DEEP NEURAL FORESTS: AN ARCHITECTURE FOR TABULAR DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural models, such as convolutional and recurrent networks, achieve phenomenal results over spatial data such as images and text. However, when considering tabular data, gradient boosting of decision trees (GBDT) remains the method of choice. Aiming to bridge this gap, we propose *deep neural forests* (DNF) – a novel architecture that combines elements from decision trees as well as dense residual connections. We present the results of extensive empirical study in which we examine the performance of GBDTs, DNFs and (deep) fully-connected networks. These results indicate that DNFs achieve comparable results to GBDTs on tabular data, and open the door to end-to-end neural modeling of multi-modal data. To this end, we present a successful application of DNFs as part of a hybrid architecture for a multi-modal driving scene understanding classification task.

1 INTRODUCTION

While deep neural models have gained supremacy in many applications, it is often the case that the winning hypothesis class in learning problems involving *tabular data* is *decision forests*. Indeed, in Kaggle competitions, gradient boosting decision trees (GBDTs) (Chen & Guestrin, 2016; Friedman, 2001) are often the superior model.[1] Decision forest techniques have several distinct advantages: they can handle heterogeneous feature types, they are insensitive to feature scaling, and perhaps, most importantly, they perform a rudimentary kind of "feature engineering" automatically by considering conjunctions of decision stumps. These types of features may be a key reason for the relative success of GBDTs over tabular data.

In contrast, deep neural models (CNNs, RNNs) have become the preeminent favorites in cases where the data exhibit a spatial proximity structure (namely, video, images, audio, and text). In certain problems, such as image classification, by restricting the model to exploit prior knowledge of the spatial structure (e.g., translation and scale invariances), these models are capable of generating problem dependent representations that almost completely overcome the need for expert knowledge. However, in the case of tabular data, it is often very hard to construct (deep) neural models that achieve performance on the level of GBDTs. In particular, the "default" fully connected networks (FCNs), which do not reflect any specific inductive bias toward tabular data, are often inferior to GBDTs on these data.

There have been a few works aiming at the construction of neural models for tabular data (see Section 2). However, for the most part, these attempts relied on conventional decision tree training in their loop and currently, there is still no widely accepted neural architecture that can effectively replace GBDTs. This deficiency prevents or makes it harder to utilize neural models in many settings and constitutes a lacuna in our understanding of neural networks.

Our objective in this work is to create a neural architecture that can be trained end-to-end using gradient based optimization and achieve comparable or better performance to GBDTs on tabular data. Such an architecture is desirable because it will allow the treatment of multi-modal data involving both tabular and spatial data in an integrated manner while enjoying the best of both GBDTs and deep models. Moreover, while GBDTs can handle medium size datasets ("Kaggle scale"), they do

[1]Kaggle winners post and analysis can be found at https://www.kaggle.com/bigfatdata/what-algorithms-are-most-successful-on-kaggle.

not scale well to very large datasets ("Google scale"), where their biggest computational disadvantage is the need to store (almost) the entire dataset in-memory[2] (see Appendix C for details as well as a real-life example of this limitation). A purely neural model for tabular data, which is trained with SGD, should be scalable beyond these limits.

A key-point in successfully applying deep models is the construction of architectures that contain inductive bias relevant to the application domain. This quest for appropriate inductive bias in the case of tabular data is not yet well understood (not to mention that there can be many kinds of tabular data). However, we do know that tree and forest methods tend to perform better than vanilla FCNs on these data. Thus, our strategy is to borrow properties of decision trees and forests into the network structure. We present a generic neural architecture whose performance can be empirically similar to GBDTs on tabular data. The new architecture, called *Deep Neural Forest* (DNF), combines elements from both decision forests and residual/dense nets. The main building block of the proposed architecture is a stack of *neural branches* (NBs), which are neural approximations of oblique decision branches that are connected via dense residual links (Huang et al., 2017). The final DNF we propose is an ensemble of such stacks (see details in Section 3).

We present an empirical study where we compare DNFs to the FCNs and GBDTs baselines, optimized over their critical parameters. We begin with a synthetic checkerboard problem, which can be viewed as a hypothetical challenging tabular classification task. We then consider several relatively large tasks, including two past Kaggle competitions. Our results indicate that DNFs consistently outperform FCNs, and achieve comparable performance to GBDTs. We also address applications of DNFs over multi-modal data and examine an integrated application of DNFs, CNNs and LSTMs over a multi-modal classification task for driving scene understanding involving both sensor recording and video (Ramanishka et al., 2018). We show that the replacement of the FCN component by DNF in the hybrid deep architecture of Ramanishka et al. (2018), which was designed to handle these multi-modal data, leads to significant performance improvement.

2 Related Work

There have been a few attempts to construct neural networks with improved performance on tabular data. In all these works, decision trees or forests are considered as the competition. A recurring idea in some of these works is the explicit use of conventional decision tree induction algorithms, such as ID3 (Quinlan, 1979), or conventional forest methods, such as GBDT (Friedman, 2001) that are trained over the data at hand, and then parameters of the resulting decision trees are explicitly or implicitly "imported" into a neural network using teacher-student distillation (Ke et al., 2018), explicit embedding of tree paths in a specialized network architecture (Seyedhosseini & Tasdizen, 2015), and explicit utilization of forests as the main building block of layers (Feng et al., 2018). This reliance on conventional decision tree or forest methods as an integral part of the proposed solution prevents end-to-end neural optimization, as we propose here. This deficiency is not only a theoretical nuisance but also makes it hard to use such models on very large datasets and in combination with other neural modules (see also discussion in Appendix Section C).

There are a few other recent techniques aiming to cope with tabular data using pure neural optimization. Yang et al. (2018) considered a method to approximate a single node of a decision tree using a soft binning function that transforms continuous features into one-hot features. The significant advantage of this tree based model is that it is intrinsically interpretable, as if it were a conventional decision tree. Across a number of datasets, this method obtained results comparable to a single decision tree and an FCN (with two hidden layers). This method, however, is limited to settings in which the number of features is small (e.g., 12). Focusing on microbiome data, a recent study by Shavitt & Segal (2018) presented an elegant regularization technique, which produces extremely sparse networks that are suitable for microbiome tabular datasets with relatively large feature spaces that only have a small number of informative features.

[2]This disadvantage is shared among all popular GBDT implementations be it XGBoost, LightGBM or CatBoost.

3 Deep Neural Forests

The main building block in our construction is a *Neural Branch* (NB). An NB represents a "soft conjunction" of a fixed number of (orthonormal) linear models. The purpose of an NB is to emulate the inductive bias existing in a path from the root to leaf in a decision tree (i.e., a branch). The second important element is the use of depth to allow for composite, hierarchical features. Thus, depth is created by vertically stacking layers of NBs using dense residual links as in DenseNet (Huang et al., 2017). We now provide a detailed description of the proposed architecture.

3.1 Neural Branches and Trees

When ignoring the hierarchy, a decision tree can be viewed as a disjunction of a set of conjunctions over decision stumps. Each conjunction corresponds to one path from the root to a leaf. Thus, any decision tree can be represented as a disjunctive normal form formula. A *Neural Tree* (NT) is an approximation of a disjunctive normal form formula. Each conjunction in the NT is called a neural branch (NB). While the basic units in a decision tree are decision stumps, the NT uses affine models, as in oblique decision trees (Murthy et al., 1994).

The NT is constructed using soft binary OR and AND gates. For a given (binary) vector $\mathbf{x} = (x_1, \ldots, x_d) \in \{-1, 1\}^d$. We implement soft, differentiable versions of such gates as follows.

$$\mathrm{OR}(\mathbf{x}) \triangleq \tanh\left(\sum_{i=1}^{d} x_i + d - 1\right), \qquad \mathrm{AND}(\mathbf{x}) \triangleq \tanh\left(\sum_{\mathrm{i}=1}^{\mathrm{d}} \mathrm{x_i} - \mathrm{d} + 1\right).$$

Notice that by replacing $\tanh$ by a binary activation, we obtain an exact implementation of the corresponding logical gates, which are well known (Anthony, 2005).[3] Importantly, both units do not have any trainable parameters.

For simplicity, given a vector $\mathbf{x} \in \mathbb{R}^d$ we define the $\mathrm{AND}(\mathbf{x})$ operator on r sub-groups, each of size k, as follows,

$$\mathrm{AND_{r,k}}(\mathbf{x}) \triangleq \big(\mathrm{AND}(\mathbf{x}_{[0,\mathrm{k}-1]}), \mathrm{AND}(\mathbf{x}_{[\mathrm{k},2\mathrm{k}-1]}), \ldots, \mathrm{AND}(\mathbf{x}_{[\mathrm{d}-\mathrm{k},\mathrm{d}-1]})\big) \in \mathbb{R}^{\mathrm{r}},$$

where $\mathbf{x}_{[i,j]}$ is a slice of $\mathbf{x}$ over the range $[i, j]$.

Formally, the NT is a three-layer network (two hidden layers), where only the first hidden layer, which represents the internal decision nodes (oblique separators), is trainable. Denoting by $\mathbf{x} \in \mathbb{R}^d$ a column of input feature vector, the functional form of an $\mathrm{NT(x)} : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}$ with a layer of r NBs, each NB with depth k, is

$$\begin{aligned}\mathrm{NB}(\mathbf{x}) &\triangleq \mathrm{AND_{r,k}}\big(\tanh(\mathbf{x}^{\mathrm{T}}\mathrm{W} + \mathrm{b})\big)\\ \mathrm{NT}(\mathbf{x}) &\triangleq \mathrm{OR}(\mathrm{NB}(\mathbf{x})) = \tanh\left(\sum_{\mathrm{i}=1}^{\mathrm{r}} \mathrm{NB}(\mathbf{x})_{\mathrm{i}} + \mathrm{r} - 1\right),\end{aligned}$$

where $\mathrm{NB}(\mathbf{x}) : \mathbb{R}^{\mathrm{d}} \to \mathbb{R}^{\mathrm{r}}$ is the output of r NBs, $W \in \mathbb{R}^{d \times kr}$ determines the (oblique) linear separators in each of the "nodes" such that each of its columns corresponds to one "node". and $b \in \mathbb{R}^{kr}$ is a bias vector term that corresponds to the threshold term in decision tree nodes. In our design, each decision node belongs only to a single branch.

When considering the decision boundaries induced by a single branch of an axis-aligned decision tree, it is clear that the decision boundary of a specific node is usually orthogonal to all the other decision boundaries defined by the other nodes in the same branch. We impose this constraint, which prevents unnecessary redundancy, by "encouraging" orthonormality through the loss function. Thus, when optimizing NTs, we include the following orthonormality constraint in our loss function, which imposes both orthogonality and unit length regularization simultaneously.

$$\text{Orthonormality Constraint} = \lambda ||W^T W - \tilde{I}||_F^2,$$

[3] See also (Shalev-Shwartz & Ben-David, 2014).

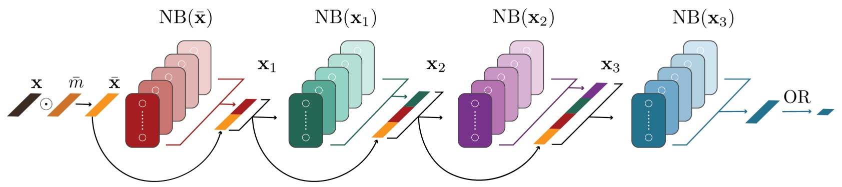


Figure 1: A DNT with four layers of NBs, each layer with five NBs. The input of each layer is a concatenation of the input of the previous layer with its output. Moreover, the input $\mathbf{x}$ is multiplied element-wise with a binary mask $\bar{m}$ (see section 3.3) before it is fed into the DNT.

where λ is a hyper-parameter, and $\tilde{I} \in \mathbb{R}^{kr \times kr}$ is defined by

$$\tilde{I}_{ij} = \begin{cases} 0, & \text{if nodes } i \text{ and } j \text{ are in the same branch;} \\ 1, & \text{if } i = j; \\ W^T W_{ij}, & \text{else.} \end{cases}$$

3.2 Deep Neural Trees

The "magic" in deep neural models is their ability to create a hierarchical structure by automatically creating composite features. One of the most interesting convolutional architectures is DenseNet (Huang et al., 2017) whose connectivity pattern ensures maximum information flow between layers. Each DenseNet layer is connected directly with each other layer. Moreover, in contrast to ResNet, which combines features through summation before they are passed into a layer, DenseNet combines features by concatenating them. A notable advantage of DenseNet is its improved flow of information and gradients throughout the network, which makes it easy to train. Moreover, the DenseNet connectivity pattern elicits the generation of composite features involving a mix of high and low-level features.

Since we want to retain these desirable properties of deep neural models, we introduce depth into our construct through dense residual links. Thus, A *Deep Neural Tree* (DNT) is a stack of layers of NBs that are interconnected using dense residual links, while the OR gate is applied only on the last NBs layer. Clearly, an NT is a DNT with a single layer of NBs and, in the sequel, we refer to an NT as a DNT. A diagram of a DNT with four layers of NBs appears in Figure 1.

3.3 Feature selection

One of the key components of decision trees is their greedy feature selection at any split. Such a component gives the decision trees, among other things, the ability to exclude irrelevant features. Li et al. (2016) presented a neural component for feature selection. Their solution is based on a heavily regularized (with elastic net regularization) mask that multiplies the input vector element-wise. In their work, they mention a crucial drawback in the proposed component, which arises in cases where the mask weight of a specific feature was approximately zero. In such cases, the corresponding weight in the first layer that multiplies this feature became very large. They tackled this problem by applying heavy regularization on the network layers.

In our study, for each DNT, we add an independent mask that multiplies the input vector element-wise. A heavy elastic net regularization is applied to the mask weights, and a binary threshold is used to circumvent the above pitfall. Denoting this mask by $m \in \mathbb{R}^d$, the feature selection component is formally defined as follows,

$$\text{elastic net regularization} \triangleq \lambda(\frac{1-\alpha}{2}||m||_2^2 + \alpha||m||_1)$$

$$\text{DifferentiableSign}(x) \triangleq \begin{cases} \text{sign}(x), & \text{forward pass;} \\ \sigma(x), & \text{backward pass;} \end{cases}$$

$$\bar{m} \triangleq \text{DifferentiableSign}(|m| - \epsilon),$$

Where $\bar{m}$ is the mask that multiplies the input, ϵ defines an epsilon neighborhood around zero for which the value of the mask is set to zero, and σ is the sigmoid activation. In words, if the value of the regularized mask is close to zero, set it to exact zero; otherwise, set it to one. Since the sign function is not differentiable, we use a smooth approximation of the sign function for calculating the gradients in the backward pass.

3.4 Deep Neural Forest

The power of decision trees can be significantly amplified when using many of them via ensemble methods, such as bagging or boosting. The final *Deep Neural Forest* (DNF) architecture is a weighted ensemble of DNTs (see a diagram on Appendix D). A DNF is implemented by concatenating the DNTs outputs and applying one fully-connected layer. The functional form of a $DNF(x)$ is,

$$DNF(\mathbf{x}) = \sum_{i=1}^{n} w_i DNT_i(\mathbf{x}),$$

where $w_i \in \mathbb{R}$ are trainable weights, which are optimized simultaneously with the DNTs. Accordingly, we will refer to a weighted ensemble of NTs as *Neural Forest* (NF).

It is well-known that high-quality ensembles should be constructed from a diverse set of low-bias base learners. To amplify ensemble diversity, we used both localization and random feature sampling. These techniques are applied individually for each base learner (DNT). Meir et al. (2000) showed the benefit of using an ensemble of localized base learners. Motivated by their result, we assign for each DNT a Gaussian with a trainable parameter mean vector μ, and a constant (isotropic) covariance matrix $\Sigma = \sigma^2 I$ (where σ is a fixed hyperparameter for the entire forest). For each instance $\mathbf{x}$, the output of the DNF is thus,

$$DNF(\mathbf{x}) = \sum_{i=1}^{n} w_i DNT_i(\mathbf{x}) \cdot D(\mathbf{x}^T W_p | \mu_i, \Sigma),$$

where D is the probability density function of the multivariate normal distribution, and W_p is a learnable projection matrix shared among all DNTs, which is used to obtain a linear embedding of the input to a low dimension. We note that this matrix is necessary to avoid learning of high-dimensional Gaussian, for which the probability density function is approximately zero (using isotropic covariance matrix $\Sigma = \sigma^2 I$ with $\sigma > 1$). This mechanism allows each DNT to specialize in a certain local sub-space and makes it oblivious to instances that are distant from its focal point μ.

Finally, another method we used is feature sampling, which is widely used in tree-based algorithms to increase diversity. Therefore, for each DNT, we randomly sample a fixed subset of features, where the number of features to be drawn is a hyper-parameter.

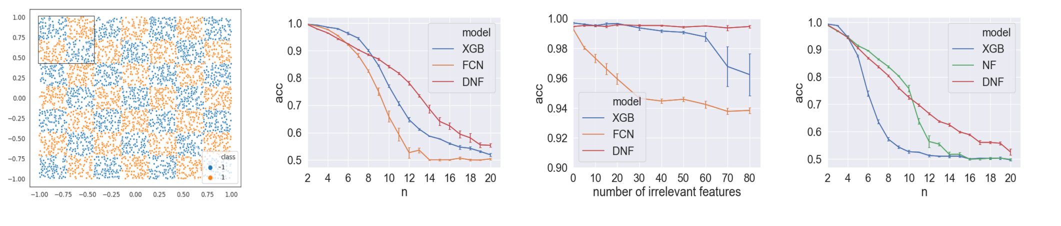
4 Empirical Study: Checkerboards

To gain some intuition and perspective on the performance of DNFs, GBDTs, and FCNs, in this section, we consider simple synthetic classification tasks that can be viewed as an extreme case of tabular data.

FCNs (even those with one hidden layer) are universal approximators (Cybenko, 1989) and can represent a good approximation to any (nicely behaved) function; nevertheless, training them using gradient methods is sometimes challenging. A well-known hard case is the problem of learning *parity*. While it is fairly easy to manually construct a small network that computes parity (Wilamowski et al., 2003), it is notoriously hard to train these networks to learn parity and similar problems using gradient methods (Shalev-Shwartz et al., 2017; Abbe & Sandon, 2018). Somewhat surprisingly, we show here that the training of FCNs is difficult even in much simpler *checkerboard* problems, which appear benign compared to parity.

In the *checkerboard* classification problem, the feature space, $\mathcal{X} = [-1, 1]^2$, is a two-dimensional square. Each of the two features is uniformly distributed. In a $n \times n$ checkerboard instance , the binary label, $\mathcal{Y} = \{\pm 1\}$, is defined by evenly dividing $\mathcal{X}$ into n^2 uniform squares, and the label is alternating along rows and columns as in the game of checkerboard. A 7×7 checkerboard example is depicted in Figure 2a, where blue and orange dots represent ± 1 labeled points sampled from the underlying distribution. Checkerboard problems naturally extend the XOR problem where a 2×2 checkerboard is XOR.

It is not hard to construct a 2-hidden layers FCN that solves the checkerboard perfectly. However, in the following experiment, we observe that such a perfect solution is not reachable via SGD training.



(a) (b) (c) (d)

Figure 2: (a) An example of a 7×7 checkerboard (XOR in the top left rectangle). (b) Mean accuracy (and standard error) of FCNs and GBDTs (XGBoost) over $n \times n$ checkerboard classification, $n = 2, \ldots, 20$. (c) A XOR-problem (2×2 checkerboard) with additional irrelevant features. (d) Low- and high-level features: XOR between checkerboards and additional binary feature.

Consider the following experiment where we tested FCNs, GBDTs and DNFs over 19 $n \times n$ checkerboard instances where $n = 2, 3, \ldots, 20$. For each of these checkerboard problems, we randomly generated 10,000 i.i.d. labeled samples (partitioned into to 1K points for training, 1K for validation and 8K for testing) over which we evaluated the performance of the three models. For GBDT we employed the powerful XGBoost implementation (Chen & Guestrin, 2016).

The hyperparameters for FCNs and GBDTs were aggressively optimized using an exhaustive grid search. For each checkerboard instance, a total of 1000 different configurations were tested for the FCNs, which included architectures with depth (number of hidden layers) in the range $[1, 4]$ and width in the set $\{64, 128, 256, 512, 1024\}$. Moreover, the hyperparameter optimization included a search for a dropout rate and L_1 regularization coefficient. The FCNs and the DNFs were trained using stochastic gradient descent (SGD) with the Adam optimizer and a learning rate scheduler. We did not limit the number of epochs, but we used an early stopping trigger consisting of 50 epochs. Accordingly, an exhaustive grid search was done for the decision tree algorithms where exact details of the hyperparameter ranges can be found on Appendix A in Table 3.

The checkerboard experiment results are depicted in Figure 2b. The x-axis is the checkerboard size (n), and the y-axis is accuracy over the test set, where each point is the mean over five independent trials and error bars represent standard error of the mean. We see that the performance of all three methods is deteriorated when the checkerboard size is increased. This tendency can be anticipated because the average number of training points in each checkerboard cell is decreasing (we keep the training set size 1000 for all boards). It is surprising, however, that FCNs completely fail to generate prediction better than random guessing for $n \geq 14$ board sizes. Moreover, it is evident that XGBoost consistently and significantly outperform the FCNs over all problems with $n > 2$. While DNFs are slightly behind at small ns, for all $n > 9$ they achieve the best results. Interestingly, the best results of the FCNs were obtained using networks containing millions of parameters, while DNFs mostly outperformed them using only (approximately) 4K trainable parameters. As a side note, we found that the batch size has a critical effect on the results; with mini-batches larger than 512, FCNs do not exhibit any advantage over random guessing for all board sizes $n \geq 14$.

Before continuing with additional synthetic setups, we emphasize that Checkerboard-like phenomena are quite common in tabular datasets. Consider, for example, the well-known Titanic dataset (Dua & Graff, 2017). Consider male age versus survival probability where a missing age is labeled with -1. Observing the figures on Appendix B, we can see a major increase from -1 to kids at ages 0–12. Beyond age 12 we see a major decrease for the adult male population. At age 25 we see a small increase and then again at age 45 we see a small decrease. This example also extends to the 2D case where both age and ticket fares are used.

4.1 XOR and Irrelevant Features

To demonstrate the ability of DNFs to deal with irrelevant features, we generate the data from a simple XOR-problem (2×2 checkerboard) with additional irrelevant features, where each irrelevant feature was drawn from the standard normal distribution. The results are depicted in Figure 2c. Clearly, DNFs sustain excellent performance with increasing numbers of irrelevant features, while we see some deterioration of the other methods. As might be expected, the top-performing FCNs were networks with one hidden layer and strong L_1 regularization. Here again, the representation efficiency of DNFs was evident; while the FCNs utilized around 10K neurons, the DNFs required less than 200 neurons. As the number of irrelevant features increases, it can be seen that XGBoost experiences some trouble, which we believe is mainly due to the high symmetry of the problem, where both relevant and irrelevant features have approximately zero information gain.

4.2 High- and Low-Level Features

The purpose of this last synthetic experiment is to examine the effect of depth (through dense residual connectivity) in DNFs. We, therefore, compare the performance of a DNF and a basic neural tree (NT), which does not include residual links (see Section 3.1). The data was generated using a checkerboard together with an additional binary feature that was uniformly sampled from $\{0, 1\}$. The label of each instance is a XOR between the binary feature and the label defined by the checkerboard. In order to solve this problem, an interaction of a low-level feature (the binary feature) with high-level features (the checkerboard pattern) must be learned. As in the first experiment, we considered 19 board sizes with $n = 2, 3, \ldots, 20$. The results can be seen in Figure 2d. While the NTs excel on checkerboards with $n \in \{4, \ldots, 8\}$, it is clear that for $n \geq 11$ the DNF is the leading model. FCNs were not included in this study because their performance on the checkerboard alone was already significantly inferior.

5 Experiments with tabular data

In this section, we examine the performance of DNFs and the baselines (FCNs and XGBoost) on several tabular datasets. The datasets used in this study are from Kaggle[4] and OpenML[5] (Vanschoren et al., 2014). A summary of these datasets appears on Appendix E.

For each dataset, all models were trained on the raw data without any feature engineering or selection. Only feature standardization was applied. Hyper-parameters for each model were optimized using a grid search (the range for each hyper-parameter in Appendix A in Table 4).

DNFs were trained using stochastic gradient descent (SGD) with the Adam optimizer and a learning rate scheduler. Dropout was applied to the layer obtained from the concatenation of the DNTs, and L_1 regularization was applied on the last layer (which computes a weighted sum of the DNTs). The FCNs were trained with SGD, Adam, and a learning rate scheduler as well. We did not limit the number of epochs but used an early stopping after 30 epochs.

The results are summarized in Table 1. For each dataset, the best result appears in bold. Notice that 'log loss' scores should be minimized and 'roc auc' – maximized. It is evident that the DNF performance is on par with the XGBoost performance, while FCNs are way behind.

Competition	Score	Deep Neural Forest	XGBoost	FCN
otto group	log loss	$\mathbf{0.43903 \pm 0.002186}$	0.44896 ± 0.00226	0.47784 ± 0.00423
otto group (kaggle test)	log loss	0.44774	$\mathbf{0.4477}$	0.48755
santander transaction	roc auc	89.047 ± 0.1568	$\mathbf{89.585 \pm 0.0612}$	86.525 ± 0.157
santander transaction (kaggle test)	roc auc	88.917	$\mathbf{89.549}$	86.525
churn	roc auc	$\mathbf{94.327 \pm 0.4024}$	93.319 $\pm$0.456	92.402 $\pm$0.781
magic telescope	roc auc	$\mathbf{94.234 \pm 0.1342}$	94.109 $\pm$0.0727	94.038 $\pm$0.122
gesture phase	log loss	$\mathbf{0.7723 \pm 0.0104}$	0.7836 $\pm$0.0128	1.0422 $\pm$0.0063
gas concentrations	log loss	$\mathbf{0.01380 \pm 0.00227}$	0.0186418 $\pm$0.00265	0.037909 $\pm$0.005316
eye movements	log loss	0.64255 $\pm$0.0041	$\mathbf{0.56091 \pm 0.0049}$	0.75584 $\pm$0.0035

Table 1: Tabular data experiments: mean score over 5-fold cross-validation. For the Kaggle competitions, we also included Kaggle-computed results obtained via the "late submission" system.

6 Application: end-to-end learning of multi-modal data

So far, GBDTs have dominated the tabular data domain, while the visual and textual domains have been entirely dominated by deep models (CNNs, RNNs). In cases of multi-modal tasks involving

[4]https://www.kaggle.com/competitions

[5]https://www.openml.org/home

tabular data as well, e.g., images, it is tempting to try and combine GBDTs and CNNs. However, as GBDTs are not differentiable (and not scaleable; see Appendix C), their integration with CNNs can be problematic. Typically, FCNs are used instead of GBDTs, and as we know, their utilization can degrade performance.

In this section, we examine the utilization of DNFs, instead of FCNs in a hybrid model to handle the multi-modal data of the Honda Research Institute Driving Dataset (HRI-DD) (Ramanishka et al., 2018). These data span 104 hours of real human driving and combine synchronized video and sensors measurements. The video consists of 1280×720 frames at 30 fps, and there are six different sensors: car speed, accelerator and braking pedal positions, yaw rate, steering wheel angle, and the rotation speed of the steering wheel. Four classification tasks were defined over these data, all of which are related to understanding driver behavior. We considered the first task: Goal-oriented action that involves the driver's manipulation of the vehicle in a navigation task. This is an 11-class task, and among the classes are 'right turn', 'left turn', 'branch', 'line change' and 'merge'.

In their study, Ramanishka et al. (2018) presented baseline results for this task. Their architecture consists of three components: for handling images, they used a CNN, whose main body is a pre-trained InceptionResnet-V2 (Szegedy et al., 2017), with an additional trainable convolutional layer. For the sensor data, they used an FCN. The embedding obtained from these two components was fused (concatenated) and then fed into an LSTM.

In our study, we utilized the exact same structure with exactly the same hyper-parameters and training parameters. The only change we made was to replace the FCN component with a DNF. We performed a comparative experiment on two tasks. The first task is to predict the navigation labels using only the sensors (i.e., in this setting the CNN was omitted), which is a composite multi-modal task that combines tabular data in a sequential manner (hence, the LSTM component remains). In the second task, we utilized both the video and the sensors. This task is a (composite) multi-modal task that combines tabular data and images as a time-series.

The results over these two tasks are summarized in Table 2. The baseline results (those with the FCN in their model) that we present were obtained by Ramanishka et al. (2018). For the sensors-only task, DNF leads to a 19.2% improvement in mean average precision. For the full data task, we obtained 23.9% improvement. These results indicate that there is much to be gained by better handling tabular data in such applications.

Models	right turn	left turn	intersection passing	railroad passing	left lane branch	right lane change	left lane change	right lane branch	crosswalk passing	merge	u-turn	mAP
FCN	74.27	66.25	36.41	0.07	8.03	13.39	26.17	0.20	0.30	3.59	33.57	23.84
DNF	79.40	77.49	46.68	0.07	22.13	5.60	12.34	3.62	0.61	8.82	55.92	**28.43**
FCN+CNN	77.47	76.16	76.79	3.36	25.47	23.08	41.97	1.06	11.87	4.94	17.61	32.71
DNF+CNN	79.81	75.04	81.27	2.73	43.50	33.12	39.60	10.56	23.13	6.94	50.31	**40.55**

Table 2: Each column is the average precision per class obtained on the test set. The last column is the mean average precision of all classes. The first two rows correspond to the sensor-only task and last two rows, to the sensors+video data.

7 Concluding Remarks

We introduced deep neural forest (DNF) – a novel deep architecture designed to handle tabular data. DNFs emulate some of the inductive bias existing in decision trees, and elicit the generation of composite features using depth and dense residual connectivity. Our empirical study of DNFs suggests that they significantly outperform FCNs over tabular data tasks, and achieve comparable performance to GBDTs, which so far were the SOTA choice for such data. Our initial study of a complex real-life multi-modal scenario of driving scene classification yielded substantial performance gains.

This work raises several interesting open challenges. First, more work is needed to fully substantiate DNFs and distill the essential elements in this architecture. Second, adopting the sequential optimization approach of GBDTs to DNFs can potentially lead to further large improvements, in the same way that GBDTs improve over random forests. Finally, we believe that a better theoretical understanding of the characteristics and inductive bias of tabular data can play a key role in achieving further performance gains in tabular and multi-modal settings.

REFERENCES

Emmanuel Abbe and Colin Sandon. Provable limitations of deep learning. *arXiv preprint arXiv:1812.06369*, 2018.

Martin Anthony. Connections between neural networks and Boolean functions. In *Boolean Methods and Models*, 2005.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794. ACM, 2016.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Ji Feng, Yang Yu, and Zhi-Hua Zhou. Multi-layered gradient boosting decision trees. In *Advances in Neural Information Processing Systems*, pp. 3555–3565, 2018.

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

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Guolin Ke, Jia Zhang, Zhenhui Xu, Jiang Bian, and Tie-Yan Liu. Tabnn: A universal neural network solution for tabular data. 2018.

Yifeng Li, Chih-Yu Chen, and Wyeth W Wasserman. Deep feature selection: theory and application to identify enhancers and promoters. *Journal of Computational Biology*, 23(5):322–336, 2016.

Ron Meir, Ran El-Yaniv, and Shai Ben-David. Localized boosting. In *COLT*, pp. 190–199. Citeseer, 2000.

Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32, 1994.

J Ross Quinlan. Discovering rules by induction from large collections of examples. *Expert systems in the micro electronics age*, 1979.

Vasili Ramanishka, Yi-Ting Chen, Teruhisa Misu, and Kate Saenko. Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Mojtaba Seyedhosseini and Tolga Tasdizen. Disjunctive normal random forests. *Pattern Recognition*, 48(3):976–983, 2015.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3067–3075. JMLR. org, 2017.

Ira Shavitt and Eran Segal. Regularization learning networks: Deep learning for tabular datasets. In *Advances in Neural Information Processing Systems*, pp. 1386–1396, 2018.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Bodgan M Wilamowski, David Hunter, and Aleksander Malinowski. Solving parity-n problems with feedforward neural networks. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 4, pp. 2546–2551. IEEE, 2003.

Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

A Hyperparameters ranges that considered in the grid search

The exact details of the hyperparameter ranges that considered in section 4 can be found in In Table . Accordingly, the details of section 5 can be found in In Table .

XGBoost		FCN	
parameter	**range**	**parameter**	**range**
n. estimators	$200-1000$	depth	$1-4$
learning rate	$0.01-0.2$	layer width	$64-1024$
max depth	$2-30$	L_1 reg lambda	$0-1e^{-4}$
sub sample	0.7	dropout	$0-0.4$

Table 3: Hyper-parameters range for the checkerboard experiment

Deep Neural Forest		XGBoost		FCN	
parameter	**range**	**parameter**	**range**	**parameter**	**range**
n. trees	1-400	n. estimators	50-1300	depth	1-3
n. layers	1-4	learning rate	0.005-0.3	layer width	64-512
n. branches	2-256	max depth	2-15	L_1 reg lambda	0.01 - 0.0001
branches depth	2-6	colsample by tree	0.2-1	dropout	0.1-0.4
keep feature prob	0-1	sub sample	0.5-1		
orthonormal lambda	0-0.1	reg lambda	0-3		
elastic net lambda	0-0.1				
elastic net alpha	0-1				

Table 4: Hyper-parameters range for the tabular datasets

B Checkerboard Effects in the Titanic Dataset

To demonstrate the checkerboard phenomena in tabular data, we plot the probability estimates for the Titanic dataset (Dua & Graff, 2017). The goal of this task is to predict individual passenger survival. For the demonstration here, we considered two real-valued features. The first is age, which might be missing and replaced with -1 (a common practice). The second is the ticket fare. The effect of gender is so great, so we chose to display plots on the male population. The first plot is univariate, where the x-axis is age and the y-axis is the survival probability. Clearly, there is a sharp transition change from -1 (missing data) to 0 (babies) and another sharp transition at 14 (kids). There are two more softer transitions at 25 and 42. These transitions indicate a checkerboard behavior. The second plot if bivariate, where the x-axis is age and the y-axis is ticket fare, while the color indicates the survival probability. The checkerboard-like pattern is apparent.

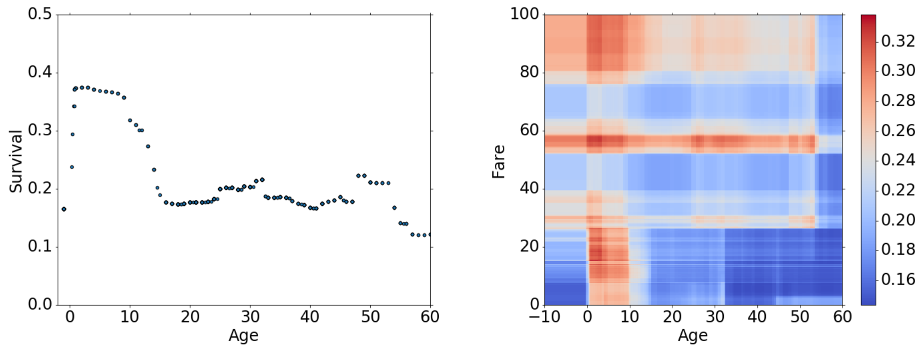


Figure 3: Survival probability of male population taken from the Titanic dataset. On the left plot, we see the survival vs age which exhibits multiple sharp transitions that resemble a 1D checkerboard behavior. On the right plot, the example is extended to a 2D checkerboard, survival vs age and fare.

C COMPUTATIONAL ISSUES WITH GRADIENT BOOSTING OF DECISION TREES

Gradient boosting (XGBoost, LightGBM, CatBoost) biggest computation disadvantage is the need to store (almost) the entire dataset in-memory. Several optimizations are deployed to help with this issue. LightGBM (max_bin parameter) and CatBoost (feature_border_type parameter) perform pre-computation that quantizes features to small integers. A random subset of the features can be selected for the entire tree (not just per node)[6]. At small to medium scale, these optimizations enable training to be performed efficiently on a single computer. But when considering large datasets, such as the Honda Research Institute Driving Dataset (HRI-DD) of Section 6, GBDT techniques are less effective. For instance, the HRI-DD set consists of $\sim$1.2M samples, where each sample is represented by 6 floats from the sensors, and $8 \times 8 \times 1536$ floats for the images. In order to hold all these data in memory we thus need $\sim$440GB of RAM. While such RAM sizes are available, to achieve reasonable performance on the HRI-DD task, one should model these data as a time-series, and each data point needs to be represented as a vector instances, resulting in memory requirements that can easily exceed the available RAM.

D DEEP NEURAL FOREST DIAGRAM

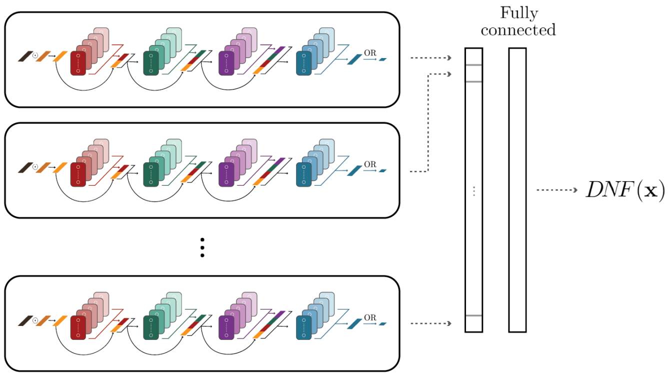


Figure 4: A DNF is implemented by concatenating the DNTs outputs and applying one fully-connected layer.

[6]In addition, a random subset of examples is used by each tree. While the main purpose of this reduction is regularization, it also reduces the memory footprint.

E TABULAR DATASETS DESCRIPTION

A description of the tabular datasets that were used in section 5,

Dataset	features	classes	instances	source
otto group	93	9	61.9k	Kaggle
santander customer transaction	200	2	200k	Kaggle
churn	22	2	5k	OpenML
magic telescope	10	2	19k	OpenML
gesture phase	32	5	10k	OpenML
gas concentrations	129	6	13.9k	OpenML
eye movements	27	3	10.9k	OpenML