisfies the Lipschitz property $|\sigma_2(u) - \sigma_2(v)| \le \gamma_2|u-v|$ for all $(u,v) \in \mathbb{R}^2$. Thus,

$$\mathscr{N}_1(\varepsilon, \mathscr{G}_3, \mathbf{X}_1^n) \le \frac{2C_1\gamma_2K_n^2}{\varepsilon}\mathscr{N}_1\Big(\frac{\varepsilon}{2\gamma_2K_n}, \mathscr{G}_2, \mathbf{X}_1^n\Big)^{K_n-1} \le \Big(\frac{72eC_1\gamma_2K_n^2}{\varepsilon}\Big)^{(4d+9)K_n+1}.$$

Also, letting

$$\mathscr{G}_4 = \{wg : g \in \mathscr{G}_3, |w| \le C_1K_n\},$$

we see, assuming without loss of generality $C_1, \gamma_2 \ge 1$, that

$$\mathscr{N}_1(\varepsilon, \mathscr{G}_4, \mathbf{X}_1^n) \le \frac{4C_1K_n}{\varepsilon}\mathscr{N}_1\Big(\frac{\varepsilon}{2C_1K_n}, \mathscr{G}_3, \mathbf{X}_1^n\Big) \le \Big(\frac{144eC_1^2\gamma_2K_n^3}{\varepsilon}\Big)^{(4d+9)K_n+2}.$$

Finally, upon noting that

$$\mathscr{F}_n = \Big\{\sum_{i=1}^{K_n} g_i + b_{\text{out}} : g_i \in \mathscr{G}_4, |b_{\text{out}}| \le C_1K_n\Big\},$$

we conclude

$$\mathscr{N}_1(\varepsilon, \mathscr{F}_n, \mathbf{X}_1^n) \le \frac{2C_1K_n(K_n+1)}{\varepsilon}\mathscr{N}_1\Big(\frac{\varepsilon}{K_n+1}, \mathscr{G}_4, \mathbf{X}_1^n\Big)^{K_n} \le \Big(\frac{144eC_1^2\gamma_2(K_n+1)^4}{\varepsilon}\Big)^{(4d+9)K_n^2+2K_n+1}. \tag{6.4}$$

Combining inequalities (6.2)-(6.4), we obtain

$$\mathbb{P}\Big\{\sup_{f\in\mathscr{F}_n}\Big|\frac{1}{n}\sum_{i=1}^n\big|Y_i - f(\mathbf{X}_i)\big|^2 - \mathbb{E}\big|Y - f(\mathbf{X})\big|^2\Big| > \varepsilon\Big\} \le 8\Big(\frac{9216eC_1^3\gamma_2(K_n+1)^5}{\varepsilon}\Big)^{(4d+9)K_n^2+2K_n+1} e^{-\frac{n\varepsilon^2}{2048C_1^4K_n^4}}.$$

Therefore, for any $\varepsilon \in (0, 1/4)$, and all n large enough,

$$
\begin{aligned}
&\mathbb{E} \sup_{f \in \mathscr{F}_n} \Big| \frac{1}{n} \sum_{i=1}^{n} \big|Y_i - f(\mathbf{X}_i)\big|^2 - \mathbb{E}\big|Y - f(\mathbf{X})\big|^2 \Big| \\
&\leq \varepsilon + 8 \int_{\varepsilon}^{\infty} \Big(\frac{9216 e C_1^3 \gamma_2 (K_n + 1)^5}{t} \Big)^{(4d+9)K_n^2 + 2K_n + 1} e^{-\frac{nt^2}{2048 C_1^4 K_n^4}} \, \mathrm{d}t \\
&\leq \varepsilon + 8 \Big(\frac{9216 e C_1^3 \gamma_2 (K_n + 1)^5}{\varepsilon} \Big)^{(4d+9)K_n^2 + 2K_n + 1} \\
&\qquad\qquad \times \Big[- \frac{2048 C_1^4 K_n^4}{n\varepsilon} e^{-\frac{n\varepsilon t}{2048 C_1^4 K_n^4}} \Big]_{t=\varepsilon}^{\infty} \\
&\leq \varepsilon + 8 \Big(\frac{9216 e C_1^3 \gamma_2 (K_n + 1)^5}{\varepsilon} \Big)^{(4d+9)K_n^2 + 2K_n + 1} \\
&\qquad\qquad \times \Big(\frac{2048 C_1^4 K_n^4}{n\varepsilon} \Big) e^{-\frac{n\varepsilon^2}{2048 C_1^4 K_n^4}} \\
&\leq \varepsilon + 8 \Big(\frac{2048 C_1^4 K_n^4}{n\varepsilon} \Big) \\
&\qquad \times \exp \Big[\big((4d+9)K_n^2 + 2K_n + 1 \big) \log \Big(\frac{9216 e C_1^3 \gamma_2 (K_n + 1)^5}{\varepsilon} \Big) \\
&\qquad\qquad - \frac{n\varepsilon^2}{2048 C_1^4 K_n^4} \Big].
\end{aligned}
$$

Thus, under the conditions $K_n, \gamma_2 \to \infty$, and $K_n^6 \log(\gamma_2 K_n^5)/n \to 0$, for all n large enough,

$$
\mathbb{E} \sup_{f \in \mathscr{F}_n} \Big| \frac{1}{n} \sum_{i=1}^{n} \big|Y_i - f(\mathbf{X}_i)\big|^2 - \mathbb{E}\big|Y - f(\mathbf{X})\big|^2 \Big| \leq 2\varepsilon
$$

and so,

$$
\mathbb{E} \sup_{f \in \mathscr{F}(\Theta, \mathscr{D}_n)} \Big| \frac{1}{n} \sum_{i=1}^{n} \big|Y_i - f(\mathbf{X}_i)\big|^2 - \mathbb{E}\big|Y - f(\mathbf{X})\big|^2 \Big| \leq 2\varepsilon.
$$

Since ε was arbitrary, the proof is complete.

6.2 Proof of Proposition 4.2

By definition,

$$
t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) = \mathbf{W}_{\text{out}}^{\star\top} \tau\Big(\mathbf{W}_2^{\star\top} \tau(\mathbf{W}_1^{\star\top} \mathbf{x} + \mathbf{b}_1^\star) + \mathbf{b}_2^\star \Big) + b_{\text{out}}^\star
$$

and

$$f_{\boldsymbol{\lambda}^\star}(\mathbf{x}) = \mathbf{W}_{\text{out}}^{\star\top}\sigma_2\Big(\mathbf{W}_2^{\star\top}\sigma_1(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)+b_{\text{out}}^\star.$$

We have, for all $\mathbf{x}\in\mathbb{R}^d$,

$$\begin{aligned}
&\big|f_{\boldsymbol{\lambda}^\star}(\mathbf{x})-t_{\boldsymbol{\lambda}^\star}(\mathbf{x})\big|^2\\
&\quad\le \|\mathbf{W}_{\text{out}}^\star\|^2\times\Big\|\sigma_2\Big(\mathbf{W}_2^{\star\top}\sigma_1(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\\
&\qquad\qquad -\tau\Big(\mathbf{W}_2^{\star\top}\tau(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\Big\|^2\\
&\qquad\text{(by the Cauchy-Schwarz inequality)}\\
&\quad\le \frac{\|Y\|_\infty^2 K_n}{4}\times\Big\|\sigma_2\Big(\mathbf{W}_2^{\star\top}\sigma_1(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\\
&\qquad\qquad -\tau\Big(\mathbf{W}_2^{\star\top}\tau(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\Big\|^2.
\end{aligned}$$

By the triangle inequality,

$$\begin{aligned}
&\Big\|\sigma_2\Big(\mathbf{W}_2^{\star\top}\sigma_1(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)-\tau\Big(\mathbf{W}_2^{\star\top}\tau(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\Big\|^2\\
&\le 2\Big\|\sigma_2\Big(\mathbf{W}_2^{\star\top}\tau(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\\
&\qquad -\tau\Big(\mathbf{W}_2^{\star\top}\tau(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\Big\|^2\\
&\quad + 2\Big\|\sigma_2\Big(\mathbf{W}_2^{\star\top}\sigma_1(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\\
&\qquad -\sigma_2\Big(\mathbf{W}_2^{\star\top}\tau(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\Big)\Big\|^2\\
&\quad := \mathbf{I}+\mathbf{II}.
\end{aligned}$$

Upon noting that, for all $u\in\mathbb{R}$, $|\sigma_2(u)-\tau(u)|\le 2e^{-2\gamma_2|u|}$, we see that

$$\mathbf{I}\le 8\sum_{i=1}^{K_n}\exp\Big[-4\gamma_2\big|\big(\mathbf{W}_2^{\star\top}\tau(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\big)_i\big|\Big].$$

But, by the very definition of $(\mathbf{W}_1^\star,\mathbf{b}_1^\star,\mathbf{W}_2^\star,\mathbf{b}_2^\star)$—see (2.2)—we have, for every i,

$$\big|\big(\mathbf{W}_2^{\star\top}\tau(\mathbf{W}_1^{\star\top}\mathbf{x}+\mathbf{b}_1^\star)+\mathbf{b}_2^\star\big)_i\big|\ge 1/2.$$

Thus, $\mathbf{I}\le 8K_ne^{-2\gamma_2}$.

For the term $\mathbf{II}$, note that σ_2 satisfies the Lipschitz property $|\sigma_2(u)-\sigma_2(v)|\le$

$\gamma_2|u-v|$ for all $(u,v) \in \mathbb{R}^2$. Hence,

$$\begin{aligned}
\mathbf{II} &\le 2\sum_{i=1}^{K_n} \gamma_2^2 \Big| \Big(\mathbf{W}_2^{\star\top} \big(\sigma_1(\mathbf{W}_1^{\star\top}\mathbf{x} + \mathbf{b}_1^\star) - \tau(\mathbf{W}_1^{\star\top}\mathbf{x} + \mathbf{b}_1^\star) \big) \Big)_i \Big|^2 \\
&\le 2\gamma_2^2 K_n^2 \big\| \sigma_1(\mathbf{W}_1^{\star\top}\mathbf{x} + \mathbf{b}_1^\star) - \tau(\mathbf{W}_1^{\star\top}\mathbf{x} + \mathbf{b}_1^\star) \big\|^2 \\
&\quad \text{(by the Cauchy-Schwarz inequality and the definition of } \mathbf{W}_2^\star) \\
&\le 8\gamma_2^2 K_n^2 \sum_{i=1}^{K_n - 1} e^{-4\gamma_1 |(\mathbf{W}_1^{\star\top}\mathbf{x} + \mathbf{b}_1^\star)_i|}.
\end{aligned}$$

For fixed i and arbitrary $\varepsilon > 0$, we have

$$\begin{aligned}
&\int_{[0,1]^d} e^{-4\gamma_1 |(\mathbf{W}_1^{\star\top}\mathbf{x} + \mathbf{b}_1^\star)_i|} \mu(\mathrm{d}\mathbf{x}) \\
&\quad = \int_{[0,1]} e^{-4\gamma_1 |x^{(j_n^\star)} - \alpha_n^\star|} \mu(\mathrm{d}\mathbf{x}) \\
&\quad\quad \text{(for some } j \in \{1, \ldots, d\}, \text{ depending upon } \Theta \text{ and } \mathscr{D}_n) \\
&\quad \le e^{-4\gamma_1 \varepsilon} + 2\varepsilon,
\end{aligned}$$

since $\mathbf{X}$ is uniformly distributed in $[0,1]^d$. We see that, for all n large enough, choosing $\varepsilon = \frac{\log(2\gamma_1)}{4\gamma_1}$,

$$\int_{[0,1]^d} e^{-4\gamma_1 |(\mathbf{W}_1^{\star\top}\mathbf{x} + \mathbf{b}_1^\star)_i|} \mu(\mathrm{d}\mathbf{x}) \le \frac{\log(2\gamma_1)}{\gamma_1}.$$

Putting all the pieces together, we conclude that

$$\mathbb{E} \int_{[0,1]^d} \big| f_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) \big|^2 \mu(\mathrm{d}\mathbf{x}) \le 2\|Y\|_\infty^2 K_n^2 \Big(e^{-2\gamma_2} + \frac{\gamma_2^2 K_n^2 \log(2\gamma_1)}{\gamma_1} \Big).$$

The upper bound tends to zero under the conditions of the proposition.

6.3 Proof of Proposition 4.3

The proof of Proposition 4.3 rests upon the following lemma, which is a multivariate extension of Technical Lemma 1 in Scornet et al. (2015).

Lemma 6.1. *Assume that* $\mathbf{X}$ *is uniformly distributed in* $[0,1]^d$, $\|Y\|_\infty < \infty$, *and* $r \in \mathscr{F}$. *Assume, in addition, that* $L^\star \equiv 0$ *for all cuts in some given nonempty cell* A. *Then the regression function* r *is constant on* A.

Proof of Lemma 6.1. We start by proving the result in dimension $d = 1$. Letting $A = [a, b]$ $(0 \leq a < b \leq 1)$, one has

$$\begin{aligned} L^\star(1, z) &= \mathbb{V}[Y|\mathbf{X} \in A] - \mathbb{P}[a \leq \mathbf{X} \leq z \,|\, \mathbf{X} \in A]\mathbb{V}[Y \,|\, a \leq \mathbf{X} \leq z] \\ &\quad - \mathbb{P}[z \leq \mathbf{X} \leq b \,|\, \mathbf{X} \in A]\mathbb{V}[Y \,|\, z < \mathbf{X} \leq b] \\ &= -\frac{1}{(b-a)^2}\left(\int_a^b r(t)\mathrm{d}t\right)^2 + \frac{1}{(b-a)(z-a)}\left(\int_a^z r(t)\mathrm{d}t\right)^2 \\ &\quad + \frac{1}{(b-a)(b-z)}\left(\int_z^b r(t)\mathrm{d}t\right)^2. \end{aligned}$$

Let $C = \int_a^b r(t)\mathrm{d}t$ and $R(z) = \int_a^z r(t)\mathrm{d}t$. Simple calculations show that

$$L^\star(1, z) = \frac{1}{(z-a)(b-z)}\left(R(z) - C\,\frac{z-a}{b-a}\right)^2.$$

Thus, since $L^\star \equiv 0$ on $\mathscr{C}_A$ by assumption, we obtain

$$R(z) = C\,\frac{z-a}{b-a}.$$

This proves that $R(z)$ is linear in z, and therefore that r is constant on $[a, b]$.

Let us now examine the general multivariate case, where the cell is a hyperrectangle $A = \Pi_{i=1}^d [a_i, b_i] \subseteq [0, 1]^d$. We recall the notation $A^{\backslash j} = \prod_{i \neq j}[a_i, b_i]$ and $\mathrm{d}\mathbf{x}^{\backslash j} = \mathrm{d}x_1 \ldots \mathrm{d}x_{j-1}\mathrm{d}x_{j+1} \ldots \mathrm{d}x_d$. From the univariate analysis above, we know that for all $j \in \{1, \ldots, d\}$ there exists a constant C_j such that

$$\int_{A^{\backslash j}} r(\mathbf{x})\mathrm{d}\mathbf{x}^{\backslash j} = C_j.$$

Since $r \in \mathscr{F}$, it is constant on A. This proves the result. □

The proof of the next proposition follows the arguments of Proposition 2 in Scornet et al. (2015), with Lemma 6.1 above in lieu of their Technical Lemma 1. The proof is therefore omitted. We let $A_n(\mathbf{X}, \Theta)$ be the cell of the tree grown with random parameter Θ that contains $\mathbf{X}$.

Proposition 6.1. *Assume that* $\mathbf{X}$ *is uniformly distributed in* $[0, 1]^d$, $\|Y\|_\infty < \infty$, *and* $r \in \mathscr{F}$. *Then, for all* $\rho, \xi > 0$, *there exists* $N \in \mathbb{N}^\star$ *such that, for all* $n > N$,

$$\mathbb{P}\left[\sup_{\mathbf{z}, \mathbf{z}' \in A_n(\mathbf{X}, \Theta)} \left|r(\mathbf{z}) - r(\mathbf{z}')\right| \leq \xi\right] \geq 1 - \rho.$$

We are now in a position to prove Proposition 4.3. Recall that

$$t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} \in A_n(\mathbf{x}, \Theta)].$$

Accordingly,

$$\begin{aligned}
\big|t_{\boldsymbol{\lambda}^\star}(\mathbf{x}) - r(\mathbf{x})\big|^2 &= \Big|\mathbb{E}\big[Y \,|\, \mathbf{X} \in A_n(\mathbf{x}, \Theta)\big] - r(\mathbf{x})\Big|^2 \\
&= \Big|\mathbb{E}\big[r(\mathbf{X}) \,|\, \mathbf{X} \in A_n(\mathbf{x}, \Theta)\big] - r(\mathbf{x})\Big|^2 \\
&\leq \sup_{\mathbf{z},\mathbf{z}' \in A_n(\mathbf{x},\Theta)} \big|r(\mathbf{z}) - r(\mathbf{z}')\big|^2,
\end{aligned}$$

since r is continuous on $[0,1]^d$. Taking the expectation on both sides, we finally obtain

$$\mathbb{E}\big|t_{\boldsymbol{\lambda}^\star}(\mathbf{X}) - r(\mathbf{X})\big|^2 \leq \mathbb{E}\left[\sup_{\mathbf{z},\mathbf{z}' \in A_n(\mathbf{X},\Theta)} \big|r(\mathbf{z}) - r(\mathbf{z}')\big|^2\right],$$

which tends to zero as $n \to \infty$, according to Proposition 6.1 (since r is bounded on $[0,1]^d$).

Acknowledgments. We greatly thank the Associate Editor and the Referee for valuable comments and insightful suggestions, which lead to a substantial improvement of the paper.

References

M. Abadi, A. Agarwal, P. Barham, and E. Brevdo et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/.

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

G. Biau and E. Scornet. A random forest guided tour (with comments and a rejoinder by the authors). *TEST*, 25:197–227, 2016.

A.-L. Boulesteix, S. Janitza, J. Kruppa, and I.R. König. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2:493–507, 2012.

L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Chapman & Hall/CRC, Boca Raton, 1984.

R.P. Brent. Fast training algorithms for multi-layer neural nets. *IEEE Transactions on Neural Networks*, 2:346–354, 1991.

H.A. Chipman, E.I. George, and R.E. McCulloch. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4:266–298, 2010.

P. Cortez and A. Morais. A data mining approach to predict forest fires using meteorological data. In J. Neves, M.F. Santos, and J. Machado, editors, *New Trends in Artificial Intelligence, Proceedings of the 13th Portugese Conference on Artificial Intelligence*, pages 512–523, 2007.

L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, New York, 1996.

M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

P. Geurts and L. Wehenkel. Closed-form dual perturb and combine for tree-based models. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 233–240, New York, 2005. ACM.

L. Györfi, M. Kohler, A. Krzyżak, and H. Walk. *A Distribution-Free Theory of Nonparametric Regression*. Springer, New York, 2002.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning. Second Edition*. Springer, New York, 2009.

Y. Ioannou, D. Robertson, D. Zikic, P. Kontschieder, J. Shotton, M. Brown, and A. Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv:1603.01250*, 2016.

H. Ishwaran, U.B. Kogalur, X. Chen, and A.J. Minn. Random survival forests for high-dimensional data. *Statistical Analysis and Data Mining*, 4: 115–132, 2011.

M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulò. Deep neural decision forests. In *International Conference on Computer Vision*, 2015.

M. Lichman. UCI Machine Learning Repository, 2013. URL http://archive.ics.uci.edu/ml.

G. Lugosi and K. Zeger. Nonparametric estimation via empirical risk minimization. *IEEE Transactions on Information Theory*, 41:677–687, 1995.

N. Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7:983–999, 2006.

B.H. Menze, B.M. Kelm, D.N. Splitthoff, U. Koethe, and F.A. Hamprecht. On oblique random forests. In D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 453–469. Springer, Berlin, 2011.

C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, 138:221–254, 2003.

F. Pedregosa, G. Varoquaux, A. Gramfort, and V. Michel et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

D. Pollard. *Convergence of Stochastic Processes*. Springer, New York, 1984.

M. Redmond and A. Baveja. A data-driven software tool for enabling cooperative information sharing among police departments. *European Journal of Operational Research*, 141:660–678, 2002.

D.L. Richmond, D. Kainmueller, M.Y. Yang, E.W. Myers, and C. Rother. Relating cascaded random forests to deep convolutional neural networks for semantic segmentation. *arXiv:1507.07583*, 2015.

L. Rokach and O. Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific, Singapore, 2008.

E. Scornet, G. Biau, and J.-P. Vert. Consistency of random forests. *The Annals of Statistics*, 43:1716–1741, 2015.

I.K. Sethi. Entropy nets: From decision trees to neural networks. *Proceedings of the IEEE*, 78:1605–1613, 1990.

I.K. Sethi. Decision tree performance enhancement using an artificial neural network interpretation. In I.K. Sethi and A.K. Jain, editors, *Artificial Neural Networks and Statistical Pattern Recognition*, volume 6912, pages 71–88. Elsevier, Amsterdam, 1991.

J. Welbl. Casting random forests as artificial neural networks (and profiting from it). In X. Jiang, J. Hornegger, and R. Koch, editors, *Pattern Recognition*, pages 765–771. Springer, 2014.

I.-C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research*, 28:1797–1808, 1998.

O.T. Yildiz and E. Alpaydin. Regularizing soft decision trees. In *Information Sciences and Systems 2013*, pages 15–21, Cham, 2013. Springer.