Representational Gradient Boosting: Backpropagation in the Space of Functions.

Gilmer Valdes, Jerome H. Friedman, Fei Jiang, and Efstathios D. Gennatas

***Abstract*—The estimation of nested functions (i.e. functions of functions) is one of the central reasons for the success and popularity of machine learning. Today, artificial neural networks are the predominant class of algorithms in this area, known as representational learning. Here, we introduce Representational Gradient Boosting (RGB), a nonparametric algorithm that estimates functions with multi-layer architectures obtained using backpropagation in the space of functions. RGB does not need to assume a functional form in the nodes or output (e.g. linear models or rectified linear units), but rather estimates these transformations. RGB can be seen as an optimized stacking procedure where a meta algorithm learns how to combine different classes of functions (e.g. Neural Networks (NN) and Gradient Boosting (GB)), while building and optimizing them jointly in an attempt to compensate each other's weaknesses. This highlights a stark difference with current approaches to meta-learning that combine models only after they have been built independently. We showed that providing optimized stacking is one of the main advantages of RGB over current approaches. Additionally, due to the nested nature of RGB we also showed how it improves over GB in problems that have several high-order interactions. Finally, we investigation both theoretically and in practice the problem of recovering nested functions and the value of prior knowledge.**

I. Introduction

Data sets often consist of mixed and complex features. A single algorithm or type of function is unlikely to be the best at analyzing all of them. For instance, a medical data set may consist of MRI or CT (images), physician notes (text), laboratory test results (tabular data), genetic information (tabular data) and physiological recordings (time series) . To model the outcome of interest, we may choose the most appropriate algorithm for each individual type of feature group (e.g. Convolutional Neural Networks, CNN, for images and Gradient Boosting, GB, for tabular data), and combine them in a way that can best predict outcomes. This modeling strategy is known as meta learning [1,2,3], which usually takes place in two steps. In the first step, a separate model is fitted to each group of features, without considering the information in the other features. We call these base models. In the second step, the estimates from the base models are combined, usually linearly, to predict the outcome.

This two-step strategy often results in suboptimal parameter estimation. If one fits the individual models separately, when combining them in the meta learning step we ignore the relationships among different categories of predictors (i.e images and tabular data), and hence certain types of information losses are unavoidable. This is to say, the individual models are fitted without taking into consideration that they will be later combined. Alternatively, each base model can be trained on all variables but using the same algorithm (i.e NN). Performance will also suffer. This is due to the fact that only one class of function (potentially sub optimal) is selected to analyze all the different groups of variables (i.e NN for tabular data or GB for images).

To address the above issues, we propose Representational Gradient Boosting, a novel representational learning algorithm based on gradient boosting (Figure 1). Instead of combining base models' outputs at the end, RGB uses backpropagation to train them in parallel so that they may compensate each other's weaknesses and/or share information. Similar to Neural Networks, RGB estimates a function comprised of many different functions stacked in a multi-layer architecture (i.e. a set of nested functions) which allows the incorporation of prior knowledge previously designed for each individual algorithm or type of data.

Unlike NNs, RGB does not assume a functional form in the nodes or output (e.g. linear models and ReLu) but estimates these transformations. Furthermore, RGB allows the combination of differentiable (e.g. NNs) and non-differentiable functions (e.g. GB) bringing together the most popular state-of-the-art algorithms today.

RGB can also be seen as a nonparametric NN where increases in complexity do not require to change the architecture (i.e wider or deeper networks) but are achieved with each iteration. Borrowing from this connection with NN and GB, RGB can also be thought of as gradient descent in the space of functions for a multilayer architecture.

Backpropagation in the space of functions had been already suggested by pioneering work from G. Mani [4] but since the key idea of Gradient Boosting (fitting the gradient with base learners) had not been explored at the time, ad-hoc or inefficient compromises were made. Other algorithms are also closely related to RGB and highlighting them provide useful insights. For instance, RGB can also be thought as a generalization of Projection Pursuit (closely related to NN) [5,6]. In Projection

Gilmer Valdes was with the Department of Radiation Oncology and Department of Epidemiology and Biostatistics, University of California, San Francisco
E-mail: gilmer.valdes@ucsf.edu
Jerome H. Friedman was with the Department of Statistics, Stanford University
E-mail: jhf@stanford.edu
Fei Jiang was with the Department of Epidemiology and Biostatistics, University of California, San Francisco
E-mail: fei.jiang@ucsf.edu
Efstathios D. Gennatas was with the Department of Radiation Oncology, Stanford University and the Department of Epidemiology and Biostatistics, University of California, San Francisco
E-mail: gennatas@stanford.edu

Pursuit, similar to vanilla Neural Networks, linear transformations of the input space are performed, followed by nonlinear smoothers. In RGB, nonlinear general transformations and output function are estimated using nonparametric models.

In the present article we performed experiments to gain intuition on the inner workings of RGB and the classes of problems best suited for it. Specifically, here we investigate:

1) RGB formalism to fit functions.
2) Would RGB improve over combination of models after they have being built (optimized stacking)?
3) Understand the type of problems best suited for RGB.
4) How much and what type of prior knowledge can help improve performance?
5) Under which conditions can/if nested functions be recovered?

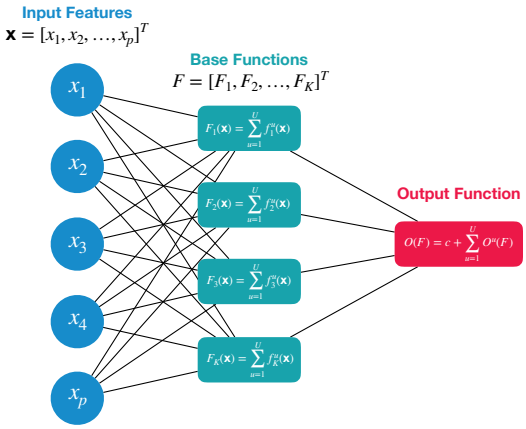


Fig. 1: Vanilla Representational Gradient Boosting (one hidden layer). p features, k base functions and u updates.

II. Related work

Machine Learning (ML) algorithms can be used to analyze both structured, a.k.a. tabular, and unstructured data, like images, or text. As a tool to analyze unstructured data, representational learning has become extremely popular in recent years[7]. This can be mainly attributed to the success that NNs have achieved in pattern recognition problems like image classification or natural language processing[8,9,10].

In general, it is thought that the tasks above are well characterized by a hierarchy where simpler, more general concepts get combined to define less abstract ones. Their multi-layer architecture allows NNs to match these type of functions well. Besides their architecture, NNs can incorporate additional prior knowledge that is also thought to help them perform well on these tasks.

In CNN, kernels are swept across images (convolution) to learn general concepts like lines or circles in the first layers which are later combined to form high order features[11]. Additionally, pooling layers are included to provide the network with a degree of invariance to feature translation (the output will be the same regardless of the specific position of a feature within the pooling layer). Therefore, both convolution and pooling are essential in explaining the performance of CNN. To further this point, Alain and Bengio[12] showed that two layers of convolution and pooling, even with random weight initialization (no training), was enough to take the test prediction error of logistic regression from roughly 8 to 2 percent on the MNIST data set.

In the analysis of tabular data other algorithms like GB excel[13,14]. There are many reasons for this. GB commonly uses trees as base learners and therefore inherits many of their advantages while mitigating their disadvantages. Trees are consistent estimators, invariant to monotonic transformations of the input variables, can handle categorical variables and missing values very efficiently, and perform intrinsic feature selection; characteristics that NN do not possess. The position of features respect to other features is not important in tabular data and prior knowledge that exploit locality (i.e convolution) does not help. Additionally, GB is very easy to regularize with three main tunable parameters (shrinkage, depth of the trees and number of trees) improving over the greedy nature of single decision trees. In fact, it has been shown that CART is the greediest version of gradient boosting with stumps[15,16].

Motivated by the success of GB in the analysis of tabular data Feng et al used the differential target backpropagation algorithm[17,18] to create a NN with GB models at its nodes[19]. Unfortunately, target back-propagation requires the calculation of an inverse function in an ill posed problem and as such can only be applied to simple architectures.

RGB provides GB with the modularity typical of an NN while providing the latter with the advantages of including any type of functions in its node, including tree-based layers. It expands the backpropagation algorithm to the space of functions with important theoretical and practical applications. It does not requiere the calculation of an inversed function as such it can be applied effectively to different architectures.

III. General RGB description

Let $\boldsymbol{x}_i \in \mathcal{X}$ be a p dimensional feature vector, and $y_i \in \boldsymbol{y}$ be its corresponding outcome. Furthermore, we assume $(\boldsymbol{x}_i, y_i)$ for $i = 1, \ldots, n$ are independent identically distributed random samples draw from an unknown distribution $\mathcal{D}$. In a two-class classification setting, $\mathcal{Y} = (\pm 1)$, and in regression, $\mathcal{Y} = \mathbb{R}$. The goal is to learn a function from $\mathcal{X}$ to $\mathcal{Y}$ that will perform well in predicting the outcome on new examples drawn from $\mathcal{D}$. Let us also assume that there are reasons to believe that the function is a multi-layer function where the initial input $\boldsymbol{x}$ is transformed by functions $F_k(\boldsymbol{x})$, $k = \{1, \ldots, K\}$ and $K \in \mathbb{N}$, which serve as input for the second layer function $O(\cdot)$ as $O\left(F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), ..., F_K(\boldsymbol{x})\right)$.

In the present article, we will limit ourselves to the investigation of the function with only one hidden layer, Figure 1. This nested function is generally not identifiable. We say that a function is identifiable if $O\left(F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), ..., F_K(\boldsymbol{x})\right) = P\left(G_1(\boldsymbol{x}), G_2(\boldsymbol{x}), ..., G_K(\boldsymbol{x})\right)$ implies that $O = P, F_1 = G_1, F_2 = G_2, ..., F_k = G_k$ for any given $\boldsymbol{x}$. Hence, we must impose certain types of identifiability conditions to perform parameter estimation and inference. We discuss several identifiablitly conditions in Appendix. However, when our goal is prediction, identifying individual parameters is not necessary.

Therefore, let $\boldsymbol{F}(\boldsymbol{x}) = [F_1(\boldsymbol{x}), \ldots, F_K(\boldsymbol{x})]^\top$. We obtain the parameter estimators through minimizing the loss function $J(O, \boldsymbol{F})$.

$$J(O, \boldsymbol{F}) = \sum_{i=1}^{N} L\left(y_i, O\left(\boldsymbol{F}(\boldsymbol{x}_i)\right)\right).$$

We estimate O and $\boldsymbol{F}$ iteratively as described below. Here we use superscript t to indicate the parameter estimator and functions at iteration t.

First, consider $\boldsymbol{F}^t$ and O^t are given and we perform a gradient descent step to update O^{t+1} using regular Gradient Boosting [13]:

$$O^{t+1} = O^t + \rho^t \Delta O^t(\boldsymbol{F}^t),$$

where $\Delta O^t(\boldsymbol{F}^t)$ is the estimation of the direction of the loss function gradient with respect to O:

$$\Delta O^t = \text{argmin}_{\Delta O} \sum_{i=1}^{N} \left(-\frac{\partial L\left(y_i, O^t\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)\right)}{\partial O^t\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)} - \Delta O\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right) \right)^2 \tag{1}$$

and ρ^t is the step size obtained as

$$\rho^t = \text{argmin}_{\rho} \sum_{i=1}^{N} L\left(y_i, O^t\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right) + \rho \Delta O^t\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)\right) \tag{2}$$

Now, once we obtain O^{t+1}, we can update $\boldsymbol{F}^t$ by propagating the gradient back and performing updates in each of the components (functions) as indicated below. This step becomes backpropagation in the space of functions:

$$F_k^{t+1} = F_k^t + \rho_F^t \Delta f_k^t \tag{3}$$

$$\begin{aligned} \Delta f_k^t \quad = \quad & \text{argmin}_{\Delta f_k} \sum_{i=1}^{N} \left(-\frac{\partial L\left(y_i, O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)\right)}{\partial O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)} \right. \\ & \left. \times \frac{\partial O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)}{\partial \boldsymbol{F}^t(\boldsymbol{x}_i)} - \Delta f_k(\boldsymbol{x}_i) \right)^2 \end{aligned} \tag{4}$$

$$\begin{aligned} \rho_F^t \quad = \quad & \text{argmin}_{\rho_F} \sum_{i=1}^{N} L\left(y_i, O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right.\right. \\ & \left.\left. + \rho_F \Delta \mathbf{f}^t(\boldsymbol{x}_i)\right)\right) \end{aligned} \tag{6}$$

where $\Delta \mathbf{f}^t(\boldsymbol{x}_i) = [\Delta f_1^t(\boldsymbol{x}_i), \ldots, \Delta f_K^t(\boldsymbol{x}_i)]^\top$.

Normally, (5) does not have a closed solution but we can approximate ρ_F^t using a single Newton-Raphson step given by

$$\begin{aligned} & - \left(\frac{\partial^2 \sum_{i=1}^{N} L\left(y_i, O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i) + \rho_F \Delta \mathbf{f}^t(\boldsymbol{x}_i)\right)\right)}{\partial \rho_F^2} \Big|_{\rho_F = 0} \right)^{-1} \\ & \times \frac{\partial \sum_{i=1}^{N} L\left(y_i, O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i) + \rho_F \Delta \mathbf{f}^t(\boldsymbol{x}_i)\right)\right)}{\partial \rho_F} \Big|_{\rho_F = 0} \end{aligned} \tag{7}$$

Algorithm 1: Representational Gradient Boosting. **(RGB)**

Input:
- $(y_i, x_i)_1^N$: data.
- t: number of iterations
- γ_o: shrinkage parameter used to regularize the output
- $\gamma_k \ \forall \ \text{k} \in (1, ..., K)$: shrinkage parameter used to regularize each channel k

Output: $O(F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), ..., F_K(\boldsymbol{x}))$

-Initialize $O^1\left(F_1^1(\boldsymbol{x}), F_2^1(\boldsymbol{x}), ..., F_K^1(\boldsymbol{x})\right)$

for *t=1:T* **do**

-Calculate $-\frac{\partial L}{\partial O^t(\boldsymbol{F^t(x_i)})}$

-Solve: $\Delta O^t =$ $\text{argmin}_{\Delta O} \sum_{i=1}^{N} \left(-\frac{\partial L\left(y_i, O^t\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)\right)}{\partial O^t\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)} - \Delta O\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right) \right)^2$

-Calculate: $\rho^t = \text{argmin}_{\rho} \sum_{i=1}^{N} L\left(y_i, O^t\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)\right) + \rho \Delta O^t\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)$

-Update the output: $O^{t+1} = O^t + \gamma_o \rho^t \Delta O^t(\boldsymbol{F}^t)$

-Calculate $-\frac{\partial L\left(y_i, O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)\right)}{\partial O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)} \frac{\partial O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)}{\partial \boldsymbol{F}^t(\boldsymbol{x}_i)}$

-$\forall$ k $\in (1, ..., K)$ Estimate:

$$\begin{aligned} \Delta f_k^t \quad = \quad & \text{argmin}_{\Delta f_k} \sum_{i=1}^{N} \left(-\frac{\partial L\left(y_i, O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)\right)}{\partial O^{t+1}\left(\boldsymbol{F}_k^t(\boldsymbol{x}_i)\right)} \right. \\ & \left. \times \frac{\partial O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i)\right)}{\partial F_k^t(\boldsymbol{x}_i)} - \Delta f_k(\boldsymbol{x}_i) \right)^2 \end{aligned}$$

-Solve: $\rho_F^t = \text{argmin}_{\rho_F} \sum_{i=1}^{N} L\left(y_i, O^{t+1}\left(\boldsymbol{F}^t(\boldsymbol{x}_i) + \rho_F \Delta \mathbf{f}^t(\boldsymbol{x}_i)\right)\right)$

- $\forall\ k \in (1, ..., K)$ update: $F_k^{t+1} = F_k^t + \gamma_k \rho_F^t \Delta f_k^t$

end

return $O^T\left(F_1^T(\boldsymbol{x}), F_2^T(\boldsymbol{x}), ..., F_K^T(\boldsymbol{x})\right)$

Finally, these updates of O and $\boldsymbol{F}$ can be repeated until the loss function reaches a certain threshold. Algorithm 1 below contains the pseudo code for RGB.

A. Examples of RGB Specific Implementations

In this section, we give examples of specific RGB implementations. In order to derive all RGB equations, we need to choose 1) the base learners, 2) the output and 3) the loss function. Many different functions can be used for the output or output updates ΔO as long as they are differentiable in order to allow calculation of $\partial O\left(\boldsymbol{F}(\boldsymbol{x}_i)\right)/\partial F_k(\boldsymbol{x}_i)$. Base learners in the first hidden layer do not have to be differentiable. We will focus on regression tasks in this paper for the derivation of the equations but by no means is RGB limited to regression and we offer a classification example without including the derivation of the equations in the paper. For regression tasks, a common loss function is the squared error loss:

$$L\left(y, O\left(\boldsymbol{F}(\boldsymbol{x})\right)\right) = \left(y - O\left(\boldsymbol{F}(\boldsymbol{x})\right)\right)^2/2,$$

The squared error loss has the advantage that the step size, ρ, for the output updates does not need to be calculated and as such it will be omitted below. Here, we investigate three different versions of RGB. First, in order to explore the RGB concept we choose linear models and trees as base learners and an output function with a convex solution. We call this implementation QuadRGB. Second, in order to compare RGB against stacking, we implement a version that has linear models in the output and a diverse set of tree-based models and linear models in the bases. We call this version sRGB. Finally, in order to explore whether using tree bases in the first layer can improve ANN performance, we explore an implementation that boosts small neural networks in the output and has tree-based learners in the hidden layer. We call this version Neural RGB (nRGB) and it is our most general implementation as both the output and the bases are fully nonparametric models.

a) QuadRGB: In this case, we choose a simple output function that includes all the bases and pairwise interactions:

$$O\left(\boldsymbol{F}(\boldsymbol{x})\right) = c_0 + \sum_{k=1}^{K} c_k F_k(\boldsymbol{x}) + \sum_{k=1}^{K-1}\sum_{j=2, j>k}^{K} c_{kj} F_k(\boldsymbol{x}) F_j(\boldsymbol{x}).$$

In the output step, for known $\boldsymbol{F}(\cdot)$, finding the coefficients is equivalent to solving a linear problem with closed solution and we do not need to apply the boosting steps. However, we saw that this strategy would make RGB converge too fast and place too much weight on the output step. To maintain more control, we therefore also boost the linear model in order to provide regularization of the output.

b) sRGB: For a fair comparison with stacking we chose a linear output model:

$$O\left(\boldsymbol{F}(\boldsymbol{x}_i)\right) = c_0 + \sum_{k=1}^{K} c_k F_k(\boldsymbol{x}_i),$$

As in the case of QuadRGB, for known $\boldsymbol{F}(\boldsymbol{x})$, finding the coefficients $c_k, k = O, \ldots, K$ is equivalent to solving a linear problem with a closed solution, although we also use boosting to approach the solution gradually.

c) nRGB: In the case that we have a summation of ReLu functions, the final output takes the form

$$O^t\left(\boldsymbol{F}(\cdot)\right) = C_0 + \sum_{u=1}^{t} \gamma_o [W_o^u ReLu\left(B_h^u + \boldsymbol{W}_h^{u^\top}\boldsymbol{F}(\cdot)\right) + B_o^u],$$

and the updates in the output function are given by

$$\Delta O^t\left(\boldsymbol{F}(\cdot)\right) = W_o^t RELU(B_h^t + \boldsymbol{W}_h^{t^\top}\boldsymbol{F}(\cdot)) + B_o^t, \quad (8)$$

where W_o^t, B_h^t, B_o^t are constants and $\boldsymbol{W}_h^t \in R^K$. The first and second order derivatives of the QuadRGB, sRGB and nRGB are given in Appendix **??**

IV. Experiments

In order to study RGB behaviour and properties, we first performed simulated studies with known data generating functions. We created a nested function defined as

$$\begin{aligned} O(x) &= F_1(x) + F_2(x) + F_3(x) + F_1(x)F_2(x) \\ &\quad + F_1(x)F_3(x) + F_2(x)F_3(x) + \epsilon \end{aligned} \quad (9)$$

where

$$\begin{aligned} F_1(x) &= 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5, \\ F_2(x) &= 0.1\exp(4x_6) + \frac{4}{1 + \exp\left(-20(x_7 - 0.5)\right)} \\ &\quad + 3x_8 + 2x_9 + x_{10}, \\ F_3(x) &= 10sin(\pi x_{11})\cos(\pi x_{12}) + 10(x_{13})^2 \\ &\quad + x_{14}x_{15}, \end{aligned} \quad (10)$$

and $\boldsymbol{x}$ is a vector of at least 15 components being all x_k drawn from the unit hyper cube and ϵ is a normally distributed variable with mean equal 0 and variance as specified on each experiment. These bases functions are similar to those used in the widely tested Friedman data set[20] while the output is a simple quadratic function of them .

A. Practical Aspects of RGB implementation

We use this section to discuss important aspects of the implementation of RGB: 1) initialization, 2) regularization via shrinkage and 3) step size during gradient descent ρ.

1) Initialization: Unlike GB, RGB can not be initialized with constants in all the functions (bases and output) because the derivatives would be zero and learning would not be possible. Careful initialization is required to account for the nested function architecture of the model. Different strategies were tested. Normally, we initialized the bases by fitting the outcome directly using random subsamples of the distributions as if there was no output function. Our idea was that initial features that predicted the outcome were good as they would correlate with some aspects of the function being fitted. We also observed that initializing the bases with a constant first plus a base fitted to the residual but with a shrinkage applied

to it helped with regularization. This was a result of having subsequent steps anchored to the dimension of the constant (chosen to be proportional to the mean value of the output for the observations used to initialize each base). Once the bases were initialized, we initialized the outcome using the same method described for Gradient Boosting [13].

2) Shrinkage γ: Using learning rate a.k.a. shrinkage is a standard technique to regularize GB. In RGB, both the output and base updates can be multiplied by a learning rate to regularize the solution:

$$O^{t+1} = O^t + \gamma_o \rho^t \Delta O^t(\boldsymbol{F}^t)$$
$$F_k^{t+1} = F_k^t + \gamma_k \rho_F^t \Delta f_k^t$$

However, there is a caveat to applying a learning rate to both the bases and the output. After the output is updated and regularized with γ_o the F_k functions will be updated. As such, the scale of Δf_k can change to cancel any shrinkage applied to the outcome. Equally, most ΔO are of the type $\Delta O\,(\mathbf{WF})$ where $\mathbf{W}$ is a linear transformation of the input $\mathbf{F}$, therefore $\mathbf{W}$ can re-scale back $\mathbf{F}$ partially canceling the shrinkage parameter, γ_k applied to the base updates. This problem can be solved by having an adaptive shrinkage parameter in the outcome that normalizes $\mathbf{W}$, for instance $\gamma_{0k} = \gamma_o / max(|\mathbf{W}|)$. This was the strategy adopted here in our cases when we used ReLUs and linear models in the output.

3) Step size ρ: Here we have included the ρ parameter to scale the step size of the updates of the functions for both the output and base learners. In ANNs this step is usually not performed but rather an adaptive learning rate is used, which decreases if the loss function increases. However, we saw that such a strategy always led to longer running times because sometimes bigger ρ could be used. As such we decided to estimate the ρ at every step as described above and adaptively decrease it by a factor of 10 if that ρ increased the loss function. Therefore, all regularization happened through the shrinkage parameter,γ in our case as every step was close to the biggest.

B. Set of Experiments 1

In this section, we analyze the value of RGB as an optimized stacking procedure. As mentioned in the introduction, when different methods are combined using stacking or blending, they are learned independently without considering that they will be subsequently combined. As such, they can not optimally compensate each others' weaknesses and focus on the aspect of the data that they are best at. For instance, one can imagine that if our problem contains imaging and tabular data, we would like to use RGB to combine CNN and GB so that each algorithm can handle the part of the data that they are best at.

We designed two experiments to address our hypothesis. In both of them we compared the linear output RGB (sRGB) against stacking the bases after they have been fully trained. Tree-based models are in general bad at modeling linear functions while the latter can't model interactions. Therefore, we used a sRGB version with two channels, one using trees as base learners and the other one using linear models. We compared this sRGB with stacking the base models (Gradient Boosting and a Linear model) after fully learned independently. The trees in the GB channel were trained with a maximum depth of 2 and a learning rate $Lr = 0.1$. The linear model was trained with a $Lr = 0.1$ as well. 1000 iterations were done in all cases. Results are shown in Table I for two different simulations using (10), one with 15 variables and no noise in the output and the other one with 30 variables and 10% noise in the output. The metric used for comparison was the squared root of the relative mean squared error (RRMSE). In both cases, sRGB improved over stacking as previously thought. Since the function space that both algorithms fit are equal (sRGB and stacking the bases), this improvement comes from optimizing them jointly as previously suggested. In section 3 we will apply to the analysis of imaging and tabular data by combining both CNNs and GB in different channels.

Stacking	sRGB	Stacking wn	sRGB wn
0.88	0.90	0.87	0.88
0.90	0.92	0.82	0.88
0.88	0.90	0.85	0.89
0.88	0.91	0.86	0.87
0.87	0.89	0.82	0.88
-0.02±0.01		-0.04±0.03	
p = 0.0002		p = 0.016	

TABLE I: Comparison of stacking vs. sRGB without noise (first two columns) and with 10% noise (last two columns). Results correspond to 5 different iterations, each row corresponding to the same test and training set. R2 is shown. In the last columns we show the mean value ± the standard deviation of each algorithm minus the results of sRGB together with the p value of one sided T test against the null hypothesis that the distributions are the same. In both cases, sRGB significantly improved over stacking.

C. Set of Experiments 2

In this section, we compare RGB against GB to illustrate a set of scenarios where the former can outperform the later. We also wanted to evaluate the value of providing RGB with prior knowledge thought to improve performance. We first generated $N = 1000$ points for training, 250 for validation and 1000 testing using (9) and (10), with 15 extra noise covariates and 10% nosie added to the output. We studied Gradient Boosting (as baseline), RGB with 3 groups of 5 bases each (12 bases total) using tree models of depth $[2, 3, 4, 5]$ and using boosted neural networks in the output (nRGB). The learning rate selected for the bases was $Lr = 0.1$. During the fitting steps of the output for the nRGB, the small networks were asked to fit 90% of the residual and stop to provide extra regularization and faster computation time (compared to fitting up to smaller percent). 2000 maximum iterations were used in all cases but early stopping was performed if the validation error increased 5 consecutive times. We tested two versions of nRGB, one where we would provide the

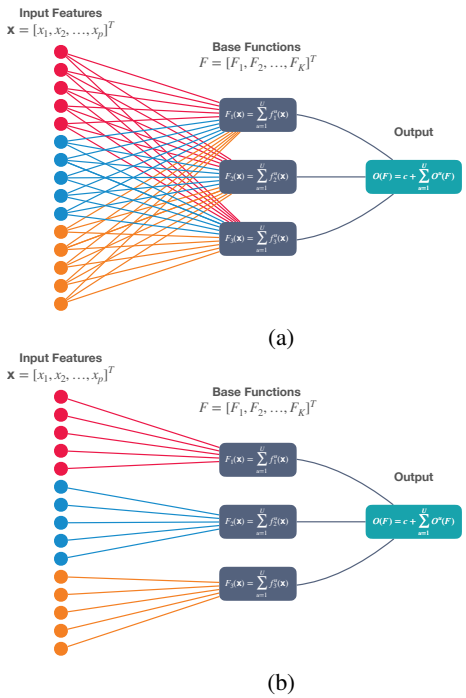


Fig. 2: (a)RGB without feature indexing. All bases receive all input variables. (b)RGB with feature indexing. Input variables are grouped to provide prior knowledge. In both cases only one base learner function per group is represented (instead of 5) for clarity of the graph.

correct variables to each group of bases (bases in group 1 only received covariates $[x_1, \ldots, x_{10}]$, bases in group 2 only received $[x_{11}, \ldots, x_{20}]$ and bases in group 3 only received $[x_{21}, \ldots, x_{30}]$), being 5 covaraites the ones used in the data generating process and the 5 others noise. This version was called "nRGB w idx". We also tested "nRGB wo idx" where all bases received the 30 covariates, Figure 2. Please note that the bases can do feature selection as they are comprised of tree based models. We called the last version "nRGB wo idx". For GB, we performed hyper parameter search for the number of iterations (max = 3000), max depth of the tree ([2,3,4,5]) and learning rate ([0.5,0.25, 0.1, 0.01]), Table II shows the RRMSE for the three algorithms.

Several conclusions can be derived from Tables II. Both versions of nRGB substantially outperformed GB. Although GB can detect high order interaction terms, RGB can perform this task substantially more efficiently. Please note that each term $F_j(x)F_k(x)$ adds between 12 and 20 new high order interactions terms to the function and as such there are 47 new interaction terms introduced in (10) beyond those included in each of the channels. All these interactions terms would have to be modeled by GB to obtain a good result but can be approximated by RGB easier and more efficiently by updating the bases on each channel. In order for GB to model high order interaction, due to the greedy nature of the trees, strong main effects need to exist (in the form of stumps) before higher order terms are discovered. This is not the case with RGB since even stump updates in the bases can result in high order interaction terms.

GB	nRGB wo idx	nRGB w idx
0.891	0.922	0.916
0.898	0.922	0.927
0.884	0.927	0.910
0.898	0.922	0.916
0.904	0.932	0.927
	-0.030±0.008	-0.024±0.004
	p = 0.003	p = 0.00009

TABLE II: Comparison of GB and nRGB without (wo) or with(w) grouping (indexing) the right variables for the different channels. Numbers correspond to R2. Each row corresponds to one experiment replication. In each experiment, new data was generated. In the last columns we show the mean value ± the standard deviation of each algorithm minus the results GB together with the p value of one sided T test against the null hypothesis that the distributions are the same.. To our surprise, nRGB without indexing achieved the top performance although only slightly better (and not statistically signifcant). Both GB and nRGB without indexing received all 30 variables in all channels with nRGB with indexing receiving 10 variables per channel where only 5 of them contribute to the outcome in the true data generating process.

GB	nRGB wo indexing	nRGB with indexing
0.922	0.840	0.870
0.910	0.856	0.840
0.927	0.863	0.798
0.924	0.840	0.824
0.922	0.863	0.760
	0.069±0.014	0.103±0.045
	p = 0.0001	p = 0.0033

TABLE III: Modeling $F_1(x) + F_2(x) + F_3(x)$ with the same settings as those used in Table II. R2 is shown. In the last columns we show the mean value ± the standard deviation of each algorithm minus the results GB together with the p value of one sided T test against the null hypothesis that the distributions are the same. If the interaction terms between the channels are removed, GB significantly outperforms RGB and the results in Table II are reversed.

To further this point, we also provide comparison when RGB and GB are used to model $F_1 + F_2 + F_3$. Here we used the same hyperparameters described above. In these type of functions where the number and order of interaction terms is substantially reduced, GB outperforms RGB, Table III. Similar results are obtained if noise is added. Therefore, those problems that are more natural to RGB are those with many interaction terms as is customary of representational problems. Of course, once could choose a set of hyper parameters that get RGB close to GB (i.e using a linear output) but in that case the advatanges of RGB might be lost unless different models are being combined (i.e optimized stacking). As such, for problems with only few interactions term GB should be preferred. As usual, there is not way to recognize if a problem has many interaction terms before hand and the practitioners should always check both. A good hint for high order interaction terms might be if the hyper parameter depth of the trees in GB is high.

Further conclusions can be derived from results in Tables II. To our surprise, providing indexing to the RGB did not

improve performance and it actually slightly worsen it. Providing indexing reduces the variance of the algorithm and in principle would not increase bias as the right function can still be recovered and should therefore improve performance. However, given that the output function is miss-specified at the beginning and we are trying to estimate it, the bases do not necessarily start approaching the estimation of the right function for each channel. This, in turns, makes the output function compensate the miss-specification of the bases converging towards a solution that approximates the nested function but not its parts. This is a result of the identifiability theorem discussed in the Appendix. Therefore, the benefit of variance reduction introduced by indexing gets negated as it becomes harder for the function to converge to a good solution. To prove that the slightly worse result when indexing is provided in our experiments are a consequence of the misspesification of the outcome experiments about identifiability are provided in the Appendix.

Finally, to make sure that our observations were not a result of running both versions of nRGB with the same number of steps and as such putting the simpler model at a disadvantage, we ran an experiment selecting the number of steps using a validation set with qualitatively equal results as those described here. As such, it seems that the reduction of variance competes with the ability to obtain similar functions that can approximate well the underlying nested function. Our results seem to indicate that not all prior knowledge can help performance and that intuition can be especially troublesome in nested functions. It also indicates that the ability to model many high order interaction terms with small updates in the bases is one of the key features to explain performance of representational learning functions like RGB and NN.

D. Set of Experiments 3

In this section, we wanted to apply RGB to real data. As illustrated in the simulation sections, RGB has 4 main advantages:

1) By learning different models that capture different aspect of the data together, it can improve over the different models or their combination after fully learned (optimized stacking).
2) It can model more efficiently functions that contain many high-order interaction terms.
3) Different from NN, RGB does intrinsic feature selection.
4) Modularity allows to reduce variance with priors although this can also potentially harm performance.

In that regard, we thought that: 1) problems with both tabular and imaging data as input potentiated the optimized stacking contribution as GB and CNN could be used together, 2) multivariate time series data and sequence data like those found in Natural Language Processing (NLP) tasks also offered the type of problems on which RGB can perform the best given that: 1) features are usually noisy and feature selection is needed 2) many high-order interaction terms are likely to exist between the variables 3) due to the nature of the data, regularization and prior knowledge might help (e.g data becomes less important as time to prediction increases) 4) different variables or channels might benefit from the use of different classes of functions.

As such, we decided to compare the performance of RGB against ML algorithms in the analysis of 1) one multi modal problem (regression), 2) a timeseries problem (regression) and one NLP problem (classification). In the three cases, we compared RGB to the state of the art approaches for all the cases. One problem was a privately collected data set (medical problem) that can not be relased due to IRB and privacy issues. The other two datasets are publicly available and can be donwloaded to replicate our results. The description of these three problems can be found below:

1) Predicting House Prices: This dataset consists of 2840 color images (227×227) with their corresponding tabular data (Latitude, Longitude, Number of Beds, Number of Baths and Zip Code). The output is the price of house in USD. The dataset was collected by Rosenfelder and analyzed in his personal blog. The same can be downloaded here Donwload Data. The original images where (224×224) so we added 3 pixels to the left of the images to make them the current size (so that it could be read by our CNN). Additionally, all the tabular data was standardized as well as the output (prices of the house) and the images rescale to be in the range [0,1]. We split the data in two sets: 80% for training and validation and 20% for testing. All preprocessing was done using the values from the training set (e.g mean). Here we decided to combine a CNN, a linar model and GB using a linear output layer as our model. For the CNN architecture we used the first 16 layers of Alexnet (removing the regression layers)[8]. The parameters of this portion of the CNN were initialized using those of Alexnet after being trained on Imagenet[8]. We also used a learning rate equal to $Lr = 10^{-5}$ during training for the parameters of this portion of the network. After the first 16 Alexnet's layers, we added a dropout layer (probability = 0.5) and a one node fully connected layer. These parameters were initialized with the glorot initializer[21] and a learning rate equal to $Lr = 10^{-4}$ was used for them. These learning rates were selected as half of those that did not cause the CNN to diverge in early explorations as suggested by Larochelle et all[22]. All these layers will be called CNN here. They received as input the images. Besides the CNN channel, we set up a linear channel(2) and a GB channel(3). Both of these channels received as input the tabular vector. The output of the CNN was then concatenated to the output of channel 2 and 3. This concatenated vector was passed to the final regression layer whose bias was set to 0 (the output had been normalized) and the rest of the parameters to $\frac{1}{3}$. We used a learning rate equal to $Lr = 10^{-2}$ for the linear channel and the GB channel and they both were initialized with a constant equal to 0 (mean of the output after normalization). The maximum depth of the trees considered for the GB channel was 5. This network is represented in Figure 3.

For training, the parameters of the whole network (the CNN + the output layer) were updated using backpropagation and the automatic differentiation toolbox from Matlab Inc. The model was trained for 60 epochs with a batch size of 512. Both channel 2 (the linear model) and channel 3 (GB) where updated using backpropagation in the space of functions. We

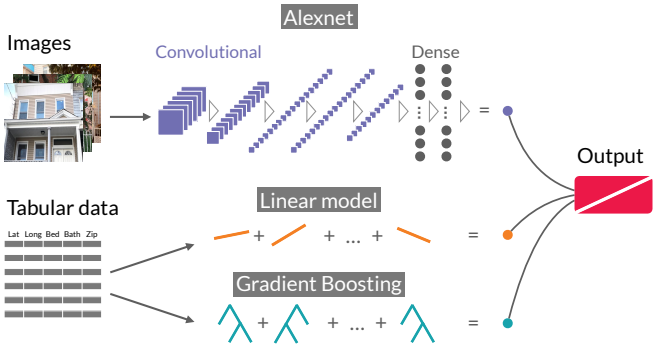


Fig. 3: Architecture of RGB combining Alexnet, a linear channel and a Gradient Boosting channel. The output layer is a linear combination of the concatenated vector of the three different algorithms.

fitted the residual divided by the coefficient from the output layer for each channel to not only get the direction of the step in the space of functions but also its magnitude at the same time as suggested by Friedman[13]:

$$\Delta F_k = \text{argmin}_{\Delta F_k} \sum_{i=1}^{N} \left(\frac{y_i - RGB(\boldsymbol{x}_i)}{c_k} - \Delta \boldsymbol{F_k}(\boldsymbol{x}_i) \right)^2 \tag{11}$$

These updates were added to each of these channels at each iteration after they have been multiplied by the learning rate. Besides training RGB, we also trained a multi input CNN (called here mCNN). The CNN channel portion of mCNN was identical to that of RGB but then the output of the CNN was concatenated with the raw tabular data (also standardized) and passed to the output layer. Additionally, we trained a GB model using only the tabular data, 10000 iterations (with early stopping), $Lr = 0.01$ and $depth = 5$. Finally we also created a stacked model (linear combination) using the output of the mCNN and GB. The results of these experiments are reported in Table IV. As it can be seen, RGB significantly outperforms any other strategy including stacking mCNN and GB. Since both classes of functions are identical, this improvement comes from optimizing them jointly as previously shown for the combination of linear models and GB in the set of experiments 1.

2) Brain Signal Dataset: This data set contains brain signals from 90 different sensors placed on patients head. The data set had 90 region of interests (ROI), each with 180 observations taken serially with time. Although no specific ROI is more important than the other, as proof of principle, we decided to predict the first channel only, using information from all the other ROIs and early data from that same ROI. We used the first 80% for training and the last 20% for testing. We also decided to predict the next value only.

For this problem, no transformation was applied to the data when GB and RGB were analyzed since these algorithms are invariant to monotonic transformations of the input variables. This together with how easy trees can handle missing values is one of the advantages or RGB although no specifically investigated here. When LSTMs were used, the data were

GB	mCNN	stacked GB + mCNN	RGB
0.523	0.551	0.590	**0.698**
0.473	0.523	0.601	**0.667**
0.424	0.615	0.668	**0.703**
0.491	0.591	0.647	**0.720**
0.447	0.505	0.640	**0.680**
-0.222 $\pm$0.041	-0.137 $\pm$0.032	-0.0642 $\pm$0.029	
p = 0.0001	p = 0.0003	p = 0.0039	

TABLE IV: R2 for each algorithm evaluated on the test set. GB was only trained on the tabular data while mCNN was trained using both the images and the tabular data. RGB was trained using both the images and tabular data with three different channels (CNN, Linear Model and GB). In the last columns we show the mean value $\pm$ the standard deviation of each algorithm minus the results RGB together with the p value of one sided T test against the null hypothesis that the distributions are the same. RGB significantly outperforms every strategy including stacking.

normalized first. For RGB and GB we decided to use the data with lag 5. We also used lag 10 but found no advantage for the extra computational time. With GB, the learning rate $[0.01, 0.01, 0.25, 0.5, 1]$ and the depth of the trees $[1, 2, 3, 4, 5]$ where cross validated using the training data and max number of iteration equal to 2000 combined with early stopping. For nRGB we used 3 versions. In all cases, a channel using trees of depth equal 2 and another channel building linear models were used. In the first version of nRGB we did not provide prior information and as such, 450(90x5) variables where provided to each channel. The learning rate was set to $Lr = 0.1$. We will call this version "nRGB wo idx". In the second version we divided the data into 5 groups of channels, two base models per channel(one using trees of depth equal 2 and the other one using linear models). Each group received indexing to only include variables corresponding to a time stamp. All the channels had the same Learning rate of $Lr = 0.1$ as well. We called this version "nRGB w idx". The prior knowledge provided here assumes that variables influencing on the outcome depends on time. The last version of nRGB was equal to the second but additionally we decreased the Lr by a factor of two with time. We are assuming that data separated longer in time is more noisy and needs more regularization, Figure 4.

Finally, we also compared it with two LSTM models, one using one hidden LSTM layer and a fully connected layer, called here "LSTM", and the other one using 5 LSTM layers with 0.2 dropout layers between them and a fully connected layer, called here "Deep LSTM". Different number of hidden units were tested in the LSTM layers that ranged from 10 to 500. Additionally, we explored different initial learning rates $[0.0001, 0.001, 0.001, 0.01, 0.1]$, with a learning rate drop period of $[10, 25, 50]$ and different learning rate drop factors [0.1, 0.2, 0.5, 0.8] and epochs $[50, 100, 200, 500, 1000]$. The networks were trained using the Adam optimizer. Results shown for the LSTM are for the best combination of hyper parameters.

Results for these experiments are shown in Table V. The reduction of variance provided by both priors (indexing and

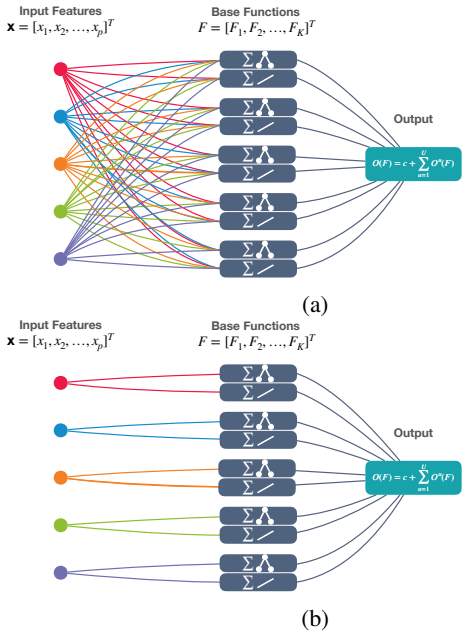


Fig. 4: (a) Architecture of RGB when there was no indexing according to the lag or time stamp (b) when variables are grouped according to their lag

GB	sRGB wo idx	sRGB w idx
0.328	0.226	0.510
sRGB w id and LR	**LSTM**	**Deep LSTM**
0.578	0.208	0.154

TABLE V: RRMSE for each algorithm. GB was better than plain sRGB and both of them better than LSTM. However, once we provided prior information to RGB in the form of variable indexing and stronger regularization with time, sRGB w id and LR resulted to be the best algorithm.

differential learning rate with time) independently improved performance, eventually making nRGB more accurate than GB. Additionally, it seems that doing feature selection is very important in this problem as both GB and nRGB significantly outperformed the LSTM models, even when substantially more time was spent looking for the right combination of hyper parameters in the latter. No time was spent searching for hyper parameters in RGB as sensible choices like those used here give good performance.

3) Classiying Weather Reports: This is a Natural Language processing task and it consisted of classifying text descriptions of weather reports. In total there were 40 categories (e.g Freezing Fog, Hurricane). This data was used as demonstration of LSTM networks by Matlab Inc and they had reported accuracy of these models to be, $Accuracy = 0.875$. In total, there were 23900 reports. The data was approximately divided into 80% for training and 20% for testing, having the same class distribution in both groups.

For this problem, a classification task, we used the LSTM network and followed the training procedure provided by Matlab Inc. This network consisted of a word embedding layer (dimension = 100) + LSTM layer (hidden units = 80) + fully connected layer + a Soft max layer. Each weather report was tokenized and 75 tokens were selected. The Adam optimizer and the same learning rate policy described above were used. We separated the data into 80% for training and 20% for testing. Using these hyperparameters we replicated the results reported in Matlab with an $Accuracy = 0.87$.

We extracted the representation learned in the word embedding layer and provided it as input to RGB. In this version of RGB we had 39 groups of channels, each using three channels with trees of depth = $[1, 3, 5]$. Each channel specialized in predicting an $F_k(x)$ that were later linearly combined into a softmax layer to produce a probability over the classes. Using the training data, we also explored combinations of linear models and trees in the groups but observed that there was always overfitting using the linear models. Therefore we concluded that feature selection was needed and used the RGB described above. The learning rate for each channel was $Lr = 0.1$. When the word representation learned by the LSTM was fed as input to RGB, we obtained an $Accuracy = 0.80$.

Our results indicate that the word embedding representation was learned to facilitate the learning of the LSTM and its features are hard to reuse. In fact, when we retrained a LSTM network using this same word embedding representation, the best accuracy that we could obtain was, $Accuracy = 0.82$. This phenomenon termed co-adaptation has been extensively reported in ANNs and it indicates that "...feature detectors are only helpful in the context of several other specific feature detectors..." as described by Hinton et al.[23].

If instead of using the representation learned by the word embedding, we trained RGB using the frequencies of the words on each document we then obtained an $Accuracy = 0.88$ which rivals that of the LSTM model. Our results indicate that for RGB to be better in NLP tasks, we would need to design an initial layer with word embedding specific for our algorithm. That work, however, is beyond the scope of this paper. Additionally, note that by no means is the accuracy obtained by the LSTM models in this task, either by Matlab Inc or us, can be considered state of the art. Transformer architectures, especially when pretrained in a large corpus of data, have dominated NLP tasks in recent years[24]. As such, we should regard this experiment as an illustration of how providing feature selection (through RGB) can improve on simpler NN approaches that do not contain it.

V. Conclusions and Future Work

Here, we introduce Representational Gradient Boosting, a novel representation learning algorithm based on gradient boosting. RGB offers unique advantages from a practical perspective. It can perform optimized stacking where different functions (e.g. a CNN and GB) are learned together to improve on each others' weaknesses, even when some of the functions are not differentiable (e.g. GB). This allows to bring together differentiables (e.g. NN) and non-differentiable functions (e.g. GB) into the same optimization umbrella. Given the multi-modal nature of many data sets today where images, texts and tabular data with categorical and continue variables are combined, no specific class of functions is likely to be the best to analyze all parts of it and an algorithm that

allows different classes to be optimized together is essential. Additionally, adding layers of tree-based models makes those layers invariant to monotonic transformations of the input and able to handle missing values. Moreover, it offers a natural way to introduce feature selection for multi-layer architectures. It is able to model functions with many high-order interaction terms (typical of nested funtions) better than GB although there is not much advatange when the underlying data generation process does not consist of many interation terms. Because of these characteristics, we expected RGB to be highly competitive in the analysis of multi modal problems, timeseries and sequences. This was demonstrated here using three real data sets and extensive experimentation on synthetic data.

We also proved that not only nested functions can not be recovered in theory but actually in practice it is a very hard task requiring extensive amount of prior knowledge. These results highlight the issue that although some representations learned in a certain manifold can help improve performance compared to the original variables, we should be cautious when we interpret their meaning. We also showed how, due to the identifiability issue, certain prior knowledge that was thought to help actually hurt performance. Our experiments highlight the difficulties of applying intuition to nested functions in order to improve performance. This is equally the case when we use this intuition to provide explanations of why a representation algorithm is obtaining better performance. As discussed above, it is RGB's efficiency in modeling multiple high-order interaction terms what resulted in better performance over GB in the problems analyzed. Without proper experimentation, we would have attributed the improvement to the prior knowledge provided. We encourage the interested reader to review our theoretical section and experiments regarding identifiability and representational learning in the Appendix.

Finally, we would like to discuss some aspects of the computational requirements of RGB. This depends on the particular models being selected for the individual channels as well as that for the output function. Generally simple base learners, as in regular gradient boosting, are preferred. For instance, if we select NN (e.g CNN) as the base learners, then complexity and memory requirements will explode if we optimize that channel as a summation of all of them (unless very small networks are selected). However, when one of the channels is a NN, for instance a CNN, there is no need to boost that channel (build a model with summation of smaller base learners) and the error could be propagated to that channel in the space of parameters (regular backpropagation) while we propagate the error to the other channels in the space of functions. This is the approach we have taken in Section 3 (Predicting House Prices). The computational complexity in this case is no larger than training the different models and performing stacking afterwards as it is customary done.

For future work, we would like to investigate other differentiable simple base learners so that deeper architectures can be designed although depth is not a requirement to increase complexity and it might not be needed as much as in current NN implementations. Additionally, the use of numerical derivatives can also be considered if layers of trees are desired in other positions other than the deepest layer.

REFERENCES

[1] Leo Breiman, "Stacked regressions," *Machine learning*, vol. 24, no. 1, pp. 49–64, 1996.

[2] Michael LeBlanc and Robert Tibshirani, "Combining estimates in regression and classification," *Journal of the American Statistical Association*, vol. 91, no. 436, pp. 1641–1650, 1996.

[3] David H Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[4] Ganesh Mani, "Learning by gradient descent in function space," in *Systems, Man and Cybernetics, 1990. Conference Proceedings., IEEE International Conference on*. IEEE, 1990, pp. 242–247.

[5] Jerome H Friedman and John W Tukey, "A projection pursuit algorithm for exploratory data analysis," *IEEE Transactions on computers*, vol. 100, no. 9, pp. 881–890, 1974.

[6] Jerome H Friedman and Werner Stuetzle, "Projection pursuit regression," *Journal of the American statistical Association*, vol. 76, no. 376, pp. 817–823, 1981.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[9] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] Frank Seide, Gang Li, and Dong Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[11] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[12] Guillaume Alain and Yoshua Bengio, "Understanding intermediate layers using linear classifier probes," *arXiv preprint arXiv:1610.01644*, 2016.

[13] Jerome H Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[14] Rich Caruana and Alexandru Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 161–168.

[15] José Marcio Luna, Efstathios D Gennatas, Lyle H Ungar, Eric Eaton, Eric S Diffenderfer, Shane T Jensen, Charles B Simone, Jerome H Friedman, Timothy D Solberg, and Gilmer Valdes, "Building more accurate decision trees with the additive tree," *Proceedings of*

the National Academy of Sciences, vol. 116, no. 40, pp. 19887–19893, 2019.

[16] Gilmer Valdes, José Marcio Luna, Eric Eaton, Charles B Simone II, Lyle H Ungar, and Timothy D Solberg, "Mediboost: a patient stratification tool for interpretable decision making in the era of precision medicine," *Scientific reports*, vol. 6, pp. 37854, 2016.

[17] Yoshua Bengio, "Deriving differential target propagation from iterating approximate inverses," *arXiv preprint arXiv:2007.15139*, 2020.

[18] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio, "Difference target propagation," in *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 2015, pp. 498–515.

[19] Ji Feng, Yang Yu, and Zhi-Hua Zhou, "Multi-layered gradient boosting decision trees," in *Advances in neural information processing systems*, 2018.

[20] Jerome H Friedman, "Multivariate adaptive regression splines," *The annals of statistics*, pp. 1–67, 1991.

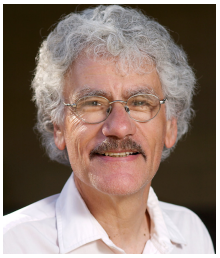
[21] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[22] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin, "Exploring strategies for training deep neural networks.," *Journal of machine learning research*, vol. 10, no. 1, 2009.

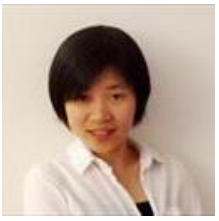
[23] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

Gilmer Valdes received the Ph.D. degree in Medical Physics from University of California, Los Angeles in 2013. He was a postdoctoral fellow at University of California, San Francisco between 2013-2014 and a medical physics resident from 2014-2016 at University of Pennsylvania. He is currently an Assistant Professor with dual appointments in the Department of Radiation Oncology and the Department of Epidemiology and Biostatistics, the University of California, San Francisco.



Jerome H. Friedman received the Ph.D. degree in High Energy Particle Physics from University of California, Berkeley in 1967. In 1972 he was appointed lead of the Computation Research Group, Stanford Linear Accelerator Center, Stanford University.He has been a Professor of the Department of Statistics at Stanford University since 1982.



Fei Jiang received the Ph.D. degree in statistics from Rice University in 2013. She was a postdoctoral fellow in Harvard between 2013–2016. From 2016 to 2019, She was an assistant professor and associate director for the artificial intelligence and data science programs in the University of Hong Kong. She joined the Department of Epidemiology and Biostatistics, the University of California, San Francisco in 2019.

Efstathios (Stathis) D. Gennatas received an MBBS degree in Medicine from Imperial College London and a PhD in Neuroscience from the University of Pennsylvania. He was Assistant Professional Researcher at the University of California San Francisco (2017-2019) and a Research Scientist at Stanford University (2019-2020). He will join the Department of Epidemiology and Biostatistics at the University of California, San Francisco in August 2020.