Sequential Training of Neural Networks With Gradient Boosting

SEYEDSAMAN EMAMI AND GONZALO MARTíNEZ-MUñOZ
Escuela Politécnica Superior, Universidad Autónoma de Madrid, 28049 Madrid, Spain
Corresponding author: Seyedsaman Emami (emami.seyedsaman@uam.es)
This work was supported by a research program under Grant PID2019-106827GB-I00/AEI/10.13039/501100011033.

ABSTRACT This paper presents a novel technique based on gradient boosting to train the final layers of a neural network (NN). Gradient boosting is an additive expansion algorithm in which a series of models are trained sequentially to approximate a given function. A neural network can also be seen as an additive expansion where the scalar product of the responses of the last hidden layer and its weights provide the final output of the network. Instead of training the network as a whole, the proposed algorithm trains the network sequentially in $T$ steps. First, the bias term of the network is initialized with a constant approximation that minimizes the average loss of the data. Then, at each step, a portion of the network, composed of $J$ neurons, is trained to approximate the pseudo-residuals on the training data computed from the previous iterations. Finally, the $T$ partial models and bias are integrated as a single NN with $T \times J$ neurons in the hidden layer. Extensive experiments in classification and regression tasks, as well as in combination with deep neural networks, are carried out showing a competitive generalization performance with respect to neural networks trained with different standard solvers, such as Adam, L-BFGS, SGD and deep models. Furthermore, we show that the proposed method design permits to switch off a number of hidden units during test (the units that were last trained) without a significant reduction of its generalization ability. This permits the adaptation of the model to different classification speed requirements on the fly.

INDEX TERMS Gradient boosting, neural network.

I. INTRODUCTION Machine learning is becoming a fundamental piece for the success of more and more applications every day. Some examples of novel applications include bioactive molecule prediction [1], renewable energy prediction [2], classification of galactic sources [3], or agriculture area for mapping soil contamination [4]. It is of capital importance to find algorithms that can efficiently handle complex data. Ensemble methods are very effective at improving the generalization accuracy of multiple simple models [5], [6] or even complex models such as MLPs [7] or DeepCNNs [8].

In recent years, gradient boosting [9], [10], a fairly old technique has gained much attention, specially due to the novel and computationally efficient version of gradient boosting called eXtreme Gradient Boosting or XGBoost [11]. Gradient boosting builds a model as an additive expansion of regressors to gradually minimize a given loss function. When gradient boosting is combined with several stochastic techniques, as bootstrapping or feature sampling from random forest [12], its performance generally improves [13]. In fact, this combination of randomization techniques and optimization has placed XGBoost among the top contenders in Kaggle competitions [11] and provides excellent performance in a variety of applications as in the ones mentioned above. Based on the success of XGBoost, other techniques have been proposed like CatBoost [14] and LightGBM [15], which propose improvements in training speed and generalization performance. More details about these methods can be seen in the comparative analysis of Bentéjac et al. [16]. Other type of widespread boosting algorithm is AdaBoost [17], initially developed for binary classification and then for multi-class classification (AdaBoost-SAMME) [18] and regression [19].
On the other hand, convolutional deep architectures have shown outstanding performances especially with structured data such as images, speech, etc. [20], [21]. However, in the context of tabular data, ensembles of classifiers or simple MLPs are generally more effective than convolutional deep neural networks [22]. In [23], the performance of Deep Neural Networks for tabular data is compared with the traditional machine learning method, such as XGBoost. The study shows that XGBoost outperforms deep models on the analyzed datasets and provides interesting insights to consider when choosing a model for real-life applications, including the model performance, computational inference cost, hyperparameter optimization time, and so on. The objective of our study is to combine the stage-wise optimization of gradient boosting into the training procedure of the last layers of a neural network. The result of the proposed algorithm is an alternative for training a single neural network (not an ensemble of networks) and is specially suited for tabular data.

Several related studies propose hybrid algorithms that, for instance, transform a decision forest into a single neural network [24], [25] or that use a deep architecture to train a tree forest [26]. In [24], it is shown that a pre-trained tree forest can be cast into a two-layer neural network with the same predictive outputs. First, each tree is converted into a neural network. To do so, each split in the tree is transformed into an individual neuron that is connected to a single input attribute (split attribute) and whose activation threshold is set to the split threshold. In this way, and by a proper combination of the outputs of these neurons (splits) the network mimics the behavior of the decision tree. Finally, all neurons are combined through a second layer, which recovers the forest decision. The weights of this network can be later retrained to obtain further improvements [25]. In [26], a decision forest is trained jointly by means of a deep neural network that learns all splits of all trees of the forest. To guide the network to learn the splits of the trees, a procedure that trains the trees using back-propagation, is proposed. The final output of the algorithm is a decision forest whose performance is remarkable in image classification tasks.

In other related line of work [27], [28], boosting is applied to the construction of Deep Residual Learning models [29]. In [27], a novel ResNet weight estimation model is proposed by generalizing the boosting functional gradient minimization [30] to the feature extraction space of the network. The work presented in [28] also builds layer-by-layer a ResNet boosting over features, however, it is based on a different boosting framework [17]. The model (called BoostResNet) works by learning a linear classifier on the output of each residual network block to build an ensemble of shallow blocks. One important advantage of BoostResNet over standard ResNet is its lower computational complexity although the reported performance is not consistently better with respect to ResNet. In contrast, the current proposal method, based on [9], builds a simple shallow network in-width rather than complex model in-depth and shows very good performance in tabular datasets with respect to standard back-propagation training methods. Furthermore, our proposal can adapt on the fly to the use of a reduced number of hidden neurons.

Another model that resembles the idea proposed in this paper —yet with a different optimization process, final model and objective— was presented in [31]. They propose a convex optimization algorithm for training a neural network that theoretically could reach the global optimum although its exact implementation is only feasible for a very low number of input features. In order to reach the global optimum they control the number of hidden neurons of the model by adding one neuron at a time to the network and by including a $L^1$ regularization on the top layer. The proposed idea is a stepwise algorithm as all weights of the network are optimized at each iteration. This is done in three optimization steps. First, a new neuron (i.e. linear model) is added and trained on a weighted loss function similarly to Adaboost. This weighted loss can only be solved exactly for a very low number of input features. Then, the output layer, and potentially all input weights, are optimized using the proposed convex formulation. Finally, the output weights are regularized to reduce the complexity of the network. This final step sets to zero some of the output weights to effectively remove the corresponding neurons. The algorithm is tested on one simple 2D problem in order to assess the validity of the global optimum approach.

In this paper, we propose a combination of ensembles and neural networks that is somehow complementary to the work of [26], which is a single neural network that is trained using an ensemble training algorithm. Specifically, we propose to train a neural network iteratively as an additive expansion of simpler models. The algorithm is equivalent to gradient boosting: first, a constant approximation is computed (assigned to the bias term of the neural network), then at each step, a regression neural network with a single (or very few) neuron(s) in the hidden layer is trained to fit the residuals of the previous models. All these models are then combined to form a single neural network with one hidden layer. This training procedure provides an alternative to standard training solvers (such as Adam, L-BFGS or SGD) for training a neural network. Other works related to the optimization and convergence of Adam have been recently proposed [32], [33], [34]. They showed that its convergence can be improved in the context of high dimensional complex image classification tasks. However, their focus is mainly in deep models for image classification tasks.

In addition, the proposed method has an additive neural architecture in which the latest computed neurons contribute less to the final decision. This can be useful in computation-ally intensive applications as the number of active models (or neurons) can be gauged on the fly to the available computational resources without a significant loss in generalization accuracy. The proposed model is tested on multiple classification and regression problems, as well as in conjunction with deep models posed as transfer learning problems. These
Given a training dataset $D = \{x_i, y_i\}_{i=1}^N$, the goal of machine learning algorithms is to find an approximation, $F(x)$, of the objective function $F^*(x)$, which maps instances $x$ to their output values $y$. In general, the learning process can be posed as an optimization problem in which the expected value of a given loss function, $E[L(y, F(x))], S. Emami, G. Martínez-Muñoz: Sequential Training of NNs With Gradient Boosting is minimized. A data-based estimate can be used to approximate this expected loss: $\sum_{i=1}^N L(y_i, F(x_i))$.

In the specific case of gradient boosting, the model is built using an additive expansion

$$F_t(x) = F_{t-1}(x) + \rho_t h_t(x), \quad (1)$$

where $\rho_t$ is the weight of the $t^{th}$ function, $h_t(x)$. The approximation is constructed stage-wise in the sense that at each step, a new model $h_t$ is built without modifying any of the previously created models included in $F_{t-1}(x)$. First, the additive expansion is initialized with a constant approximation

$$F_0(x) = \arg\min_{a} \sum_{i=1}^N L(y_i, a) \quad (2)$$

and the following models are built in order to minimize

$$\rho_t, h_t(x) = \arg\min_{\rho, h_t} \sum_{i=1}^N L(y_i, F_{t-1}(x) + \rho h_t(x)). \quad (3)$$

However, instead of jointly solve the optimization for $\rho$ and $h_t$, the problem is split into two steps. First, each model $h_t$ is trained to learn the data-based gradient vector of the loss-function. For that, each model, $h_t$, is trained on a new dataset $D = \{x_i, r_{it}\}_{i=1}^N$, where the pseudo-residuals, $r_{it}$, are the negative gradient of the loss function at $F_{t-1}(x_i)$

$$r_{it} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \bigg|_{F(x_i)=F_{t-1}(x)} \quad (4)$$

The function, $h_t$, is expected to output values close to the pseudo residuals at the given data points, which are parallel to the gradient of $L$ at $F_{t-1}(x)$. Note, however, that the training process of $h$ is generally guided by square-error loss, which may be different from the given objective loss function. Notwithstanding, the value of $\rho_t$ is subsequently computed by solving a line search optimization problem on the given loss function

$$\rho_t = \arg\min_{\rho} \sum_{i=1}^N L(y_i, F_{t-1}(x_i) + \rho h_t(x_i)). \quad (5)$$

The original formulation of gradient boosting (as given in [9]) is, in some final derivations, only valid for decision trees. Here, we present an extension of the formulation of gradient boosting to be able to use any possible regressor as base model and we describe how to integrate this process to train a single neural network, which is the focus of the paper.

### B. Binary Classification

For binary classification, in which $y \in \{-1, 1\}$, we will consider the logistic loss

$$L(y, F(\cdot)) = \ln (1 + \exp(-2yF)), \quad (6)$$

which is optimized by the logit function $F(x) = \frac{1}{2} \ln \frac{p(y = 1)}{p(y = -1)}$. For this loss function the constant approximation of Eq. 2 is given by

$$F_0 = \arg\min_{a} \sum_{i=1}^N \ln(1 + \exp(-2y_i\alpha))$$

$$= \frac{1}{2} \ln \frac{p(y = 1)}{p(y = -1)} = \frac{1}{2} \ln \frac{1 - \bar{y}}{1 + \bar{y}}, \quad (7)$$

where $\bar{y}$ is the mean value of the class labels $y_i$. The pseudo-residuals given by Eq. 4 on which the model $h_t$ is trained for the logistic loss can be calculated as

$$r_{it} = (2y_i - 1)\frac{2}{1 + \exp(2y_i F_{t-1}(x_i)))}. \quad (8)$$

Once $h_t$ is built, the value of $\rho_t$ is computed using Eq. 5 by minimizing

$$f(\rho) = \sum_{i=1}^N \ln(1 + \exp(-2y_i(F_{t-1}(x_i) + \rho h_t(x_i))). \quad (9)$$

There is no close form solution for this equation. However, the value of $\rho$ can be approximated by a single Newton-Raphson step

$$\rho_t \approx \frac{f'(\rho = 0)}{f''(\rho = 0)} = \frac{\sum_{i=1}^N r_{it} h_t(x_i)}{\sum_{i=1}^N r_{it}^2 (2y_i - r_{it}) h_t^2(x_i)}. \quad (10)$$

This equation is valid for any base regressor and not only for decision trees as in the general gradient boosting framework [9]. The formulation in [9] is adapted to the fact that decision trees can be seen as piecewise-constant additive models, which allows for the use of a different $\rho$ for each tree leaf.

Finally, the output for binary classification of gradient boosting composed of $T$ models for instance $x$ is given by the probability of $y = 1|x$

$$p(y = 1|x) = \frac{1}{1 + \exp(-2F_T(x))}. \quad (11)$$
C. Multi-Class Classification

For multi-class classification with \( K > 2 \) classes, the labels are defined with 1-of-\( K \) vectors, \( y \), such that \( y_k = 1 \) if the instance belongs to class \( k \) and \( y_k = 0 \), otherwise. In this context the output for \( x \) of the model is also a \( K \)-dimensional vector \( F(x) \). The cross-entropy loss function is used in this context

\[
L(y, F(x)) = -\sum_{k=1}^{K} y_k \ln p_k(x), \tag{12}
\]

where \( p_k(x) \) is the probability of a given instance of being of class \( k \)

\[
p_k(x) = \frac{\exp(F_k(x))}{\sum_{l=1}^{K} \exp(F_l(x))} \tag{13}
\]

The additive model is initialized with constant value 0 as \( F_{k,0} = 0 \ \forall k \), that correspond to a probability equal to \( 1/K \) for all classes and instances.

The pseudo-residuals, given by Eq. 4, on which the model \( h_t \) is trained for the multi-class loss are the derivative of Eq. 12 with respect to \( F_{k,t-1} \)

\[
r_{ik} = \sum_{j=1}^{K} y_{ij} \frac{\partial p_k(x)}{\partial F_k(x)} \bigg|_{F_k(x)=F_{k,t-1}(x)}
= \sum_{j=1}^{K} y_{ij} (\delta_{kj} - p_{k,t-1}(x_i))
= y_{ik} - p_{k,t-1}(x_i) \tag{14}
\]

where \( \delta_{kj} \) is Kronecker delta and the fact that \( \sum_{j=1}^{K} y_{ij} = 1 \ \forall i \) is used in the final step. In this study, a single model, \( h_t \), is going to be trained per iteration to fit the residuals for all \( K \) classes. In contrast to the \( K \) decision trees per iteration that is used in the final step. In this study, a single model, \( h_t \), is going to be trained per iteration to fit the residuals for all \( K \) classes. In contrast to the \( K \) decision trees per iteration that are built in gradient boosting. Then a line search for each of the \( K \) outputs of the model is computed by minimizing

\[
f(\rho_k) = -\sum_{i=1}^{N} y_{ij} \sum_{j=1}^{K} \exp(F_{j,t-1}(x_i) + \rho_j h_{j,t}(x_i)) \ln \left[ \frac{\exp(F_{j,t-1}(x_i) + \rho_j h_{j,t}(x_i))}{\sum_{l=1}^{K} \exp(F_{l,t-1}(x_i) + \rho_l h_{l,t}(x_i))} \right] \tag{15}
\]

with a Newton-Raphson step

\[
\rho_{k,t} = -\frac{f'(\rho_k = 0)}{f''(\rho_k = 0)} - \left[ \frac{\sum_{i=1}^{N} h_{k,t}(x_i)(y_{ik} - p_{k,t-1}(x_i))}{\sum_{i=1}^{N} h_{k,t}^2(x_i)p_{k,t-1}(x_i)(p_{k,t-1}(x_i) - 1)} \right] \tag{16}
\]

In the same way as for Eq. 10, this equation is valid for all types of base learners and not specific for decision trees as in the original formulation \[9\], which opens the possibility to apply gradient boosting to other base learners.
have a single or several units in the hidden layer. In the remainder of this section we will assume that at each step of the boosting procedure a network with one single unit in the hidden layer is used. Generalizing this to larger networks is trivial. Fig 1 (right diagrams) shows, highlighted in black, the regression neural network with a single neuron in the hidden layer that is trained in iteration $t$ and that corresponds to model $h_t$. After model $t$ has been trained, the value of $\rho_t$ is computed using Eq. 10 for binary problems and $\rho_{k,t}$ (Eq. 16) for multi-class classification. Once all $T$ models have been trained, a neural network, as shown in Fig 1 (left diagrams), with $T$ units in the last hidden layer is obtained by assigning all the weights necessary to compute the $z_t$ variables (i.e. $v_t'$ in Fig 1 right) to the corresponding weights in the final NN (i.e. $v_t$ in Fig 1 left) for binary and multi-class tasks

\[ v_t = v_t' \quad t = 1, \ldots, T \]  

(20)

and the weights $\omega_t$ of the output layer for binary classification are assigned to

\[ \omega_0 = F_0 + \sum_{t=1}^{T} 2\rho_t \omega_{t0}' \]  

(21)

\[ \omega_t = 2\rho_t \omega_t' \quad t = 1, \ldots, T \]  

(22)

where the $\omega_t'$ and $\omega_{t0}'$ are the weights from the hidden neuron to the output and the bias term respectively for the $h_t$ model (as shown in Fig 1 right-top diagram). For multi-class with $K$ classes final the assignment is

\[ \omega_{0k} = \sum_{t=1}^{T} \rho_{k,t} \omega_{t0k}' \]  

(23)

\[ \omega_{tk} = \rho_{k,t} \omega_{tk}' \quad t = 1, \ldots, T \quad k = 1, \ldots, K \]  

(24)

where the $\omega_{tk}'$ and $\omega_{t0k}'$ are the output weights for the $h_t$ multi-class model (Fig 1 right-bottom diagram).

Finally, to recover the probability of $y = k|\mathbf{x}$ in the output of the NN as given by Eq. 11 and Eq. 17, the activation function should be a sigmoid (i.e. $\sigma(x) = 1/(1 + \exp(-x))$) for binary classification with the logistic loss (Eq. 6), and the soft-max activation function (i.e. $\sigma(x) = \exp(x_k)/\sum_k \exp(x_k)$) for multi-class tasks when using the cross-entropy loss (Eq. 12). This training procedure can be easily modified to larger increments of neurons, so that instead of a single neuron per step (a linear model), a more flexible model, comprising more units ($J$), can be trained at each iteration. The outline of the proposed method is shown in Algorithm 1.

The proposed training procedure can be further tuned by applying subsampling and/or shrinking, as generally used in gradient boosting [9], [11], [13]. In shrinking, the additive expansion process is regularized by multiplying each term $\rho_t h_t$ by a constant learning rate, $\nu \in (0,1)$, to prevent overfitting when multiple models are combined [9]. Subsampling consists of training each model on a random subsample without replacement from the original training data. Subsampling has shown to improve the performance of gradient boosting [13].

The overall computational time complexity to train the proposed algorithm is the same as that of a standard NN with the same number of hidden neurons and training epochs. Note that, at each step of the proposed algorithm, one NN with one (or few) neurons in the hidden layer is trained. Hence, at the end of the process, the same number of weights updates are performed. Note, however, that the proposed method is sequential and cannot be easily parallelized. Once the model is trained, the computational complexity for classifying new instances is equivalent to that of a standard NN as the generated model is a standard neural network.

### Algorithm 1 Training Neural Network as an Additive Expansion

**Input**

1. Input data $D = \{\mathbf{x}_i, y_i\}_{i=1}^N$
2. Number of neurons $T$
3. Loss function

**Training the model**

1. Initialize $\hat{F}_0$
2. for $t = 1$ to $T$
3. Compute the pseudo-residual with Eqs. 8 14 ($r_{it}$ for binary and $r_{it}$ for multi-class classification)
4. Fit a new regressor network model on the residuals.
5. Compute the gradient descent step through the Newton-Raphson step ($\rho_t$ for binary, Eq. 10 and $\rho_{kt}$ for multi-class classification, Eq. 16)
6. Update the model using Eq. 1
7. end for
8. Create the final network by assigning the weights with Eqs. 20 22 24

**E. ILLUSTRATIVE EXAMPLE**

To illustrate the workings of this algorithm, we show its performance in a toy classification problem. The toy problem task consists in a 2D version of the ringnorm problem [35]: where both classes are 2D Gaussian distribution, one with $(0,0)$ mean and covariance four times the identity matrix, and the second class with mean value at $(2/\sqrt{2}, 2/\sqrt{2})$ and the identity matrix as covariance.

The proposed gradient boosted neural network (GBNN) with $T = 100$ and one hidden layer is trained on 200 randomly generated instances of this problem. In addition, 100 independent neural nets with hidden units in the range $[1,100]$ are also trained using the same training set. In Fig 2, the boundaries for the different stages of the process are shown graphically. In detail, the first and second rows show the results for GBNN and NN, respectively. Each column shows the results for $T = 1, T = 2, T = 3, T = 4$ and $T = 100$ neurons in the hidden layer respectively. Note that the plots for GBNN are sequential; That is, the first column
shows the first trained model, the second column, the first two models combined, and so on. For the NN, each column corresponds to a different NN with a different number of neurons in the hidden layer. For each column, the architecture of the networks and the number of weights (but not their values) are the same. The color of the plots represent the probability $p(y = 1 \mid x)$ given by the models (Eq. 11) using the viridis colormap. In addition, all plots show the training points.

As we can see both GBNN and NN start, with very similar models (column $T = 1$). As the number of models (neurons for NN) increases, GBNN builds up the boundary from previous models. On the other hand, the standard neural network, as it creates a new model for each size, is able to adjust faster to the data. However, as the number of neurons increases, NN tends to overfit in this problem (as shown in the bottom right-most plot). On the contrary, GBNN tends to focus on the unsolved parts of the problem: the decision boundary becomes defined only asymptotically, as the number of models (neurons) becomes large.

### III. EXPERIMENTAL RESULTS

In this section, an analysis of the efficiency of the proposed neural network training method based on gradient boosting is tested on twelve binary classification tasks, eight multi-class problems, seven regression datasets from the UCI repository [36], and two datasets related to image processing [37], [38]. These datasets, shown in Table 1, have different number of instances and attributes and come from different fields of application. The Energy dataset has two different target columns (cooling and heating), so the different algorithms were executed for both objectives separately. We modified some of the datasets. For Diabetes duplicated instances and instances with missing values were removed. In addition, categorical values were substituted by dummy variables in: German Credit Data, Hepatitis, Indian Liver Patient, MAGIC and Tic-tac-toe. Finally, CIFAR-10 and MNIST were normalized by dividing the attributes by 255.

Two batches of experiments were carried out: experiments on tabular data and experiments on image and large datasets. In the first batch, the proposed method is compared with respect to standard neural networks using different solvers and with respect to dense deep neural networks. In the second batch, a transfer learning approach was followed, training the last layer of deep models with the proposed method and fully connected NN. The first experiment is carried out in all the classification and regression tasks shown in Table 1, except for Covertype, Poker Hand, MNIST and CIFAR-10. For these large datasets, the comparison of the proposed method was carried out with respect to deep dense or convolutional neural networks, depending on the type of problem. For the first batch of experiments, the scikit-learn package [39] was used. For the second batch, we adopted Keras library [40]. The implementation of the proposed method (GBNN) is done in python following the standards of scikit-learn. The implementation of the algorithm is available on https://github.com/GAA-UAM/GBNN/.

#### A. EXPERIMENTS WITH TABULAR DATA

For the first set of experiments on tabular regression and classification, single hidden layer networks were trained using three different standard solvers (Adam, L-BFGS and SGD), and using the proposed gradient boosting approach. Also, we considered a three-layer deep dense neural network trained with the Adam solver. Furthermore, AdaBoost [17] using small neural networks as the base models was also included in the comparison (AdaBoost–NN). Note that, this approach is different to the proposed GBNN in two aspects. First, for classification tasks, the base models in AdaBoost are classifiers that are combined by weighted majority voting. Hence, the final model is a collection of small base classifiers and not a single neural network as in GBNN. Second, AdaBoost is based on modifying the instance weights during training so that difficult instances tend to get higher weights. This poses a difficulty in the training of the neural networks as they do not handle weighted instances. In order to run Adaboost with neural networks we included a weighted resampling step prior the training of each individual network. The scikit-learn library does not include this functionality in Adaboost. The comparison for classification and regression problems was carried out using 5 × 10-fold cross-validation in order to have stable results. In addition, the same splits were used for training of each individual network.

| Dataset                  | Area         | Instances | Attributes |
|--------------------------|--------------|-----------|------------|
| Australian Credit Approval | Financial    | 690       | 14         |
| German Credit Data       | Financial    | 1,000     | 20         |
| Banknote                 | Computer     | 1,372     | 5          |
| Spambase                 | Computer     | 4,601     | 57         |
| Tic-tac-toe              | Game         | 958       | 9          |
| Breast cancer            | Life         | 569       | 32         |
| Diabetes                 | Life         | 381       | 9          |
| Hepatitis                | Life         | 155       | 19         |
| Indian Liver Patient     | Life         | 583       | 10         |
| MAGIC Gamma Telescope    | Physical     | 19,020    | 11         |
| Ionosphere               | Physical     | 351       | 34         |
| Sonar                    | Physical     | 208       | 60         |
| **Multi-class classification** |             |           |            |
| Digits                   | Computer     | 1,797     | 64         |
| Poker Hand               | Game         | 1,025,010 | 10         |
| Iris                     | Life         | 150       | 4          |
| Covertype                | Life         | 581,012   | 54         |
| Vehicle                  | Transportation | 946     | 18         |
| Vowel                    | Education    | 990       | 11         |
| Waveform                 | Physical     | 5,000     | 21         |
| Wine                     | Physical     | 178       | 13         |
| **Regression**           |              |           |            |
| Concrete                 | Physical     | 1,030     | 9          |
| Energy                   | Computer     | 768       | 8          |
| Power                    | Computer     | 9,568     | 4          |
| Boston Housing           | Business     | 506       | 14         |
| Wine quality-red         | Business     | 1,599     | 12         |
| Wine quality-white       | Business     | 4,828     | 12         |
| **image processing**     |              |           |            |
| MNIST                    | handwritten digits | 60,000 | 28x28 |
| CIFAR-10                 | labeled images | 60,000 | 32x32 |

| **A. EXPERIMENTS WITH TABULAR DATA** | | |
|--------------------------------------|-------------------|
| **Dataset**                         | **Area**          | **Instances** | **Attributes** |
| Australian Credit Approval           | Financial         | 690           | 14             |
| German Credit Data                   | Financial         | 1,000         | 20             |
| Banknote                             | Computer          | 1,372         | 5              |
| Spambase                             | Computer          | 4,601         | 57             |
| Tic-tac-toe                          | Game              | 958           | 9              |
| Breast cancer                        | Life              | 569           | 32             |
| Diabetes                             | Life              | 381           | 9              |
| Hepatitis                            | Life              | 155           | 19             |
| Indian Liver Patient                 | Life              | 583           | 10             |
| MAGIC Gamma Telescope                | Physical          | 19,020        | 11             |
| Ionosphere                           | Physical          | 351           | 34             |
| Sonar                                | Physical          | 208           | 60             |
| **Multi-class classification**       | **Area**          | **Instances** | **Attributes** |
| Digits                               | Computer          | 1,797         | 64             |
| Poker Hand                           | Game              | 1,025,010     | 10             |
| Iris                                 | Life              | 150           | 4              |
| Covertype                            | Life              | 581,012       | 54             |
| Vehicle                              | Transportation    | 946           | 18             |
| Vowel                                | Education         | 990           | 11             |
| Waveform                             | Physical          | 5,000         | 21             |
| Wine                                 | Physical          | 178           | 13             |
| **Regression**                       | **Area**          | **Instances** | **Attributes** |
| Concrete                             | Physical          | 1,030         | 9              |
| Energy                               | Computer          | 768           | 8              |
| Power                                | Computer          | 9,568         | 4              |
| Boston Housing                       | Business          | 506           | 14             |
| Wine quality-red                     | Business          | 1,599         | 12             |
| Wine quality-white                   | Business          | 4,828         | 12             |
| **image processing**                 |                   |               |                |
| MNIST                                | handwritten digits| 60,000        | 28x28          |
| CIFAR-10                             | labeled images    | 60,000        | 32x32          |

#### TABLE 1. The details of datasets, which used in experimental analysis.

| Dataset                  | Area         | Instances | Attributes |
|--------------------------|--------------|-----------|------------|
| Australian Credit Approval | Financial    | 690       | 14         |
| German Credit Data       | Financial    | 1,000     | 20         |
| Banknote                 | Computer     | 1,372     | 5          |
| Spambase                 | Computer     | 4,601     | 57         |
| Tic-tac-toe              | Game         | 958       | 9          |
| Breast cancer            | Life         | 569       | 32         |
| Diabetes                 | Life         | 381       | 9          |
| Hepatitis                | Life         | 155       | 19         |
| Indian Liver Patient     | Life         | 583       | 10         |
| MAGIC Gamma Telescope    | Physical     | 19,020    | 11         |
| Ionosphere               | Physical     | 351       | 34         |
| Sonar                    | Physical     | 208       | 60         |
| **Multi-class classification** |             |           |            |
| Digits                   | Computer     | 1,797     | 64         |
| Poker Hand               | Game         | 1,025,010 | 10         |
| Iris                     | Life         | 150       | 4          |
| Covertype                | Life         | 581,012   | 54         |
| Vehicle                  | Transportation | 946     | 18         |
| Vowel                    | Education    | 990       | 11         |
| Waveform                 | Physical     | 5,000     | 21         |
| Wine                     | Physical     | 178       | 13         |
experiments that showed that L-BFGS provided good results for all datasets. This decision was made after some preliminary analysis. For the sub-networks are trained using the L-BFGS solver for the learning rate, $[0, 25, 0, 025]$. Moreover, for the proposed procedure, the number of neurons were set respectively to the following pairs: (50, 4), (100, 2) and (67, 3). For multi-class problems the pair (50, 4) was also considered. This is done in order to produce a model with a combined total number of 200 hidden neurons as for previous networks. Subsequently, the best sets of hyper-parameters obtained in the grid search for each method were used to train the whole training set, one standard neural network for each solver, dense deep neural networks, AdaBoost–NN, and the proposed gradient boosted neural network. Finally, the average generalization performance of the models was estimated in the left-out test set.

For all analyzed datasets, the average generalization performance and standard deviations are shown in Table 2 for the proposed method (column GBNN), the three-layer deep neural network (column Deep-NN), standard neural networks trained with Adam (NN–Adam), L-BFGS (NN–L-BFGS), SGD (NN–SGD), and AdaBoost–NN. For classification problems the generalization performance is given as average accuracy and, for regression, as average root mean square error. The Table 2 is structured in three blocks depending on the problem type: binary classification tasks, multi-class classification, and regression. The best result for each method is highlighted with a light yellow background.

FIGURE 2. Classification boundaries for gradient boosted neural network (top row) and for a standard neural network (bottom row). Each column shows the results for a combination of a different number of models (hidden units). Top plots are the sequential results of a single GBNN model, whether bottom plots are independent neural networks models.
although the differences between these methods are small:

In classification, the differences in average accuracy among the different methods are

respectively. And finally, the AdaBoost–NN got the best

solvers obtained the best outcome in two and one datasets,

tured the best results in three datasets. SGD and L-BFGS

The neural network method with Adam as its solver cap-

26 tasks. The deep-NN model performed best in five datasets.

GBNN. This method reached the best results in 13 out of

seven regression and all datasets respectively, considering six

reference (CD

is considered significant is shown in the plot for

statistical differences between methods are determined using

Demsˇar plot for binary and multiclass datasets. b) The Dems ˇar plot for

FIGURE 3. Average ranks (higher rank is better) for GBNN, Deep-NN,

NN–Adam, NN–L–BFGS, NN–sgd and AdaBoost–NN for 26 datasets. a) The

plots show the average rank for the studied methods across

analyzed datasets where a higher rank (i.e. lower values)

indicates better results. The figure shows the Demsár plots

for the classifications tasks only (subplot a), for regression

only (subplot b), and for all analyzed tasks (subplot c). The

statistical differences between methods are determined using

a Nemenyi test. In the plot, the difference in the average rank

of the two methods is statistically significant if the methods

are not connected with a horizontal solid line. The critical
distance (CD) in average rank over which the performance of

two methods is considered significant is shown in the plot for

reference (CD = 1.72, 2.84, and 1.47 for 19 classification,

seven regression and all datasets respectively, considering six

methods and p-value < 0.05).

From Table 2 and Fig 3, it can be observed that, for

the studied datasets, the best overall performing method is

GBNN. This method reached the best results in 13 out of

26 tasks. The deep-NN model performed best in five datasets.
The neural network method with Adam as its solver captured
the best results in three datasets. SGD and L-BFGS solvers

obtained the best outcome in two and one datasets,

respectively. And finally, the AdaBoost–NN got the best

performance in three datasets. In classification, the differences

in average accuracy among the different methods are
generally favorable to GBNN, NN–Adam and Deep-NN,

although the differences between these methods are small

in many datasets. For instance, in Magic, the accuracy of

GBNN is only 0.13 percent points worse than NN–Adam.

However, small differences are not always the case. One of

the most notable differences between GBNN and second top

ranked method is in Tic-tac-toe where Deep–NN is 2.92% worse

than the result obtained by GBNN and NN-Adam is

more than eight points worse. In contrast, the most favorable

outcome for Deep–NN is obtained in Vowel and Sonar, where

its accuracy is 6.83% and 7.38% better than that of GBNN,

respectively. The results in classification for NN–L–BFGS,

NN–SGD and AdaBoost–NN are generally worse than those

of the other three methods.

In regression, the results are more clearly in favor

of GBNN. The proposed method obtains the best performance

in all tested datasets except for one dataset, Concrete, in which

GBNN get the second best result. The performance of

NN–Adam in regression is suboptimal getting the worst per-

formances in general. Solver L-BFGS perform more closely

to the performance of GBNN.

These results can also be observed from Fig 3. The

performance of NN–L–BFGS, NN–SGD and AdaBoost–NN in

the classification tasks is worse than that of NN–Adam,

GBNN and Deep–NN (subplot a). NN-Adam and GBNN take

the highest rank in the statistical test. For regression

(subplot b), the best-performing method is GBNN. The

performance of GBNN is significantly better than that of NN–

Adam, NN–SGD and AdaBoost–NN. The overall results

(subplot c) show that GBNN has the best rank followed

by NN–Adam and Deep–NN. Overall, the performance

of GBNN is statistically better than the performance of NN–

SGD and AdaBoost-NN.

To analyze the evolution of the networks generated by the

proposed method as more hidden units are included, another

experiment was carried out to build a GBNN with 200 hidden

units, and neural networks trained using the Adam solver with

1 to 200 neurons in the hidden layer. In addition, the final

accuracy of a three-layer deep neural network with 100 neu-

rons in each layer is also computed for reference. For this

experiment, 10-fold cross-validation was used. The average

evolution is shown in Fig 4 for Spambase (left plot), Tic-
tac-toe (middle plot) and Energy-Heating (right plot). Due to

computational limitations in this case, the hyper-parameters

were not tuned for each partition. Instead, they were set to

the values more often selected in the previous experiment for
each dataset. Specifically, they were set to \( J = 3 \), learning

rate to 0.5 and subsampling to 1 for Tic-tac-toe, to \( J = 1 \),

learning rate to 0.5 and subsampling to 0.75 for Spambase and
to \( J = 4 \), learning rate to 0.5 and subsampling to 1 for Energy-

Heating for all partitions. Note that for each sequence of

GBNN, only one model is trained. For NN, 200 independent

models with 1 to 200 neurons need to be trained in order to

obtain the sequence.

From Fig 4, it can be observed that the average test

accuracy of GBNN improves as more units are considered.

More importantly, we can observe that GBNN tends to sta-

bilize with the number of units. Hence, if the latter units
TABLE 2. Average generalization performance and standard deviation for gradient boosted neural network (GBNN), Deep neural network, neural networks trained with solvers Adam, L-BFGS, SGD and AdaBoost. Using accuracy for the classification tasks and root mean square error (RMSE) for regression problems as the performance measurement. The best results for each dataset are highlighted with a light yellow background.

| Dataset                        | GBNN       | Deep-NN    | NN–Adam    | NN–L-BFGS  | NN–SGD     | AdaBoost–NN |
|-------------------------------|------------|------------|------------|------------|------------|-------------|
| **Binary classification**     |            |            |            |            |            |             |
| Australian Credit Approval    | 85.91% ± 4.81 | 82.81% ± 4.26 | 85.88% ± 4.43 | 86.20% ± 4.32 | 86.35% ± 4.28 | 85.48% ± 3.32 |
| Banknote                      | 99.99% ± 0.04 | 100.00% ± 0.00 | 99.99% ± 0.04 | 99.91% ± 0.42 | 97.32% ± 1.15 | 100.00% ± 0.00 |
| Breast cancer                 | 96.87% ± 1.93 | 97.40% ± 2.06 | 97.36% ± 1.97 | 96.83% ± 1.55 | 96.55% ± 1.69 | 96.45% ± 4.72 |
| Diabetes                      | 76.12% ± 3.93 | 71.30% ± 4.59 | 76.35% ± 4.05 | 76.67% ± 4.15 | 76.96% ± 4.21 | 74.81% ± 3.25 |
| German Credit Data            | 74.16% ± 3.64 | 69.84% ± 4.53 | 73.74% ± 3.99 | 72.34% ± 3.41 | 73.28% ± 3.87 | 72.44% ± 4.42 |
| Hepatitis                     | 82.81% ± 10.60 | 84.10% ± 10.83 | 85.10% ± 10.86 | 82.11% ± 9.21 | 85.09% ± 10.65 | 85.54% ± 7.82 |
| Indian Liver Patient          | 72.51% ± 5.30 | 70.88% ± 5.20 | 70.69% ± 5.81 | 69.44% ± 5.88 | 71.10% ± 5.52 | 69.26% ± 4.84 |
| Ionosphere                    | 90.94% ± 4.86 | 93.28% ± 3.69 | 91.34% ± 3.92 | 90.43% ± 4.58 | 87.90% ± 5.96 | 92.25% ± 3.52 |
| MAGIC Gamma Telescope         | 87.52% ± 0.67 | 85.62% ± 0.85 | 87.65% ± 0.57 | 87.58% ± 0.62 | 86.28% ± 1.48 | 87.19% ± 0.75 |
| Sonar                         | 78.84% ± 7.30 | 86.22% ± 7.57 | 85.56% ± 6.48 | 85.29% ± 5.07 | 77.67% ± 7.67 | 85.91% ± 8.26 |
| Spambase                      | 94.44% ± 1.00 | 94.08% ± 1.22 | 94.61% ± 1.01 | 93.43% ± 1.27 | 93.42% ± 1.27 | 74.20% ± 13.16 |
| Tic-tac-toe                   | 98.70% ± 1.17 | 95.78% ± 1.80 | 90.44% ± 3.19 | 93.53% ± 2.56 | 79.94% ± 4.30 | 87.65% ± 3.69 |
| **Multi-class classification**|            |            |            |            |            |             |
| Digits                        | 97.18% ± 1.21 | 97.55% ± 1.06 | 98.04% ± 1.11 | 97.17% ± 1.10 | 96.44% ± 1.09 | 95.66% ± 1.18 |
| Iris                          | 95.73% ± 6.02 | 94.40% ± 6.58 | 95.33% ± 5.58 | 94.93% ± 6.57 | 85.07% ± 8.71 | 94.72% ± 2.16 |
| Vehicle                       | 84.61% ± 3.92 | 83.41% ± 3.82 | 83.43% ± 3.52 | 82.96% ± 4.21 | 72.99% ± 4.51 | 72.13% ± 4.15 |
| Vowel                         | 89.86% ± 2.57 | 96.71% ± 1.96 | 94.28% ± 2.36 | 93.13% ± 3.09 | 93.04% ± 4.34 | 51.45% ± 3.37 |
| Waveform                      | 87.00% ± 1.16 | 82.60% ± 1.62 | 86.65% ± 1.40 | 86.46% ± 1.22 | 86.68% ± 1.37 | 84.99% ± 1.46 |
| Waveform-300                  | 82.94% ± 0.17 | 82.07% ± 0.63 | 83.69% ± 0.70 | 79.51% ± 0.02 | 83.92% ± 0.77 | 84.55% ± 0.40 |
| Wine                          | 98.88% ± 2.35 | 97.87% ± 3.35 | 97.77% ± 3.22 | 97.65% ± 3.50 | 97.72% ± 4.91 | 97.32% ± 3.48 |
| **Regression**                |            |            |            |            |            |             |
| Boston Housing                | 3.03% ± 0.74 | 3.18% ± 0.79 | 4.12% ± 0.77 | 3.50% ± 0.98 | 3.40% ± 0.87 | 14.43% ± 1.49 |
| Concrete                      | 4.80% ± 0.59 | 5.20% ± 0.56 | 9.61% ± 0.58 | 4.73% ± 0.59 | 6.04% ± 0.47 | 26.78% ± 1.10 |
| Energy-Cooling                | 0.95% ± 0.16 | 2.16% ± 0.31 | 3.45% ± 0.46 | 1.14% ± 0.17 | 3.12% ± 0.41 | 15.91% ± 1.69 |
| Energy-Heating                | 0.43% ± 0.08 | 1.40% ± 0.27 | 2.96% ± 0.38 | 0.49% ± 0.06 | 2.75% ± 0.33 | 12.95% ± 1.29 |
| Power                         | 3.84% ± 0.18 | 4.40% ± 0.39 | 4.25% ± 0.16 | 4.11% ± 0.17 | 4.14% ± 0.15 | 123.37% ± 9.70 |
| Wine quality-red              | 0.60% ± 0.04 | 0.68% ± 0.06 | 0.64% ± 0.05 | 0.64% ± 0.07 | 0.65% ± 0.04 | 0.83% ± 0.06 |
| Wine quality-white            | 0.67% ± 0.03 | 0.72% ± 0.04 | 0.68% ± 0.03 | 0.70% ± 0.03 | 0.70% ± 0.03 | 0.74% ± 0.02 |

FIGURE 4. Average generalization accuracy for GBNN (blue curve), NN (orange curve), and Deep-NN (green curve) with respect to the number of hidden units.

are removed, the performance in the model accuracy is not damaged to a great extent. For instance, in Spambase, if the number of units is reduced from $T = 200$ to $T = 100$ the model accuracy only drops from 94.78% to $\approx 94.63%$. In Tic-tac-toe the same size reduction does not reduce the accuracy of the model. This observation also applies for the Energy-Heating regression task. This property can be useful, as one can adopt a single model to different computational requirements on the fly. This shows that the proposed method is not very prone to over-fitting when more units are included. This could be explained by the fact that later units in GBNN are trained on smaller pseudo residuals. In consequence, later models (units) have less influence in the final NN. This is consistent with gradient boosting ensembles, in which the important aspect is to tune other regularization hyper-parameters, like the learning rate, instead of the number of trees (see
TABLE 3. Total training time in seconds for gradient boosted neural network (GBNN), deep neural network, and neural networks with Adam solver.

| Dataset                  | GBNN | Deep-NN | NN-Adam | NN-L-BFGS | NN-SGD |
|--------------------------|------|---------|---------|-----------|--------|
| Binary classification    |      |         |         |           |        |
| Australian Credit Approval | 1.105 | 0.236   | 0.093   | 0.614     | 0.359  |
| Banknote                 | 0.454 | 0.391   | 0.515   | 0.065     | 1.055  |
| Breast cancer             | 0.165 | 0.295   | 0.305   | 0.510     | 0.054  |
| Diabetes                 | 0.882 | 0.253   | 0.698   | 0.619     | 0.221  |
| German Credit Data       | 0.129 | 0.187   | 0.208   | 0.130     | 0.524  |
| Hepatitis                | 0.316 | 0.060   | 0.023   | 0.247     | 0.046  |
| Indian Liver Patient     | 0.557 | 0.220   | 0.288   | 0.073     | 0.400  |
| Ionosphere               | 0.791 | 0.618   | 0.431   | 0.085     | 0.400  |
| MAGIC Gamma Telescope    | 5.273 | 37.267  | 6.177   | 12.931    | 5.038  |
| Spambase                 | 2.240 | 2.461   | 0.903   | 5.057     | 0.639  |
| Sonar                    | 0.568 | 0.581   | 0.327   | 0.166     | 0.307  |
| Tic-tac-toe              | 0.732 | 1.215   | 1.011   | 0.413     | 0.956  |
| Multi-class classification|      |         |         |           |        |
| Digits                   | 0.976 | 1.095   | 1.016   | 0.463     | 2.569  |
| Iris                     | 1.288 | 0.321   | 0.108   | 0.142     | 0.102  |
| Vehicle                  | 0.256 | 0.519   | 0.269   | 0.549     | 0.391  |
| Vowel                    | 1.085 | 1.992   | 1.419   | 0.578     | 1.085  |
| Waveform                 | 3.896 | 8.676   | 5.452   | 3.644     | 5.146  |
| Wine                     | 0.116 | 0.348   | 0.022   | 0.071     | 0.041  |
| Regression               |      |         |         |           |        |
| Boston Housing           | 0.636 | 1.026   | 0.468   | 0.344     | 0.030  |
| Concrete                 | 0.453 | 0.650   | 0.672   | 0.673     | 0.045  |
| Energy-Cooling           | 0.077 | 0.768   | 0.632   | 0.573     | 0.037  |
| Energy-Heating           | 0.083 | 1.023   | 0.643   | 0.488     | 0.037  |
| Power                    | 0.198 | 4.093   | 1.709   | 1.330     | 0.321  |
| Wine quality-red         | 1.192 | 1.748   | 0.865   | 0.995     | 0.450  |
| Wine quality-white       | 1.148 | 1.953   | 2.647   | 3.150     | 0.220  |

On the other hand, the performance of NN could be higher than that of GBNN at some stages (as shown for Spambase), however, the number of hidden neurons to use have to be decided during train to reduce over-fitting. In addition, the performance of two neural networks with $T$ and $T+1$ trained on the same data present a higher variance than a single GBNN model using $T$ and $T+1$. Hence, the performance of the sequence of NN is not as monotonic as the sequence of GBNN.

In order to compare the computational performance of the tested methods, the training time for GBNN, Deep-NN, NN–Adam, NN-L-BFGS and NN-SGD are shown in Table 3. For this experiment, we applied 10-fold cross-validation and computed the average fit time of the final model that used the set hyper-parameter most frequently selected in the cross-validation. The size for GBNN and the different networks was set to 200. Deep-NN was built, as before with three layers of 100 neurons each. GBNN uses the solver L-BFGS. This experiment is done on CPU using AMD Ryzen 7 5800H 3.20 GHz processor. As Table 3 illustrates, the most computationally efficient method is NN–SGD followed by NN–L-BFGS. In general, the differences are rather contained among most methods and datasets. Some exceptions include the Deep-NN with respect to the other methods, which could be up to $\times 20$ slower (e.g. Power or Energy-Heating). The variance in the average times among the different methods for each dataset is also due to the fact that the default hyper-parameters for all methods include an early-stopping strategy in which the training stops if the loss does not drop at least 0.0001 in ten training iterations. In order to visualize this effect, in Fig 5 we show the average training time with respect of the number of neurons in the hidden layers for GBNN, Deep-NN and NN-Adam for two datasets (one favorable to GBNN and one favorable to NN). The left plot shows the results considering early stopping and the right plot forces the networks to train for 200 epochs. Note that the results of GBNN are monotonic since a single model is used to obtain the training time sequence. For Deep-NN and NN-Adam, a different model is trained every 20 hidden neurons, which explains the peaks in the curves especially when early stopping is active (left plots). When the models are forced to train for 200 epochs, the evolution of the shallow models show a clear linear complexity with respect to the number of hidden units. The deep model shows a quadratic complexity in accordance with the growth of the number of weights. In addition, the training time performance of GBNN is also affected by the model setting. This can account for the favorable and unfavorable results for GBNN shown in Fig 5. In particular, in some exploratory experiments we carried out using different learning rates illustrated that the higher the learning rate, the slower the training time specially for classification.

B. EXPERIMENT WITH LARGE DATASETS AND DEEP MODELS

For the second batch of experiment, three-layer deep dense network and CNNs were used. In order to compare the deep models with the proposed method, we followed a transfer learning approach. For this, once the deep models are trained, the last dense layer and the output layer of the CNN and Deep-NN are removed. Then, the weights of the first layers are frozen. Finally, a GBNN model is trained linked to the frozen layers using the same training instances. We term this model as Deep-GBNN. For Covertype and Poker Hand, a three-layer deep dense neural network with 100 hidden networks per layer were trained for 200 epochs. The activation functions are ReLu for the internal layers and Sigmoid for the output layer. The solver SGD was applied. For CIFAR-10 and MNIST, a CNN was trained for 200 epochs on the training set. The CNN includes the following layers: two convolutions (32), max-pooling ($2 \times 2$), dropout (0.2), two convolutions (64), max-pooling ($2 \times 2$), dropout (0.3), two convolutions (128), max-pooling ($2 \times 2$), dropout (0.4), two dense (128), and one output dense layer for ten classes. The activation functions are ReLu for the internal layers and SoftMax for the output layer. For these datasets, a single partition train/test was carried out. The dataset is divided into a training and a test set as defined by the dataset: 50,000 train and 10,000 test for CIFAR-10; 60,000 train and 10,000 test for MNIST and 25,010/1,000,000 for Poker. For Covertype, a random stratified partition of
TABLE 4. Average generalization performance (first two columns) and in-train performance (second two columns) for deep gradient boosted neural networks (Deep-GBNN) and Deep Convolutional Neural Networks (Deep-CNN).

| Dataset    | Test estimation | In-train estimation |
|------------|-----------------|---------------------|
|            | Deep-GBNN       | Deep-CNN            |
| CIFAR-10   | 84.09           | 84.12               |
| MNIST      | 99.52           | 99.46               |
| CoverType  | 90.00           | 88.24               |
| Poker Hand | 99.45           | 99.33               |

TABLE 5. The generalization performance of the fine-tuned transfer learning models for each model (InceptionV3 and VGG16). And generalization and in-train performance of a one-layer GBNN and NN-trained models on the output of pre-trained models on the CIFAR-10.

| Pre-trained model | Fine-tuned Test | GBNN Test | NN Test | Fine-tuned In-train | GBNN In-train | NN In-train |
|-------------------|-----------------|-----------|---------|---------------------|---------------|-------------|
| InceptionV3       | 93.12           | 93.20     | 99.99   | 93.04               | 99.98         | 99.98       |
| VGG16             | 92.92           | 92.99     | 99.98   | 93.00               | 99.98         | 99.98       |

FIGURE 5. Average training time (in seconds) for gradient boosted neural network (blue curve), standard neural network (orange), and deep neural network (green), for waveform (top row) and Indian liver patient (bottom row). In the left column, the default early stopping procedure was set to train the networks of all models (tolerance is set to 10−4 and iterations with no change to ten). In the right column, all networks are force to train during 200 epochs (tolerance is set to 0 and max iterations without change to 200).

70%-30% was done. The optimum hyper-parameters configuration was measured using within-train 5-fold cross-validation, except for Covertype where a 2-fold cross-validation was used. For Deep-GBNN, the values for the grid search were in the following ranges: [0.1, 1] for the learning rate, [0.25, 1] for subsample and [1, 15] for the step size. Also, the solver for the GBNN method was set to Adam, due to better performance in dealing with high-dimensional datasets. In addition, we run the final experiment using pre-trained networks. For that, we used the InceptionV3 [42] and VGG16 [43] models pre-trained on the imagenet dataset [44]. These models were loaded and subsequently fine-tuned on the CIFAR-10 dataset using the default train partition composed of 50,000 instances. They are validated on the remaining 10,000 test instances. Finally, we froze the weights of VGG16 and InceptionV3 and replaced their last dense layer with a optimized GBNN and NN with 500 units, in which the optimization had done using grid search using train. Finally, the accuracy of NN and GBNN is estimated on the test set.

The Table 4 presents the achieved accuracy for four datasets in the test set (first two columns) and the estimation of the generalization accuracy obtained in the in-train cross-validation (last two columns). The best in-train and test results are highlighted with a yellow background. The results show uneven performance for the different dataset. The proposed method manages to improve the performance of the deep models in MNIST, Cover type and Poker hand by 0.06, 1.76 and 0.12 percentage points respectively. In CIFAR-10, performance drops by 0.03%. Even if the differences are in general marginal, an interesting aspect is that the in-train performance can be used to select the best model on each dataset except in MNIST. Although the differences between method in this dataset both in-train and in test are negligible. Finally, the performances of the GBNN and NN models trained on the output of InceptionV3 and VGG16, for CIFAR-10, and the fine-tuned transfer learning models as well are shown in Table 5. The application of GBNN on top of the pre-trained and fine-tuned models, gained 0.08 and 0.07 percent points in accuracy with respect to InceptionV3 and VGG16 respectively. The results of NN are 0.08 percentage points worse and 0.08 better than the deep fine-tuned models. The accuracy improvements of using GBNN on top of the deep models is small but as it is the additional computational training cost. The classification computational cost remains the same.

IV. CONCLUSION

In this paper, we present a novel iterative method to train a neural network based on gradient boosting. The proposed algorithm builds at each step a regression neural network with one or few (J) hidden unit(s) fitted to the data residuals. The weights of the network are then updated with a Newton-Raphson step to minimize a given loss function. Then, the data residuals are updated to train the model of the next iteration. The resulting T regressors constitute a single neural network with T × J hidden units. In addition, the formulation derived for this works opens the possibility to create gradient boosting ensembles composed of base models different from decision trees, as done in previous implementations of gradient boosting.

In the analyzed problems, the proposed method achieves a generalization accuracy that converges with the number of combined regressors (or hidden units). This quality of the proposed method allows us to use the combined model fully or partially by deactivating the units in order...
inverse to their creation, depending on classification speed requirements. This can be done on the fly during the test. In addition, we showed that the training complexity is equivalent to that of training a network with a standard solver.

The proposed method tested on a variety of classification, regression and image processing tasks. The results show a performance favorable to the proposed method in general. 

The proposed approach showed the best overall average rank in the tested classification and regression problems with statistically significant differences with respect to SGD and L-BFGS approaches. In addition, for deep models a transfer learning approach was followed and the results were favorable to the proposed method in some of the tasks although the differences were small.

Notwithstanding, the proposed iterative training procedure opens novel alternatives for training neural networks. This particularly evident for regression tasks where the proposed method achieved the best result in most of the analyzed datasets.

REFERENCES

[1] I. Babajide Mustapha and F. Saeed, “Bioactive molecule prediction using extreme gradient boosting,” *Molecules*, vol. 21, no. 8, p. 983, Jul. 2016.

[2] A. Torres-Barrán, Á. Alonso, and J. R. Dormonosoro, “Regression tree ensembles for wind energy and solar radiation prediction,” *Neurocomputing*, vol. 326–327, pp. 151–160, Jan. 2019.

[3] N. Mirabal, E. Charles, E. C. Ferrara, P. L. Gonthier, A. K. Harding, M. A. Sánchez-Conde, and D. J. Thompson, “3GFCL demographics outside the galactic plane using supervised machine learning: Pulsar and dark matter subhalo interpretations,” *Astrophys. J.*, vol. 825, no. 1, p. 69, Jun. 2016.

[4] X. Jia, Y. Cao, D. O’Connor, J. Zhu, D. C. W. Tsang, B. Zou, and D. Hou, “Mapping soil pollution by using drone image recognition and machine learning at an arsenic-contaminated agricultural field,” *Environ. Pollut.*, vol. 270, Feb. 2021, Art. no. 116281.

[5] M. Fernández-Delgado, E. Ceradas, S. Barro, and D. Amorim, “Do we need hundreds of classifiers to solve real world classification problems?” *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 3133–3181, 2014.

[6] R. Caruana and A. Niculescu-Mizil, “An empirical comparison of supervised learning algorithms,” in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, 2006, pp. 161–168.

[7] H. Schwenk and Y. Bengio, “Boosting neural networks,” *Neural Comput.*, vol. 12, no. 8, pp. 1869–1887, 2000.

[8] M. Moghimi, S. Belongie, M. Saberian, J. Yang, N. Vasconcelos, and L.-J. Li, “Boosted convolutional neural networks,” in *Proc. Brit. Mach. Vis. Conf.*, York, U.K., 2016, p. 6. [Online]. Available: http://vision.cornell.edu/se3/wp-content/uploads/2016/08/boosted-convolutional-neutral-l.pdf

[9] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.

[10] J. Friedman, T. Hastie, and R. Tibshirani, “Additive logistic regression: A statistical view of boosting (with discussion and a rejoinder by the authors),” *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, Apr. 2000.

[11] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, New York, NY, USA, Aug. 2016, pp. 785–794.

[12] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[13] J. H. Friedman, “Stochastic gradient boosting,” *Comput. Statist. Data Anal.*, vol. 38, no. 4, pp. 367–378, 2002.

[14] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “CatBoost: Unbiased boosting with categorical features,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 6638–6648.

[15] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “LightGBM: A highly efficient gradient boosting decision tree,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3146–3154.
[43] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.

[44] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2009, pp. 248–255.

SEYEDSAMAN EMAMI received the B.Sc. degree in industrial engineering and the M.Sc. degree in financial engineering from Islamic Azad University, in 2012 and 2018, respectively. He is currently pursuing the Ph.D. degree in computer and telecommunications engineering with a specialization in information engineering with Universidad Autónoma de Madrid (UAM). He also gained valuable practical experience through collaborations with industry partners and participation in research projects. They have demonstrated strong problem-solving skills, attention to detail, and the ability to work collaboratively with diverse teams. Overall, he is a dedicated and accomplished researcher with a strong track record in machine learning, and their work promises to make significant contributions to the field. His research interests include machine learning topics, such as ensemble learning, Bayesian networks, deep learning frameworks, and the application of machine learning in the domains of science, media, and education.

GONZALO MARTÍNEZ-MUÑOZ received the university degree in physics and the Ph.D. degree in computer science from Universidad Autónoma de Madrid (UAM), Madrid, in 1995 and 2006, respectively. From 1996 to 2002, he worked in industry. Until 2008, he was an Interim Assistant Professor with the Department of Computer Science, UAM. From 2008 to 2009, he was a Fulbright Postdoctoral Researcher with the Group of Prof. Thomas G. Dietterich, Oregon State University, a world reference in the field of machine learning. He is currently an Assistant Professor with the Department of Computer Science, UAM, as well as a Coordinator of the Master’s Program on Bioinformatics and Computational Biology. In addition, he is the IP of several research projects related to AI. He has published his research in some of the journals with the highest impact on machine learning. His research interests include machine learning, decision trees, ensemble learning, and applications of machine learning to science, sports, and education.