Gradient Boosted Neural Decision Forest

Manqing Dong, *Member, IEEE,* Lina Yao, *Member, IEEE,* Xianzhi Wang, *Member, IEEE,* Boualem Benatallah, *Member, IEEE,* Shuai Zhang, *Member, IEEE,* and Quan Z. Sheng, *Member, IEEE*

Abstract—Tree-based models and deep neural networks are two schools of effective classification methods in machine learning. While tree-based models are robust irrespective of data domain, deep neural networks have advantages in handling high-dimensional data. Adding a differentiable neural decision forest to the neural network can generally help exploit the benefits of both models. Therefore, traditional decision trees diverge into a bagging version (i.e., random forest) and a boosting version (i.e., gradient boost decision tree). In this work, we aim to harness the advantages of both bagging and boosting by applying gradient boost to a neural decision forest. We propose a gradient boost that can learn the residual using neural decision forest, considering the residual as a part for the final prediction. Besides, we design a structure for learning the parameters of neural decision forest and gradient boost module in contiguous steps, which is extendable to incorporate multiple gradient-boosting modules in an end-to-end manner. Our extensive experiments on several public datasets demonstrate the competitive performance and efficiency of our model against a series of baseline methods in solving various machine learning tasks.

Index Terms—Neural Decision Forest, Gradient Boost, Neural Network.

1 Introduction

MACHINE learning techniques have shown great power and efficacy in dealing with various tasks in the past decade [1]. Among them, tree-based models [2] and deep neural networks [3] are two promising classes of machine learning methods that are proven successful in many applications. Random forest [2] and Gradient Boost Decision Tree [4] are two representative models for **tree-based models**. *Random forest* is the ensemble of tree predictors [2]; it takes the mean prediction of the individual decision trees as the prediction result, thereby alleviating over-fitting while preserving the explanation ability of a single decision tree and the robustness on data domain [5]. *Gradient Boost Decision Tree* (GBDT) [4] is a boosting version of decision tree. The idea of gradient boost is to build a new tree towards the residuals of the previous prediction result for each step. While GBDT uses a decision tree as the base learner, the idea of gradient boosting is proven to improve the prediction performance on several traditional machine learning techniques such as linear regression and support vector machine [4]. In comparison, **deep neural networks** have just begun to revolutionize domain applications, such as speech recognition [6] and computer vision tasks [7]. Neural networks generally have more parameters and have advantages in capturing complex information [8]. Supported by increasing computing power, deep neural networks are becoming bigger, deeper, and more complicated. For example, AlexNet, the champion of ImageNet Large Scale Visual Recognition Challenge (ILSVRC), had nine layers in the neural network model in 2012 [9]. This number had increased to 152 in the best model, ResNet, in 2015's contest [7].

Despite their advantages, the two categories of models have the following limitations. While neural networks show stengths in dealing with large-scale data, it poses the risk of over-fitting on smaller datasets [10]. Deep neural networks, in comparison, face the black-box problem [3], meaning those models generally lack explainability of their results. Likewise, tree-based methods are not suitable for all occasions. For example, the performance of tree-based models highly rely on the quality of the features, due to the greedy tree construction process [5]. Tree-based models generally show good and fast prediction on small datasets with fewer features but fall insufficient in dealing with more complex features concerning images, speeches, and texts [10].

To address the above challenges, some recent studies aim to combine the strength of both models [11], [12], [13]. One direction is ***integrating random forest with neural networks***. Some work leverages the '**deep**' idea from deep neural networks to improve tree-based methods' ability in dealing with complex inputs while keeping the greedy-splitting criterion of decision trees. For example, Zhou et al. [10] propose a deep forest that repeatedly feeds features into random forests and use the outputs of random forests for the further random forests. Similarly, Feng et al. [14] construct the random forests that follow the structure of autoencoders [3]. Some work leverages the '**end-to-end**' training strategy and propose neural decision trees [12] so that the tree-based methods can be optimized by the optimization function for different purposes. Such a strategy improves the flexibility of the tree-based models in dealing with a wide range of tasks. Johannes et al. [12] point out that any decision tree can be represented as a two-layer Convolutional Neural Network (CNN), where the first layer includes the decision nodes with denoting the probability of delivering the inputs to the left branch or right branch, and the second comprises leaf nodes. The prediction is the average of votes

- *M. Dong, L. Yao, and B. Benatallah are with University of New South Wales.*
- *X. Wang is with University of Technology Sydney.*
- *S. Zhang is with ETH Zürich.*
- *Quan Z. Sheng is with Macquarie University.*

from the whole tree ensemble. Kontschieder et al. [15] introduce statistics into the neural network to construct a neural decision forest. In particular, they represent decision nodes as probability distributions to make the whole decision differentiable. Another direction is towards ***integrating boosting idea into the neural networks***. An example is to use neural networks as the base classifier for an AdaBoost framework [16]. Besides, Zhang et al. [17] use CNNs as weak learners in gradient boosting for image classification.

In this work, we propose a general framework Gradient Boosted Neural Decision Forest (GrNDF) that is flexible, stable, and applicable to a wide range of tasks, by integrating the advantages of both tree-based methods and neural networks. GrNDF can effectively address the challenges raised by traditional tree-based methods and neural networks by improving the flexibility and explainabilty. Specifically, GrNDF is flexible to handle different type or size of features by mapping the inputs to embedding layers; it is also made explainable by delivering the inputs via probabilistic neural decision trees, where the split nodes show the probability of the inputs transmitted to the leaf nodes, and the leaf nodes show different probabilistic distributions for prediction. To the best of our knowledge, our proposed framework is the first work that integrates the ideas of boosting and bagging under the neural network structure. The model can be trained in an end-to-end manner, suitable for different optimization tasks.

In a nutshell, we make the following contributions:

- We bring the gradient boost idea into neural decision forest to take the advantages of both bagging and boosting methods and to provide robust predictions for different tasks. We unify the different methods elegantly in an end-to-end trainable way.
- We describe how to construct a gradient boost module for the neural decision forest and further make the gradient boost module extendable to enhance model robustness and performance.
- We have conducted extensive experiments under different parameter settings and ablation studies to evaluate the model's performance on several public datasets. The proposed model can achieve superior prediction performance when compared with a series of baselines and state-of-the-art.

The rest of the paper is organized as follows: Section 2 reviews the related work; Section 3 introduces the technical details of the proposed method; Section 4 reports the experimental results under different parameter settings, as well as the ablation studies on different datasets; finally, Section 5 offers the concluding remarks.

2 Related Work

Tree-based methods [4], [2] and neural networks [3] are two important scopes of machine learning techniques and have been applied to various tasks [18], [19], [20].

2.1 Tree-based Methods

Among tree-based methods, **random forest** [2] and **gradient boosting decision tree** (GBDT) [4] are two popular models that have been successfully applied to real-world problems [18], which can be regarded as bagging and boosting version of decision tree, respectively.

The commonly-used **random forest** combines several randomized decision trees and aggregates their predictions by averaging [21]. Most random forests use bagging [2] and CART (Classification And Regression Trees) [22] for aggregating trees. Bagging is a general aggregation method which constructs a predictor from each sample and then takes the average prediction for the decision. It is regarded as one of the most effective computationally intensive procedures to improve on unstable estimates [5].

Gradient boost decision tree (GBDT) embraces the idea of gradient boost, which originated from the observation by Breiman [2] that boosting can be interpreted as an optimization algorithm on a suitable cost function. Based on this idea, Friedman [4] developed a gradient boosting machine (GBM) and consistently provided more accurate results than the conventional single machine learning models from which it originated. Friedman gave examples for adding the gradient boost idea into traditional machine learning models (e.g., logistic regression) and proposed gradient boost decision tree (GBDT). Over the last few years, many empirical studies revealed the power of gradient boost machine in various areas. Most of them focused on taking decision trees as base learners [23].

2.2 Neural Networks

The concept of deep learning originated from the study of artificial neural networks (ANNs) [24]. Milestones of deep learning can be traced from 2006 when Hinton proposed a novel deep structured learning architecture called deep belief network (DBN) [25]. Normally, we call a neural network with three or more layers a deep neural network (DNN). Supported by the increasing computing power, neural networks have become deeper and more complex, multiple variants of simple neural layers are proposed for different purpose, e.g., convolutional neural networks (CNNs), recurrent neural networks (RNNs), autoencoders [3], and graph neural networks (GNNs) [26]. The increasing number of parameters enables neural networks to have more advantages in capturing complex information. They have made significant advances in dealing with image recognition [27], activity recognition [28], document classification [29], etc.

2.3 Hybrid Methods

Despite their advantages, tree-based methods and deep neural networks have several limitations. Deep neural networks face challenges in explainability and complexity. A complicated network structure with a huge number of parameters can improve the model's performance while costing more time in searching for the optimal solution, meaning the performance depends heavily on the parameter tuning [10]. Besides, the black-box training process makes it unclear about how each component in the neural network contributes to the final prediction. For tree-based methods, a key issue is that the model's performance is relevant to the quality of the inputs. It is proven that random forest is prone to over-fitting when handling noisy data [5]. Besides, the tree-based models become time-costly in handling high-dimensional data sets. For the above reasons, some research seeks to bridge the two models for leveraging the benefits of both.

One direction is integrating random forest with neural networks. **1)** Some work leverages the '**deep**' idea that they construct cascade random forests with following the structure of neural networks while keeping the core concepts of the random forest, i.e., using greedy splitting criteria of the decision trees. For example, Zhou et al. [10] proposed deep forest that repeatedly fed the features into random forests and used the outputs of random forests as the inputs for the subsequent random forests. The final prediction is taking the average of the last layer of random forests. Similarly, Feng et al. [**?**] constructed random forests that follow the structure of autoencoders. Those structures optimize the split functions one node at a time according to some splitting criteria, where the greedy procedure may lead to sub-optimal trees. **2)** An alternative way is leveraging the '**end-to-end**' training strategy of neural networks, i.e., learning the random forests with back-propagation. Such way retains the principles of neural networks but requires transforming the random forests into differentiable embedding layers before adding it to neural networks. Sethi [30] made the first attempt in the 1990s by proposing entropy net, which encoded decision trees into neural networks. Welbl et al. [12] further suggested that any decision trees can be represented as a two-layer Convolutional Neural Network (CNN) [9], where the first layer includes the decision nodes and the second layer includes leaf nodes.

Some other researchers tried to design differentiable decision trees from a statistical perspective. Montillo [31] did some early work in this direction by investigating the use of sigmoid functions in differentiable information gain maximization. Peter et al. [15] introduced a stochastic, differentiable, and back-propagation compatible version of decision trees to guide representation learning in the lower layers of deep convolutional networks. Specifically, they regarded decision nodes in the neural tree as probabilistic distributions of passing down the parent nodes and leaf nodes as the probabilistic distributions over predictions. They then implemented their model over several benchmark image recognition datasets and beat a series of other classical deep neural networks such as CNN. In this work, we follow the idea of [15] in constructing the neural decision forest. Adding statistical methods to neural networks makes the networks more explainable, while randomly initialized leaf nodes have the limitation of yielding biased prediction when the forest comprises a small number of trees and leaf nodes.

There is also some work that integrates the idea of gradient boosting with neural networks [17]. The first few attempts include Schwenk's work [16], where they investigate different techniques of using neural networks as base classifiers for the AdaBoost framework. Similarly, Zhang et al. [17] proposed to incorporate gradient boost machine with convolutional neural networks (CNN), where they use CNN as weak learners for dealing with image classification tasks. Feng et al. [32] propose a general differentiable gradient boosting framework for taking neural networks or traditional classifiers as weak learners.

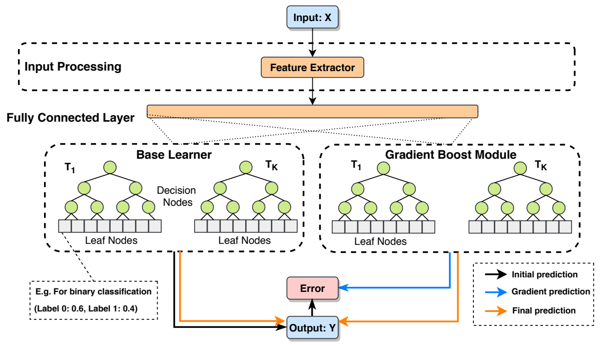


Fig. 1. The proposed gradient boosted neural decision forest.

Our proposed framework is designed to provide a more explainable and flexible solution that can also be applied to a wide range of tasks to address the challenges faced by tree-based models and deep neural networks alone. It differs from the previous research by being the first work to combine the advantages of bagging and boosting ideas under the neural network architecture. On the one hand, the nature of ensemble learning can alleviate the over-fitting phenomenon of the deep learning models, and the neural decision forest can also improve the explainability of the traditional neural layers (see Section 4.5.3). On the other hand, deep learning methods need less efforts in feature engineering, thus bring more flexibility and capture more interaction knowledge among the features than the traditional tree-based methods.

3 Gradient Boosted Neural Decision Forest

Figure 1 shows the overall workflow of the proposed method with one gradient boost module. The workflow includes three main components from top to bottom: *the input processing layer*, *the base learner (the neural decision forest)*, and *the gradient boost module*. Suppose we have input-output pairs $\{(x_n, y_n)|n = 1, 2, ..., N\}$, our target is training on those pairs and providing prediction for new inputs x_{new}. Specifically, we first train the base learner to get an initial prediction: the input will go through an input processing layer to get a latent representation H, followed by fully-connected layers in mapping the latent representation to nodes in K neural decision trees. The prediction is taking the average of each neural decision trees. Second, a gradient boost module, i.e., a new neural decision forest, will be constructed for predicting the gradient of the loss for the initial prediction. More gradient boost modules can be constructed for targeting the gradient of the loss from the previous prediction. The final prediction will be the sum of the results from the base learner and the following gradient boosting modules. The input processing layer is shared across weak learners. The module iteratively optimizes the whole process until reaching the optimal results.

In the following subsections, we introduce each component in detail.

3.1 Input Processing Layer

Usually, we can select different input processing layers for different purposes. Here, we introduce convolutional

TABLE 1
Notations in this work

Notation	Meaning
(x, y)	input and output pairs
N	number of samples
M	number of classes
K	number of trees in a forest
$\mathcal{F}$	a forest $\mathcal{F} = \{T_1, ..., T_K\}$
I	number of gradient boost modules
W_*	weight matrices
b_*	biases
H	latent representation of x
f_*	functions
σ	activation function
$D\&L$	decision & leaf nodes
$\mathbb{P}$	probability distribution

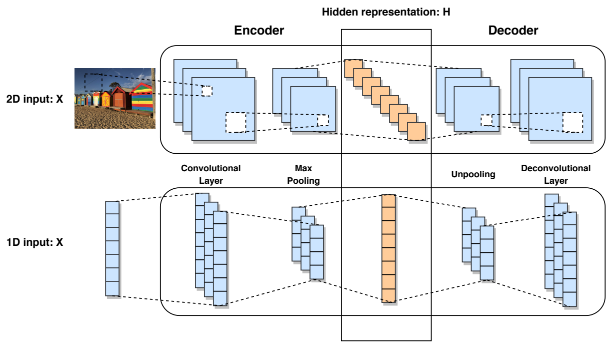


Fig. 2. An example of using convolutional autoencoder as input processor, which can deal with both 1D and 2D inputs.

autoencoder (CAE), which can deal with both 1D and 2D inputs, as our input processing module. An autoencoder (AE) is an unsupervised component for producing dense feature representations from the input [33], i.e., dimension deduction. It consists of an encoder and a decoder. The encoder takes input x to several neural layers for getting the latent representation H with

$$H = f_E(W_E x + b_E) \tag{1}$$

where $f_E(*)$ is a non-linearity function (e.g., the sigmoid function [34]) and W_E and b_E are the corresponding weight and bias, respectively. The encoder tries to map the latent representation H to the reconstruction of input x, which can be denoted by x_c, through several neural layers:

$$x_c = f_D(W_D H + b_D) \tag{2}$$

where x_c can be regarded as a prediction of x, given the latent code H. Similarly, W_D and b_D are the weights and bias variables for the decoder layers.

There are different ways to construct the encoder and decoder layers, e.g., fully connected layers in the simplest case. Here, we introduce convolutional autoencoder as the input processor, which uses convolutional layers in the encoder and decoder layers [35]. Convolutional layers differ from fully connected layers in considering local information and being less time-consuming [3]. It can also deal with high-dimensional inputs such as images, thus exhibiting high flexibility and efficiency. Generally, the autoencoder models are optimized in an unsupervised manner, where the loss function is defined to minimize the average reconstruction error, e.g., in terms of the squared error: $\mathcal{L}(x, x_c) = \parallel x - x_c \parallel^2$. In this work, we update the parameters of CA considering both reconstruction error and the task prediction error, which can improve the robustness of the latent representation H and the prediction performance by using the hidden representation H. We will provide details of our optimization strategy later.

We use the transformation of latent representation H as the input for the further neural decision forests. Specifically, the latent representation H will first go through several fully-connected layers to get x_{FC} as the input for the neural decision forest, i.e., $x_{FC} = f_{FC}(W_{FC} X_H + b_{FC})$.

3.2 Neural Decision Forest

In this subsection, we introduce how the input nodes x_{FC} are transferred through the tree layers in the neural decision forest, which shares similar spirits of the structure suggested by [15].

Suppose we have a number of K trees in a forest, each being a structured classifier consisting of decision (or split) nodes D and leaf (or prediction) nodes L. The leaf nodes are the terminal nodes of the tree and each leaf node $l \in L$ holds a probability distribution $P_l(Y|x)$ over the class distribution Y. Each decision node $d \in D$ in tree T_k holds a decision function $\mathcal{D}_d(x; \Theta) \in [0, 1]$, which stands for the probability that a sample reaches decision node d and is sent to the left sub-tree. More specifically, each decision node holds the following representation:

$$\mathcal{D}_d(x; \Theta) = \sigma(f_d(x_{FC})) \tag{3}$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function; $f_d(x_{FC}) = W_T x_{FC}$ is the transfer function for x_{FC} (W_T is weight variable); and Θ stands for the set of previous parameters used in autoencoder and fully-connected layers, i.e., $\{W_E, W_D, W_{FC}, W_T, b_E, b_D, b_{FC}\}$.

In this paper, we suppose all the trees follow a classical binary tree structure, meaning each node has two subtrees. Therefore, once the depth n_depth is defined, we can get 2^{n_depth} decision nodes and $2^{n\_depth+1}$ leaf nodes. For example, after setting the depth n_depth of a tree to 2 (as shown in Figure 1), the tree has two decision nodes and four leaf nodes.

The probability of a sample arriving at tree T_k and reaching in leaf l is as follows:

$$Q_l = \Pi_{d \in D} \mathcal{D}_d(x; \Theta)^{\mathbb{1}_{left}} \overline{\mathcal{D}_d}(x; \Theta)^{\mathbb{1}_{right}}, \tag{4}$$

where $\overline{D_d}(h; \Theta) = 1 - D_d(h; \Theta)$, $\mathbb{1}_{left}$ ($\mathbb{1}_{right}$) stands for the indicator function for the nodes going left (right). Meanwhile, the probability of this sample to be predicted as class y is:

$$\mathbb{P}_k[y|x, \Theta, \Lambda] = \Sigma_{l \in L} P_{l_y} \cdot Q_l, \tag{5}$$

where P_{l_y} stands for the probability of the nodes in leaf l predicted to be label y according to the previous defined probability distribution $P_l(Y|x)$; and Θ and Λ are parameters for input processing layers and neural decision forest, respectively.

For the forest of neural decision trees, it is an ensemble of decision trees $\mathcal{F} = \{T_1, ..., T_K\}$ and delivers a prediction for sample x by averaging the output of each tree, which can be shown from:

$$\mathbb{P}_{\mathcal{F}}[y|x] = \frac{1}{K}\Sigma_{k=1}^{K}\mathbb{P}_k[y|x] \tag{6}$$

The prediction for a label, e.g., y, is as follows:

$$\hat{y} = \underset{y}{\operatorname{argmax}}\,\mathbb{P}_{\mathcal{F}}[y|x] \tag{7}$$

3.3 Gradient Boost Module

Here, we introduce the basic concepts of the gradient boost idea and our design in integrating gradient boost with neural decision forest. Figure 1 gives an example for only one gradient boost module, where the gradient boost module shares same structure as base classifier, i.e., neural decision forest in this case.

3.3.1 Traditional gradient boost algorithms

The principle of boosting methods is to hold out optimization in the function space [4]. The key definitions in gradient boosting include *weak learner*, *base learner*, and *gradient boosting modules (boosts)*. Generally, for a given problem, the estimated function $\hat{f}(x)$ is in an additive way:

$$\hat{f}(x) = \hat{f^I}(x) = \sum_{i=0}^{I}\hat{f}_i(x) \tag{8}$$

where $\hat{f}_0$ is the initial prediction made by the *base learner*, $\{\hat{f}_i\}_{i=1}^{I}$ are the function increments (also called as 'boosts'), I is the number of *gradient boost modules* (iterations), and $\hat{f^i}$ is the ensemble prediction in iteration i. The *weak learner* is the option for the predictor of the *base learner* and the *gradient boost modules*. For example, GBDT is taking decision tree as the *weak learner*.

Under the idea of gradient boosting, each weak learner $h(x,\theta)$ in this incremental functions are formulated in a 'greedy stage-wise' way. In particular, the model sets $\hat{f}_0$ with $h(x,\theta_0)$ and then chooses a new function $h(x,\theta_i)$ to be the most parallel to the negative gradient $g_i(x)$ along the observed data.

$$g_i(x) = E_y[\frac{\partial\mathcal{L}(y,f(x))}{\partial f(x)}|x]_{f(x)=\hat{f^{i-1}}(x)} \tag{9}$$

where $\mathcal{L}$ is the loss function. By setting each step with step size ρ, we choose the new function increment to be the most correlated with $-g_i(x)$. Suppose we have N samples, we can formulate the optimization by least-squares minimization:

$$(\rho_i,\theta_i) = \underset{\rho,\theta}{\operatorname{argmin}}\sum_{j=1}^{N}[-g_i(x_j)+\rho h(x_j,\theta)]^2 \tag{10}$$

3.3.2 Proposed gradient boost module

In our model, the weak learner $h(x,\theta)$ is a neural decision forest, meaning that we construct new neural decision forest for each gradient boost step but share the input processing layers across the weak learners (i.e., the base learner and the gradient boost modules). After learning the initial model $f^0 = h(x,\theta_0)$, we get the the prediction $\hat{f}^0 = \hat{y}$ according to equation 7. Suppose we have N samples and M classes, the loss is defined as:

$$\mathcal{L}(y,f^0) = -\Sigma_{m=1}^{M}y_m\log p_m(x), \tag{11}$$

where $y_m \in \{0,1\}$, and $p_m(x) = \mathbb{P}(y_m = 1|x)$ or equivalently, $p_k(x) = exp(\mathbb{P}(y=y_m))/\Sigma_{m=1}^{M}\mathbb{P}(y=y_m))$.

Then, for the first gradient boost module, the prediction target is:

$$\begin{aligned} G_1 = &-[\frac{\partial L(y,f^0)}{\partial f^0}]_{f^0\mathbb{P}_{\mathcal{F}}[y|x]} \\ &\sim Y - p(x) = Y - softmax(f^0) \end{aligned} \tag{12}$$

where G_1 is the 'gradient boost'. In practice, we modify this gradient as $Error(1) = softmax(Y - f^0)$. As an example, consider a binary classification problem where $M = 2$ and the probability distribution estimation for sample x is $(0.4, 0.6)$. The sample will be predicted to be class y_2. Suppose x actually belongs to class y_1, and we hope it to take the distribution $(1,0)$ and to increase the probability that x is predicted to class y_1. Matching the 'boosts' $h(x,\theta_1)$ with $(0.6,-0.6)$ to achieve the ideal distribution is unreasonable since the range of $\mathbb{P}_{\mathcal{F}}[y|x]$ is $[0,1]$. Therefore, we fit the 'boosts' into a softmax version: $(\frac{e^{0.6}}{e^{0.6}+e^{-0.6}}, \frac{e^{-0.6}}{e^{0.6}+e^{-0.6}})$. More specifically, for the first 'boost' module, we take the next few steps:

$$\begin{cases} Error(1) =& softmax(Y - \mathbb{P}_0(Y|x)) = \mathbb{P}_{GB_1}(Y|x) \\ f^1 =& f^0 + \rho_1\cdot Error(1) \\ =& (\mathbb{P}_0(y_1)+\rho_1\cdot\mathbb{P}_{GB_1}(y_1),\ldots,\mathbb{P}_0(y_M) \\ & +\rho_1\cdot\mathbb{P}_{GB_1}(y_M)) \\ =& (\mathbb{P}_1(y_1),\ldots,\mathbb{P}_1(y_M)) \\ \overset{ScaleTo1}{=} & (\frac{\mathbb{P}_1(y_1)}{\sum_{m=1}^{M}\mathbb{P}_1(y_m)},\ldots,\frac{\mathbb{P}_1(y_M)}{\sum_{m=1}^{M}\mathbb{P}_1(y_m)}) \\ \hat{f}^1 =& \underset{y}{\operatorname{argmax}}\, f^1 \end{cases} \tag{13}$$

where $Error(1)$ is the prediction goal for the first gradient boost module, which is the distance between the prediction and the label. We update the prediction f^1 by adding this distance with a step size hyper-parameter ρ to the previous guess of prediction f^0 and scale the sum of each probability nodes in f^0 to 1.

We take similar approaches for the following gradient boost modules. Suppose we have I gradient boost modules, then for i-th module, the prediction target is $Error(i)$. Then after I steps of gradient boosting, we get the final prediction $\hat{y}$ by:

$$\begin{cases} Error(I) =& softmax(Y - f^{I-1}) = \mathbb{P}_{GB_I}(Y|x) \\ f^I =& f^{I-1} + \rho_I\cdot Error(I) \\ =& (\mathbb{P}_{I-1}(y_1)+\rho_I\cdot\mathbb{P}_{GB_I}(y_1),\ldots,\mathbb{P}_I(y_M)) \\ & +\rho_M\cdot\mathbb{P}_{GB_M}(y_K)) \\ =& (\mathbb{P}_M(y_1),\ldots,\mathbb{P}_M(y_K)) \\ \overset{ScaleTo1}{=} & (\frac{\mathbb{P}_M(y_1)}{\sum_{i=1}^{K}\mathbb{P}_M(y_i)},\ldots,\frac{\mathbb{P}_M(y_K)}{\sum_{i=1}^{K}\mathbb{P}_M(y_i)}) \\ \hat{y} =& \underset{y}{\operatorname{argmax}}\, f^M \end{cases} \tag{14}$$

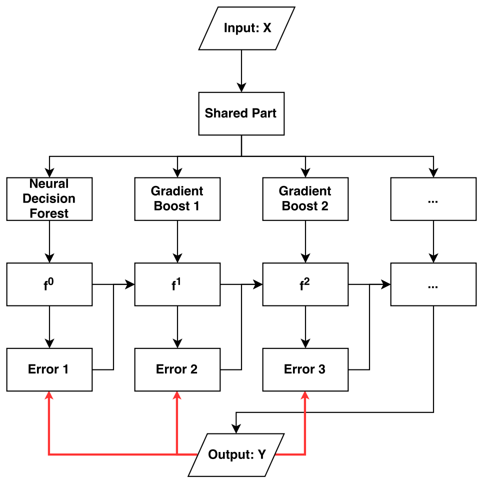


Fig. 3. **Flowchart for training process with multiple gradient boost modules**. The shared part is our input processing module, which in this case contains autoencoder and fully-connected layers. First, the whole framework will be initialized with random variables, and the input x will go through the black lines. The first neural decision forest is the base learner, which processes the data flow and makes predictions. The label Y will feed prediction loss back to update the parameters in the base learner and the input processing layers, where the prediction error, i.e., Error 1, will be taken as the prediction target for the first gradient boost module. Similar approaches will be taken for the following gradient boost modules. The final prediction is the summation of the outputs from the weak learners.

3.4 Training Process

The prediction of a GrNDF, i.e., $\hat{y}$, is given in equation 14. We use cross-entropy [3] as the loss function for classification tasks:

$$\mathcal{L}(y,\hat{y}) = -(y \times log(\mathbb{P}[y|x,\Theta,\Lambda]) + (1-y)log(1-\mathbb{P}[y|x,\Theta,\Lambda])) \tag{15}$$

where Θ stands for the parameters used in the input processing layers, and Λ stands for the parameters used in neural decision forest.

In this case, the input processing layer is convolutional autoencoder, the reconstruction loss for an autoencoder is:

$$\mathcal{L}_A(x, x_c) = \parallel x - x_c \parallel^2 \tag{16}$$

Then, the total loss for the model is:

$$\mathcal{L} = \mathcal{L}(y,\hat{y}) + \mathcal{L}_A(x, x_c) \tag{17}$$

which is equivalent to solving the optimal parameters:

$$(\Theta, \Lambda) = \underset{\Theta,\Lambda}{\operatorname{argmin}}\, \mathcal{L} \tag{18}$$

Notice that the middle layer of convolutional autoencoder will be further fed into several fully connected layers to be the input for each neural decision forest, the parameters of the autoencoder and the neural decision forest will update simultaneously. In this way, the latent representation of x will be not only representative for the input but also effective for the final prediction. We use RMSProp optimization methods [36] and update the parameters one by one. In particular, the learning rate will be divided by an exponentially decaying average of squared gradients.

At the very first stage, the parameters will be given randomly initialized values. Then, we can get the initial prediction f^0 and the according loss $\mathcal{L}$. To update the parameters Θ and Λ, we first set Λ unchanged and update Θ by back-propagation following the updating strategy below at step t:

$$g_t^\Theta = \frac{\partial \mathcal{L}}{\partial \Theta} \tag{19}$$

$$G_t^\Theta = G_t^\Theta + g_t^\Theta \odot g_t^\Theta \tag{20}$$

$$\Theta_t = \Theta_{t-1} - \frac{\eta^\Theta}{\sqrt{G_t^\Theta + \epsilon}} \odot g_t^\Theta \tag{21}$$

where g_t^Θ is the gradient of parameter Θ; η^Θ and ϵ are the corresponding learning rates; and the symbol $\odot$ stands for dot product. Then, we take a similar updating strategy for updating the parameters Λ in neural decision forest. There are two groups of parameters in neural decision forest: parameters for decision nodes, and the probability distribution of the leaf nodes. For the probability distribution of the leaf nodes, we add a softmax function to the updated parameters. The details about how to calculate the derivative of loss function for each parameter is included in paper [15].

For the following gradient boost modules, we change the loss function to equation 10 and update the parameters Λ for each gradient boost module in a similar way as in updating the base learner f^0. Notice that we only update the parameters Λ for each gradient boost module, which means the parameters Θ are shared across the weak learners. This way, the input processing layers will try to extract the most useful latent feature patterns for the prediction.

The hyper-parameter ρ, i.e., the learning rate, can be regarded as either a constant (e.g., between 0 and 1) or a parameter that can be updated during the training process. Here, we set this learning rate as a constant, and we will discuss the model's performance under different settings to this learning rate.

The training process for the model with multiple gradient boost modules is shown in Figure 3, where the black lines indicate the data flow and the red lines stand for the back-propagation process. The following shows the pseudo-code for the training and testing process of the proposed framework.

Algorithm 1 Training for GrNDF

```
Require: Input $x$, labels $y$
 1: Generalizing the structure for the forest with $K$ trees each with depth $n\_depth$
 2: Randomly initialize parameters $\Theta, \Lambda$
 3: Randomly shuffle the datasets
 4: for epoch in Epochs do
 5:     for batch in Batches do
 6:         Get the outputs of the input processing layer
 7:         Get the prediction of the base learner
 8:         Update $\Theta$ and $\Lambda$
 9:         for i in I do
10:             Calculate Error(i) for i-th boost module
11:             Update $\Theta$ and $\Lambda_i$
12:         end for
13:     end for
14: end for
```

Algorithm 2 Testing for GrNDF

```
Require: Input $x_{new}$, Learned $\Theta$, Learned $\Lambda$
Output: Prediction of label $\hat{y}$
 1: Initialize GrNDF with the learned parameters $\Theta$, $\Lambda$
 2: Get hidden representation $H$ of $x_{new}$ by the encoder:
 3: $H = f_E(W_E x_{new} + b_E)$
 4: Get input for decision trees:
 5: $x_{FC} = f_{FC}(W_{FC} H + b_{FC})$
 6: Get initial prediction $\hat{f}_0(x_{new})$ by eq. 7
 7: for i in I do
 8:     Get prediction $\hat{f}_i(x_{new})$by eq. 13 and eq. 14
 9: end for
10: Get the final prediction by:
11: $\hat{y} = \hat{f}(x) = \hat{f}^I(x) = \sum_{i=0}^{I} \hat{f}_i(x)$
12: return y
```

3.5 Computational Complexity

The proposed framework contains three parts: the input processing layers, the neural decision forest, and the gradient boost modules. The computational complexity of input processing layers varies according to different network options. In this case, the computational complexity for the convolutional autoencoder is around $\mathcal{O}(k^2 n)$, where k here is the kernel size, and n here is the number of neurons in a layer. Notice that the computational complexity of a fully connected layer is about $\mathcal{O}(n^2)$ and $k << n$, using convolutional layers will take less computing time than fully connected layers. The computational complexity of the neural decision forest is $\mathcal{O}(2^d nt)$, where d is the depth of a tree, 2^d is the number of decision nodes in the tree, n is the number of neurons in the output of the input processing layer, and t is the number of trees. The neural decision forest takes less computation time than fully connected layers with shallow (small d) and few (small t) trees. As for m gradient boost modules, the computational complexity is $(m+1) \times \mathcal{O}(2^d nt)$. In our default settings, there are $t = 50$ trees in a forest each with depth $d = 3$, and there is only $m = 1$ gradient boost module, where the computational complexity is same as of a fully connected layer with hundreds of neurons. According to our experiments, we can achieve outstanding performance with the default settings. Overall, the proposed framework takes no more computing time than traditional neural networks in most scenarios.

4 Experiments

In this section, we explore the efficiency of the proposed framework on a wide variety of tasks and evaluate the flexibility of our framework on several datasets. our experimental results show the effectiveness of our proposed framework, which outperforms a series of baselines. We also evaluate the performance of the proposed framework under different parameter settings.

4.1 Dataset description

We test our model over different scopes of datasets, including a seizure recognition dataset, an activity recognition dataset, a fake review dataset, and a public image recognition dataset (the last row in Table 2).

Fig. 4. Example for fashion MNIST. Each class is represented by nine cases.

Epileptic Seizure Recognition Dataset [37] is a pre-processed and re-structured version of a commonly used dataset for epileptic seizure detection. The original dataset has 500 individuals with each has 4097 data points for 23.5 seconds. Each data point contains the value of the EEG recording at a different point in time. The reshaped one shuffled every 4097 data points into 23 chunks. Each chunk contains 178 data points for 1 second. Thus there are 11,500 pieces of information (row) in total. Each piece of information includes 178 data points for 1 second (column). Each instance will be given one of the following labels: "eyes open", "eyes closed", "they identify where the region of the tumor was in the brain and recording the EEG activity from the healthy brain area", "they recorded the EEG from the area where the tumor was located", or "recording of seizure activity".

Amazon review dataset is a benchmark dataset for fake review detection. The raw dataset contains over 100 million reviews covering 24 product categories. Each review contains the textual information, user's ratings, received helpful votes, and review time. Here, we used a subset with 7951 samples and obtained the labels from [38].

MHEALTH dataset [39] comprises recordings of the body motions and vital signs of 10 volunteers performing 12 physical activities (e.g., standing, walking, and running). Each subject carried three sensors on the chest, right wrist and left ankle. There are 23 channels for recording and about 3,000 data instances for each activity (except for the 12th activity), where each instance corresponds to a data point recorded at a sampling rate of 50 Hz (i.e., 50 samples per second). We slice the original data with half-seconds and get 13635 instances, each corresponding to 25 time points times 23 recording channels.

Fashion-MNIST [40] is a dataset of Zalando's article images. This dataset consists of a training set of 60,000 samples and a test set of 10,000 samples. Each sample is associated with a label from 10 classes: t-shirt/top(label 0), trouser(1), pullover(2), dress(3), coat(4), sandal(5), shirt(6), sneaker(7), bag(8), ankle boot(9). Each image is 28 pixels in height and 28 pixels in width. Each pixel has an integer pixel-value between 0 and 255, indicating the lightness or darkness of that pixel. An example of the images is shown in Figure 4.

4.2 Settings

4.2.1 *Parameter Settings*

For the Epileptic Seizure Recognition dataset, we randomly separate the dataset into 8,000 training samples and 3,500 testing samples and then transform the label into the one-hot format. For the MHEALTH and the Amazon review

TABLE 2
Dataset Description

Dataset Name	#Classes	#Instances	#Attributes	Attribute Type
Epileptic Seizure Recognition dataset	5	11,500	178	Integer, Real
Amazon Review	2	7,951	48	Integer, Real
MHEALTH dataset	12	13,635	25×23	Real
Fashion MNIST	10	70,000	784 (28×28)	Integer(Pixel)

dataset, we randomly separate the datasets into 80% training data and 20% test data. For Fashion MNIST, which contains 60,000 training samples and 10,000 test samples, we transform the label into the one-hot format as well. Each image is in the shape of 28 pixels high and 28 pixels wide. We tile each image into a vector of 784 nodes if not using the convolutional layers. For the input processing layer, we use two convolutional layers in the autoencoder, followed by one fully-connected layer. For the neural decision forest, we set the depth of the tree as 3 and add only one gradient boost module on the base learner. The batch size is around 10% of the whole dataset. The number of neurons in a layer varies according to the dimension of the feature vectors, which will not greater than four times the number of features. All weights and bias are initialized with randomly distributed values. Each training model is iterated 200 times.

The model's performance under different parameter settings can be found in Section 4.4.

4.2.2 Comparison Methods

We compare with three baseline methods and four state-of-the-arts for evaluating the efficiency of the proposed framework. We first compare our model with three related baseline methods, namely gradient boost decision trees (GBDT), random forest, and convolutional neural networks (CNN):

- **Gradient Boost Decision Trees (GBDT)** [4]: is the boost version of decision trees. We use GBDT that follows Friedman's design for the comparison. Specifically, we set the number of estimators (decision trees) as 100 and the max depth of a tree as 10.
- **Random Forest** [21]: is known as the bagging version of decision trees. Here, we set the number of estimators (trees) as 100, the max depth as 100 and the criterion as entropy.
- **Convolutional Neural Networks (CNN)** [3]: has been widely used in various tasks these years. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field. Here, the model includes two convolutional neural with max-pooling followed by two fully-connected layers. We control the number of nodes of each layer according to the dimension of the input space.

Also, we compare our work with several state-of-the-arts: neural decision forest, gradient boost convolutional neural networks, autoencoder based method, and a model that uses both convolutional neural networks and convolutional autoencoders:

- **Peter et al. [15]**: proposed deep neural decision forests similar to our model. The original work combined deep neural decision forests with convolutional neural networks to deal with image recognition tasks. In our experiment, we also follow the original design and set around 50 trees in a forest.
- **Zhang et al. [17]**: proposed gradient boost convolutional neural networks, which used the convolutional neural network as the base learner. The base learner and the 'boosts' are two convolutional layers followed by a fully-connected layer.
- **Li et al. [?]**: proposed an autoencoder and a special prototype layer, where each unit of that layer stores a weight vector that resembles an encoded training input followed by a fully-connected layer.
- **Dennis et al. [41]**: designed a two-channel convolutional neural network, where one channel uses traditional CNN filters, and another one uses convolutional autoencoders as filters. They further concatenated the outputs from the two channels and fed them into several fully-connected layers.
- **Zhou et al. [10]**: constructed cascade random forests that follow the structure like deep neural networks. They firstly fed the features into random forests and then used the outputs of random forests for the further random forests. The above was repeated for several steps.

To compare with these methods, we either use the original codes or rewrite them following the original structure. Some models are sensitive to the normalization methods of inputs. Therefore, we feed the same inputs that are not normalized to all models (including ours). Besides, we compare the proposed gradient boost neural decision forest (GrNDF) with the one only has the base learner, i.e., neural decision forest (NDF) with convolutional autoencoder as the input processing layer.

4.2.3 Evaluation metrics

For a binary classification problem, the predicted value could be either true positive (TP), false positive (FP), false negative (FN), and true negative (TN). The four criteria for classification are defined as: $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$, $precision = \frac{TP}{TP+FP}$, $recall = \frac{TP}{TP+FN}$, $F1-score = 2 \cdot \frac{precision \cdot recall}{precision+recall}$. Among them, TP is the number of correctly predicted positive objectives, TN is the number of accurately predicted negative objectives, FP is the number of falsely predicted positive objectives and FN is the number of incorrectly predicted negative objectives. For a multi-classification problem, we take the one-vs-rest strategy, which regards one label as positive and the others are

TABLE 3
Comparison with Baseline Methods on Four Datasets

Epileptic	Method	GBDT	Random Forest	CNN	Peter et al.	Zhang et al.
	Accuracy	0.4961	0.5684	0.7531	0.7493	0.7485
	Precision	0.4937	0.5695	0.7496	0.7456	0.7468
	Recall	0.4992	0.5668	**0.7673**	0.7473	0.7481
	F1 score	0.4847	0.5679	0.7510	0.7462	0.7464
	Method	Li et al.	Dennis et al.	Zhou et al.	**NDF**	**GrNDF**
	Accuracy	0.7232	0.7479	0.6049	0.7570	**0.7650**
	Precision	0.7241	0.7464	0.5933	0.7546	**0.7611**
	Recall	0.7262	0.7461	0.6014	0.7549	0.7611
	F1 score	0.7244	0.7460	0.5925	0.7544	**0.7602**
Fashion MNIST	Method	GBDT	Random Forest	CNN	Peter et al.	Zhang et al.
	Accuracy	0.8837	0.8790	0.8994	0.8564	0.9164
	Precision	0.8820	0.8764	0.8994	0.8566	0.9163
	Recall	0.8840	0.8787	0.8987	0.8589	0.9173
	F1 score	0.8824	0.8764	0.8985	0.8564	0.9149
	Method	Li et al.	Dennis et al.	Zhou et al.	**NDF**	**GrNDF**
	Accuracy	0.8860	0.9159	0.8972	0.9150	**0.9201**
	Precision	0.8860	0.9159	0.8923	0.9072	**0.9201**
	Recall	0.8912	0.9178	0.8986	0.9083	**0.9203**
	F1 score	0.8870	0.9159	0.8954	0.9075	**0.9201**
MHEALTH	Method	GBDT	Random Forest	CNN	Peter et al.	Zhang et al.
	Accuracy	0.9414	0.5759	0.9799	0.9758	0.9809
	Precision	0.9422	0.6112	0.9810	0.9741	0.9809
	Recall	0.9352	0.5399	0.9811	0.9762	0.9810
	F1 score	0.9383	0.4979	0.9810	0.9751	0.9809
	Method	Li et al.	Dennis et al.	Zhou et al.	**NDF**	**GrNDF**
	Accuracy	0.9232	0.9797	0.7278	0.9563	**0.9820**
	Precision	0.9110	0.9796	0.7189	0.9543	**0.9824**
	Recall	0.9243	0.9811	0.7262	0.9583	**0.9825**
	F1 score	0.9157	0.9801	0.7233	0.9589	**0.9825**
Amazon Review	Method	GBDT	Random Forest	CNN	Peter et al.	Zhang et al.
	Accuracy	0.9220	0.7703	0.9025	0.9799	0.9538
	Precision	0.9301	0.7780	0.8898	0.9791	0.9527
	Recall	0.9118	0.7470	0.9126	0.9798	0.9524
	F1 score	0.9184	0.7529	0.8975	0.9794	0.9526
	Method	Li et al.	Dennis et al.	Zhou et al.	**NDF**	**GrNDF**
	Accuracy	0.7937	0.9610	0.8441	0.9828	**0.9880**
	Precision	0.7691	0.9596	0.8633	0.9828	**0.9883**
	Recall	0.8095	0.9602	0.8228	0.9819	**0.9871**
	F1 score	0.7767	0.9599	0.8324	0.9824	**0.9877**

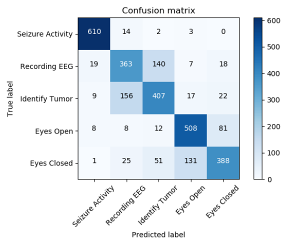
negative ones, by averaging the corresponding criteria for all the results.

4.3 Evaluation

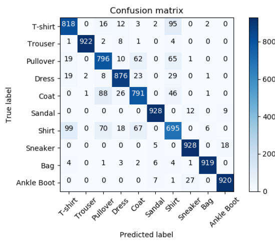
Here, we report the comparison of the proposed model with related work on four datasets: the Epileptic Seizure Recognition dataset, the Fashion MNIST, the MHEALTH dataset, and the Amazon Review dataset. Table 3 shows the comparison results. For each dataset, the first three methods are baseline methods followed by five state-of-the-arts, NDF is a base learner of our method, and GrNDF is the proposed framework.

We could see that neural network-based methods generally outperform the boosting or bagging of decision trees (i.e., GBDT and Random Forest). The above observation may result from the capability of neural networks for capturing the complex non-linear combination of features. The advantage is more evident on the Epileptic dataset, which contains sequential information that may not show significant patterns. Models that contain a CNN module (Peter et al., Zhang et al., Dennis et al., and our designs) perform better than the models without CNN modules. This is because of CNN's capability in capturing the neighborhood information, especially when dealing with images. The performance of the work by Li et al. is unwell on three datasets except for the Fashion MNIST. This may attribute to the structure of their proposed framework that: the prototype layer contains 'memorized' patterns for the past inputs, which requires similar patterns for new incoming inputs to provide accurate predictions. Then, for the dataset that the input distribution is diverse, the prototype layer may fail in capturing all the input patterns. When comparing our base learner NDF with the boosted version GrNDF, we could see that adding gradient boost module does improve the accuracy of the base learner. And in general, our proposed model, which combined neural decision forest with gradient boosting ideas, performs better than most of the comparison methods. Summarizing the above, we conclude that the proposed gradient boost neural decision forest model is a competitive, if not the best, classification model for various tasks.

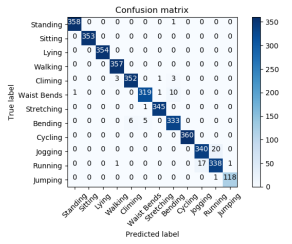
Figure 5 gives a confusion matrix about the details about



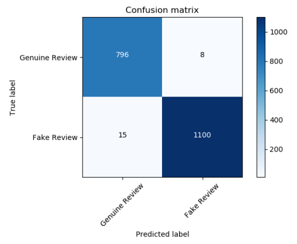
(a)



(b)



(c)



(d)

Fig. 5. Confusion Matrix on the prediction of (a) Epileptic Dataset, (b) Fashion MNIST dataset, (c) MHEALTH dataset, and (d) Amazon review dataset.

the prediction for four datasets. Notice that for the Fashion MNIST dataset, see Figure 5 (b), that instances that labeled as T-shirt, Pullover, Coat or Shirt are most difficult to discriminate. The predictor performs well in detecting sandal, sneaker and ankle boot, but sometimes these three also confuse the predictor. It fits the common sense that those four types of clothes and three types of shoes mentioned above sometimes are looking similar to each other.

4.4 Parameter Tuning

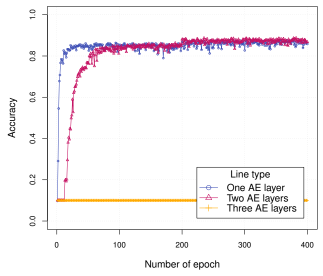
As mentioned above, we mainly have parameters in the input processing layers, the neural decision forest, and the gradient boost module. The input processing layer, in this case, is convolutional autoencoders. Here, we consider the model's performance under different numbers of autoencoder layers and numbers of fully connected layers that follow the latent representation of the autoencoder. The parameters of the neural decision forest include the depth of a neural decision tree and the number of trees in a forest. As for the gradient boost module, we consider the performance under the model without gradient boost, with one gradient boost module, or with two gradient boost modules. We also consider the activation function of each layer that may affect the model's efficiency (e.g., the Relu activation function ignores negative values). Table 4 summarizes the settings we used in experiments.

We tune the parameters on the Fashion-MNIST dataset, which delivers similar results as on other datasets. As a default setting, the input processing layer includes two convolutional layers and one fully-connected layer; the neural decision forest is with 20 trees and depth of 3; the number of gradient boost module is 1; the number of neurons in each neural layer is controlled under a certain scope (e.g., four times of the number of features).

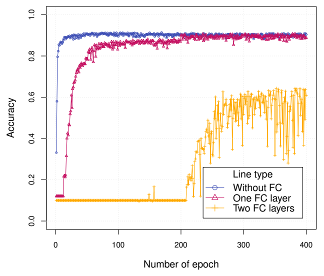
The results are shown in Figure 6. As for the option for the number of convolutional layers in the autoencoder, we can see that (in figure 6(a)) a model with two convolutional layers performs better than the model with only one convolutional layer, but the one-layer-model can quickly capture the feature pattern and get acceptable performance. Besides, it is hard for a three-layer-model to learn from the inputs. A similar conclusion can be drawn from figure 6 (b), where a model without fully-connected layers can rapidly converge to an acceptable accuracy level. Meanwhile, with the increase in the number of layers, it takes more epochs for a model to obtain convergence. As for the settings towards the neural decision trees, we can see that deeper trees obtain good performance faster than the shallow trees

TABLE 4
Parameter options

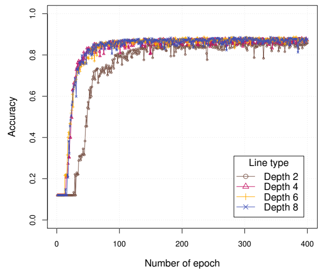
Parameter	Options
# autoencoder layers	{1, 2, 3}
# fully connected layers	{0, 1, 2}
Depth of a tree	{2, 4, 6, 8}
Number of trees in a forest	{5, 10, 20, 50, 100}
# GB modules	{0, 1, 2}
Activation function	{*sigmoid*, *ReLU*, *ReLU6*, *LeakyReLU*, *Tanh*}



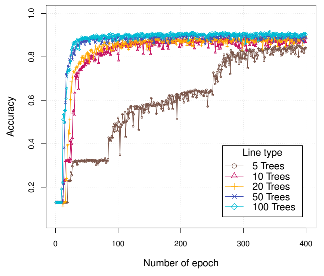
(a)



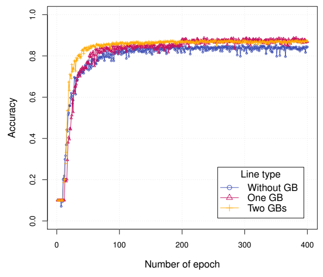
(b)



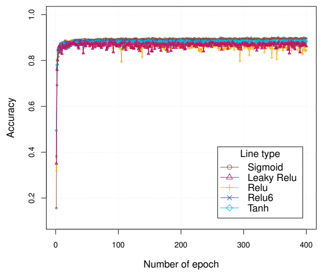
(c)



(d)

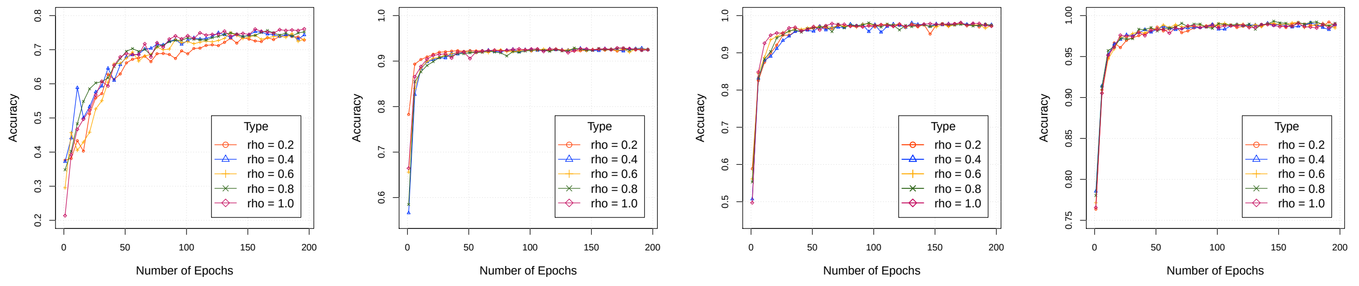


(e)



(f)

Fig. 6. The accuracy of different parameter settings on: (a) the number of autoencoder layers, (b) the number of fully-connected layers, (c) the depth of a neural decision tree, (d) the number of trees in a forest, (e) the number of gradient boost modules, (f) the activation function type.



(a)Epileptic dataset (b)Fashion MNIST (c)MHEALTH dataset (d)Amazon Review dataset

Fig. 7. The accuracy of different settings of hyper parameter ρ on dataset: (a)Epileptic dataset, (b) Fashion MNIST, (c) MHEALTH dataset and (d) Amazon Review dataset.

(shown in figure 6 (c)). According to figure 6 (d), a large forest (with many trees) learns faster than a small forest (with few trees). As for the settings of the gradient boost module, we can see that adding gradient boost modules does improve the model's performance, and model with two gradient boost modules performs better than the one with only one gradient boost module. Besides, the performance with two gradient boost modules is more stable during the training process. Lastly, for the settings of the activation function, we can see that different options have limited impact on the final results, where the model's performance fluctuates during the training process when using ReLU and LeakyReLU function. Overall, the above experiments show the importance of adding gradient boost module to the base model, which further supports the effectiveness of the proposed framework.

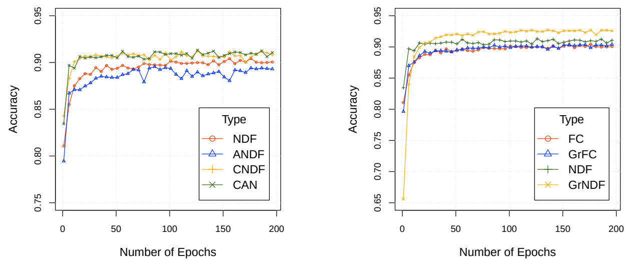
We further discuss the model's performance under different settings of hyper-parameter ρ, which stands for the step size when adding gradient boost modules. We present the results on four larger datasets in Figure 7. We can observe that ρ has slight differences in the impacts on different datasets, where Epileptic dataset is more sensitive to the ρ value, and a bigger ρ provides more stable prediction performance. For example, when $\rho = 1.0$, we can see from Figure 7 (a) that the model shows worse performance in the first few iterations than other settings, but then increases fast in the latter iterations and achieves the best performance. In a general case, a bigger ρ provides better performance while requiring more training iterations for the model to converge. This phenomenon also suggests the importance of the gradient boost module.

4.5 Ablation Study

Here, we discuss the performance of several variants of the proposed framework. Specifically, we consider the option for the input processing layer, the efficacy of the gradient boost module and the explainability of the proposed framework.

4.5.1 *The option for the input processing layer*

We show the model's performance with different input processing layers. In particular, we consider four modules: the fully-connected layers (NDF), the autoencoders (ANDF), the convolutional layers (CNDF), and the mentioned convolutional autoencoders (CAN). Notice that the abbreviations



(a)The option for the input processing layer (b)The efficacy of gradient boost

Fig. 8. Ablation study on (a)the way of learning the inputs and (b)the efficacy of gradient boost.

here are different from the ones in comparison results (section 4.3). For a fair comparison, the dimension of the neural layers and the settings for the base-learner and the gradient boost modules are the same for all methods. The results are shown in Figure 8 (a), which are based on the Fashion MNIST dataset. We could observe that the way of processing the inputs does affect the prediction results. In this case, CNN-based methods perform better than the others, which may attribute to the advantages of CNN-based methods in dealing with images information. Comparing CNDF and CAN, we can see that adding autoencoders can slightly improve model's performance. So generally, we can choose different input processing layers for different tasks, e.g., CNN-based modules for images and RNN-based modules for sequential inputs. The proposed framework is flexible in targeting different tasks.

4.5.2 *The efficacy of the gradient boost module.*

Our previous experiments have shown the effectiveness of adding the gradient boost modules in improving the model's performance over different tasks. Here, we verify such improvements when using different base learners to check whether the gradient boosting idea is also suitable for other base learners. We consider two base learners, i.e., the fully-connected layers (FC) and the proposed neural decision forest (NDF), and their according boosted versions (GrFC and GrNDF). The results in Figure 8 (b) show that adding gradient boost module improves the model's performance for different settings of the base learners, while such improvement is marginal when using fully-connected

Fig. 9. Example for the three classes in Iris dataset.

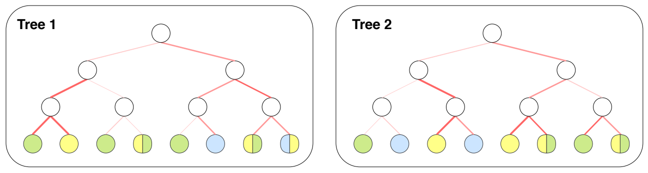


Fig. 10. An example of the prediction process of the neural decision forest when taking a Iris Versicolor as the input. The colored nodes are leaf nodes and have different distribution over the three classes (blue for Iris Setosa, yellow for Iris Versicolor, and green for Iris Virginica). The white nodes are decision nodes with different probabilities delivering the inputs to the leaf nodes.

layers as the base learner. This suggests that the proposed gradient boosting neural decision forest is efficient.

4.5.3 The explainability of the proposed framework

Here, we present a toy example of the explainability of the neural decision forest and show the model's performance on Iris dataset. Figure 9 gives examples of the three classes of iris plant. Figure 10 shows the prediction process of a neural decision forest (the base learner) that takes an Iris Versicolor as the input, where we set both the depth of a tree and the number of trees in a forest to 2. We can see that the model will deliver the inputs to the yellow nodes (for Iris Versicolor) with higher probability than two other classes. The model also shows noticeable probability of delivering the inputs to green nodes, i.e. Iris Virginica, which consist to the observation in Figure 9 that Iris Versicolor and Iris Virginica are similar in shape of the sepal and the petal but with different sizes. The neural decision forest enables the explainability and the flexibility of the prediction process: the prediction flow can be clearly presented; the prediction is made by all trees in a forest, where some trees may make errors in decisions but the prediction result can still be satisfactory contribute to other accurate trees.

5 Conclusion

Tree-based methods and deep neural networks are two scopes of machine learning methods that have been applied to various tasks. The traditional decision tree has both bagging and boosting versions: random forest and gradient boost decision tree; and the traditional idea of generalizing a tree is based on rules, making the learned tree sensitive to new inputs. In this work, we leverage the advantages of both tree-based methods and deep learning methods by combining the idea of bagging and boosting of trees on the level of neural networks. Specifically, we take neural decision forest as the base learner and add gradient boost modules to improve the model's performance. The resulting framework is both flexible and explainable: we can choose different input processing layers for different tasks; the dimension of the neural trees and the gradient boost modules can be modified for different scale of datasets; and the neural decision trees can show a probabilistic distribution of the prediction. We also demonstrated that the proposed framework does not take more computational power than traditional neural networks. Our extensive experiments demonstrate the advantages of the proposed framework.

References

[1] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.
[2] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[4] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.
[5] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, no. 2, pp. 197–227, 2016.
[6] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (icassp), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6645–6649.
[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
[8] G. Biau, E. Scornet, and J. Welbl, "Neural random forests," *Sankhya A*, pp. 1–40, 2016.
[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
[10] Z.-H. Zhou and J. Feng, "Deep forest: Towards an alternative to deep neural networks," *arXiv preprint arXiv:1702.08835*, 2017.
[11] M. Dong, L. Yao, X. Wang, B. Benatallah, C. Huang, and X. Ning, "Opinion fraud detection via neural autoencoder decision forest," *Pattern Recognition Letters*, vol. 132, pp. 21–29, 2020.
[12] J. Welbl, "Casting random forests as artificial neural networks (and profiting from it)," in *German Conference on Pattern Recognition*. Springer, 2014, pp. 765–771.
[13] M. Dong, L. Yao, X. Wang, B. Benatallah, X. Zhang, and Q. Z. Sheng, "Dual-stream self-attentive random forest for false information detection," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.
[14] J. Feng and Z.-H. Zhou, "Autoencoder by forest," in *Thirty-second AAAI conference on artificial intelligence*, 2018.
[15] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulò, "Deep neural decision forests," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1467–1475.
[16] H. Schwenk and Y. Bengio, "Boosting neural networks," *Neural Computation*, vol. 12, no. 8, pp. 1869–1887, 2000.
[17] F. Zhang, B. Du, and L. Zhang, "Scene classification via a gradient boosting random convolutional network framework," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 3, pp. 1793–1802, 2016.
[18] T. Hastie, R. Tibshirani, and J. Friedman, "Random forests," in *The elements of statistical learning*. Springer, 2009, pp. 587–604.
[19] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.
[20] S. Ghosh, N. Das, I. Das, and U. Maulik, "Understanding deep learning techniques for image segmentation," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–35, 2019.
[21] T. K. Ho, "Random decision forests," in *Document analysis and recognition, 1995., Proceedings of the Third International Conference on*, vol. 1. IEEE, 1995, pp. 278–282.
[22] L. Breiman, *Classification and regression trees*. Routledge, 2017.
[23] R. Johnson and T. Zhang, "Learning nonlinear functions using regularized greedy forest," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 5, pp. 942–954, 2014.
[24] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996.

[25] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[26] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[27] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[28] F. J. Ordóñez and D. Roggen, “Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition,” *Sensors*, vol. 16, no. 1, p. 115, 2016.

[29] J. Song, J. Xiao, F. Wu, H. Wu, T. Zhang, Z. M. Zhang, and W. Zhu, “Hierarchical contextual attention recurrent neural network for map query suggestion,” *IEEE Transactions on Knowledge & Data Engineering*, vol. 29, no. 09, pp. 1888–1901, 2017.

[30] I. K. Sethi, “Entropy nets: from decision trees to neural networks,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1605–1613, 1990.

[31] A. Montillo, J. Tu, J. Shotton, J. Winn, J. E. Iglesias, D. N. Metaxas, and A. Criminisi, “Entanglement and differentiable information gain maximization,” in *Decision Forests for Computer Vision and Medical Image Analysis*. Springer, 2013, pp. 273–293.

[32] J. Feng, Y. Yu, and Z.-H. Zhou, “Multi-layered gradient boosting decision trees,” *Advances in neural information processing systems*, vol. 31, pp. 3551–3561, 2018.

[33] J. Deng, Z. Zhang, F. Eyben, and B. Schuller, “Autoencoder-based unsupervised domain adaptation for speech emotion recognition,” *IEEE Signal Processing Letters*, vol. 21, no. 9, pp. 1068–1072, 2014.

[34] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” in *Neural Networks for Perception*. Elsevier, 1992, pp. 65–93.

[35] M. Chen, X. Shi, Y. Zhang, D. Wu, and M. Guizani, “Deep features learning for medical image analysis with convolutional autoencoder neural network,” *IEEE Transactions on Big Data*, no. 1, pp. 1–1, 2017.

[36] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[37] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, “Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state,” *Physical Review E*, vol. 64, no. 6, p. 061907, 2001.

[38] A. Mukherjee, B. Liu, and N. Glance, “Spotting fake reviewer groups in consumer reviews,” in *Proceedings of the 21st International Conference on World Wide Web*. ACM, 2012, pp. 191–200.

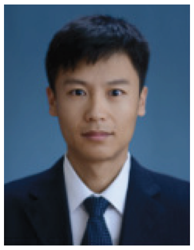
[39] O. Banos, C. Villalonga, R. Garcia, A. Saez, M. Damas, J. A. Holgado-Terriza, S. Lee, H. Pomares, and I. Rojas, “Design, implementation and validation of a novel open framework for agile development of mobile health applications,” *Biomedical engineering online*, vol. 14, no. 2, p. S6, 2015.

[40] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.

[41] D. Hamester, P. Barros, and S. Wermter, “Face expression recognition with a 2-channel convolutional neural network,” in *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015, pp. 1–8.

Manqing Dong is currently a Postdoctoral Research Fellow at the School of Computer Science and Engineering, University of New South Wales. She received her BE degree from Jilin University and her MSc in the City University of Hong Kong. Her research interests include anomaly detection, data mining, deep learning, and recommender systems. She is a member of IEEE and ACM.

Lina Yao received her B.Sc. degree in Computer Science from Shandong University, and M.Sc. and PhD degrees in computer science from the University of Adelaide. She is an associate professor in the School of Computer Science and Engineering, UNSW Sydney. Her research interests lie in data mining and machine learning applications, with a focus on Internet of Things analytics, human activity recognition, and Brain-Computer Interfaces. She is a member of the IEEE and the ACM.



Xianzhi Wang received his B.Sc. degree in Computer Science from Xi’an Jiaotong University, and M.Sc. and PhD degrees in computer science, from Harbin Institute of Technology. He is a lecturer at the School of Computer Science, University of Technology Sydney. His research interests include Internet of Things, data fusion, and machine learning. He is a member of the IEEE and the ACM.

Boualem Benatallah received his PhD in computer science from Grenoble University, France. He is a Scientia professor and research group leader at University of New South Wales. His research interests include Web services composition, quality control in crowdsourcing services, crowdsourcing for vulnerability discovery, data curation, cognitive services engineering, cloud services orchestration. He is a member of IEEE.

Shuai Zhang is a Postdoctoral Researcher at ETH Zürichis. He received his PhD degree from the school of computer science and engineering at the University of New South Wales. He received a Bachelor degree from the School of Information Management, Nanjing University. His major research interests lie in the field of recommender systems, deep learning and internet of things. He is a student member of the IEEE and ACM.

Quan Z. Sheng is a professor and Head of Department of Computing at Macquarie University. He received his PhD degree in computer science from the University of New South Wales. His research interests include service-oriented architectures, distributed computing, Internet computing, and Web of Things. He is the recipient of ARC Future Fellowship in 2014, Chris Wallace Award in 2012, and Microsoft Research Fellowship in 2003. He is a member of the IEEE and ACM.