Sparse tree-based Initialization for Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

Dedicated neural network (NN) architectures have been designed to handle specific data types (such as CNN for images or RNN for text), which ranks them among state-of-the-art methods for dealing with these data. Unfortunately, no architecture has been found for dealing with tabular data yet, for which tree ensemble methods (tree boosting, random forests) usually show the best predictive performances. In this work, we propose a new sparse initialization technique for (potentially deep) multilayer perceptrons (MLP): we first train a tree-based procedure to detect feature interactions and use the resulting information to initialize the network, which is subsequently trained via standard stochastic gradient strategies. Numerical experiments on several tabular data sets show that this new, simple and easy-to-use method is a solid concurrent, both in terms of generalization capacity and computation time, to default MLP initialization and even to existing complex deep learning solutions. In fact, this wise MLP initialization raises the resulting NN methods to the level of a valid competitor to gradient boosting when dealing with tabular data. Besides, such initializations are able to preserve the sparsity of weights introduced in the first layers of the network through training. This fact suggests that this new initializer operates an implicit regularization during the NN training, and emphasizes that the first layers act as a sparse feature extractor (as for convolutional layers in CNN).

1 Introduction

Neural networks have become hegemonic in machine learning in particular when dealing with very structured data. They indeed provide state-of-the-art performances for applications with images or text. However, they still perform poorly on tabular inputs, for which tree ensemble methods remain the gold standards (Grinsztajn et al., 2022).

Tree ensemble methods Tree-based methods are widely used in the ML community, especially for processing tabular data. Two main approaches exist depending on whether the tree building process is parallel (e.g. Random Forest, RF, see Breiman, 2001b) or sequential (e.g. Gradient Boosting Decision Trees, GBDT, see Friedman, 2001). In these tree ensemble procedures, the final prediction relies on averaging predictions of randomized decision trees, coding for particular partitions of the input space. The two most successful and most widely used implementations of these methods are XGBoost and LightGBM (see Chen & Guestrin, 2016; Ke et al., 2017) which both rely on the sequential GBDT approach.

Neural networks Neural Networks (NN) are efficient methods to unveil the patterns of spatial or temporal data, such as images (Krizhevsky et al., 2012) or texts (Liu et al., 2016). Their performance results notably from the fact that several architectures directly encode relevant structures in the input: convolutional neural networks (CNN, LeCun et al., 1995) use convolutions to detect spatially-invariant patterns in images, and recurrent neural networks (RNN, Rumelhart et al., 1985) use a hidden temporal state to leverage the natural order of a text. However, a dedicated *natural* architecture has yet to be introduced to deal with tabular data. Indeed, designing such an architecture would require to detect and leverage

the structure of the relations between variables, which is much easier for images or text (spatial or temporal correlation) than for tabular data (unconstrained covariance structure).

NN initialization and training In the absence of a suitable architecture for handling tabular data, the Multi-Layer Perceptron (MLP) architecture (Rumelhart et al., 1986) remains the obvious choice due to its generalist nature. Apart from the large number of parameters, one difficulty of MLP training arises from the non-convexity of the loss function (see, e.g., Sun, 2020). In such situations, the initialization of the network parameters (weights and biases) are of the utmost importance, since it can influence both the optimization stability and the quality of the minimum found. Typically, such initializations are drawn according to independent uniform distributions with a variance decreasing w.r.t. the size of the layer (He et al., 2015). Therefore, one may wonder how to capitalize on methods that are inherently capable of recognizing patterns in tabular data (e.g., tree-based methods) to propose a new NN architecture suitable for tabular data and an initialization procedure that leads to faster convergence and better generalization performance.

1.1 Related works

How MLP can be used to handle tabular data remains unclear, especially since a corresponding prior in the MLP architecture adapted to the correlations of the input is not obvious, to say the least. Indeed, none of the existing NN architectures can consistently match the performance of state-of-the-art tree-based predictors on tabular data (Shwartz-Ziv & Armon, 2022; Gorishniy et al., 2021; and in particular Table 2 in Borisov et al., 2021).

Self-attention architectures Specific NN architectures have been proposed to deal with tabular data. For example, TabNet (Arik & Pfister, 2021) uses a sequential self-attention structure to detect relevant features and then applies several networks for prediction. SAINT (Somepalli et al., 2021), on the other hand, uses a two-dimensional attention structure (on both features and samples) organized in several layers to extract relevant information which is then fed to a classical MLP. These methods typically require a large amount of data, since the self-attention layers and the output network involve numerous MLP.

Trees and neural networks Several solutions have been proposed to leverage information from pre-trained tree-based methods to develop NN that are capable of efficiently processing tabular data. For example, TabNN (Ke et al., 2018) first trains a GBDT on the available data, then extracts a group of features per individual tree, compresses the resulting groups, and uses a tailored Recursive Encoder based on the structure of these groups (with an initialization based on the tree leaves). Another approach consists in using pre-trained tree-based methods to initialize MLP. Translating decision trees into MLP was first achieved by Sethi (1990) and Brent (1991), and later exploited by Welbl (2014), Richmond et al. (2015) and Biau et al. (2019). Such procedures can be seen as a way to relax and generalize the partition geometry produced by trees and their aggregation. However, these approaches are restricted to shallow NN, as a decision tree is unfolded into a 3-layer neural network.

1.2 Contributions

In this work, we propose a new method to initialize a potentially deep MLP for learning tasks with tabular data. Our initialization consists in first training a tree-based method (RF, GBDT or Deep Forest, see Section 2.1) and then using its translation into a MLP as initialization. This procedure is shown to outperform the widely used uniform initialization of MLP (default initialization in Pytorch Paszke et al., 2019) in the following manner.

1. **Improved performances.** For tabular data, the predictive performances of the MLP after training are improved compared to MLP that use a random initialization. Our procedure also outperforms more complex deep learning procedures based on self-attention and is on par with classical tree-based methods (such as XGBoost).

2. **Faster optimization.** The optimization following a tree-based initialization is boosted in the sense that it enjoys a faster convergence towards a (better) empirical minimum: a tree-based initialization results in faster training of the MLP.

Regarding deep MLP, we show that initializing the first layers with tree-based methods is sufficient, other layers being randomly initialized in a conventional way. This supports the idea that in our method, the (first) tree-based initialized layers act as relevant embeddings that allow this standard NN architecture to detect correlations in the inputs.

Outline In Section 2, we introduce the predictors in play and describe how tree-based methods can be translated into MLP. The core of our analysis is contained in Section 3, where we describe in detail the MLP initialization process and provide extensive numerical evaluations showing the benefits of this method.

2 Equivalence between trees and MLP

Consider the classical setting of supervised learning in which we are given a set of input/output samples $\{(X_i, Y_i)\}_{i=1}^n$ drawn i.i.d. from some unknown joint distribution. Our goal is to construct a (MLP) function to predict the output from the input. To do so, we leverage the translation of tree-based methods into MLP.

2.1 Presentation of the predictors in play

Tree-based methods We consider three different tree ensemble methods: Random Forests (RF), Gradient Boosting Decision Trees (GBDT) and Deep Forests (DF). They all share the same base component: the Decision Tree (DT, for details see Breiman et al., 1984). We call its terminal nodes *leaf nodes* and the other nodes the *inner nodes*. RF (Breiman, 2001a) is a predictor consisting of a collection of M independently trained and randomized trees. Its final prediction is made by averaging the predictions of all its DT in regression or by a majority vote in classification. GBDT (Friedman, 2001) aims at minimizing a prediction loss function by successively aggregating DT that approximate the opposite gradient of that loss function (see Chen & Guestrin, 2016, for details on XGBoost). DF (Zhou & Feng, 2019) is a hybrid learning procedure in which random forests are used as elementary components (neurons) of a neural-network-like architecture (see Figure 5 and Appendix A for details).

Multilayer Perceptron (MLP) The multilayer perceptron is a predictor consisting of a composition of multiple affine functions, with (potentially different) nonlinear activation functions between them. Standard activation functions include, for instance, the rectified linear unit or the hyperbolic tangent. Deep MLP are a much richer class of predictors than tree-based methods which build simple partitions of the space and output piecewise constant predictions. Therefore, any of the tree-based models presented above can be approximated and in fact exactly rewritten as an MLP as follows.

2.2 An exact translation of tree-based methods into MLP

From decision tree to 3-layer MLP Recall that a decision tree codes for a partition of the input space in as many parts as there are leaf nodes in the tree. Given an input x, we can identify the leaf where x falls by examining for each hyperplane of the partition whether x falls on the right or left side of the hyperplane. The prediction is then made by averaging the outputs of all the points falling into the leaf of x. A DT can be thus translated into a two-hidden-layer MLP:

1. the first layer contains a number of neurons equal to the number of hyperplanes in the partition, each neuron encoding by ± 1 whether x falls on the left or right side of the hyperplane,
2. the second layer contains a number of neurons equal to the number of leaves in the DT. Based on the first layer, it identifies in which leaf x falls and outputs a vector with a single 1 at the leaf position and -1 everywhere else,

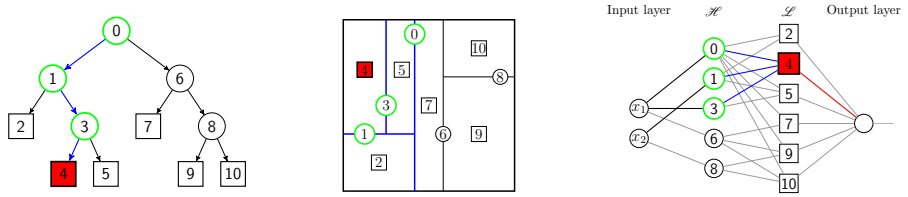

Figure 1: Illustration of a decision tree, its induced feature space partition and its corresponding MLP translation on a problem with two explanatory variables. Figure taken from Biau et al. (2019).

3. the last layer contains a single output neuron that returns the tree prediction. Its weights encode the value stored in each leaf of the tree.

The translation procedure is explained in details (and formally written mathematically) in Biau et al. (2019) and in Appendix B for completeness.

From RF/GBDT to 3-layer MLP Although RF and GBDT are constructed in different ways, they both average multiple DT predictions to give the final result. Thus, to translate a RF or a GBDT into an MLP, we simply turn each tree into a 3-layer MLP as described above, and concatenate all the obtained networks to form a wider 3-layer MLP. When concatenating, we set all weights between the MLP translations of the different trees to 0, since the trees do not interact with each other in predicting the target value for a new feature vector. Note that no additional layer is required in the MLP translation for averaging the results of multiple decision trees, as this operation can directly be carried out by the third layer. To this end, the last layer of the MLP translation uses an identity activation function and can therefore include the averaging step across trees, which is itself a linear transformation of the tree outputs.

From Deep Forests to deeper MLP As a Deep Forest is a cascade of Random Forests, it can be therefore translated into an MLP containing the MLP translations of the different RF in cascade, resulting in a deeper and wider MLP (note that the obtained MLP counts a number of hidden layers as a multiple of 3). Furthermore, in the Deep Forest architecture, the input vector is concatenated to the output of each intermediate layer, as shown in Figure 5. To mimic these skip connections in the MLP, we add additional neurons to each layer, except for the last three, which encode an identity mapping. Figure 6 illustrates this concept. Note that recasting a deep forest into a deep MLP using this method may suffer from numerical instabilities altering the predictive behaviour, see Appendix C. This is due to a phenomenon of catastrophic cancellation, more likely to occur with deep MLP translations. This does not impact the sequel of the study, relying on the MLP approximations considered hereafter.

2.3 Relaxing tree-based translation to allow gradient descent training

As shown in the previous section, one can construct an MLP that exactly reproduces a tree-based predictor. However this translation involves (i) piecewise constant activation functions (sign) and (ii) different activation functions in a same layer (sign and identity when translating DF). These constraints can hinder the MLP training, relying on stochastic gradient strategies (requiring differentiability), as well as efficient implementation tricks, given that automatic differentiation libraries only support one activation function per layer. Therefore, given a pre-trained tree-based predictor (RF, GBDT or DF), we aim at relaxing its translation into a MLP, mimicking its behavior as closely as possible but in a compatible way with standard NN training.

From tree-based methods to differentiable MLP To do so, Welbl (2014); Biau et al. (2019) consider the differentiable $\tanh$ activation, well suited for approximating both the sign and identity functions. Indeed, this can be achieved by multiplying or dividing the output of

a neuron by a large constant before applying the function $\tanh$ and rescaling the result accordingly if necessary, i.e. for $a, c > 0$ large enough, $\text{sign}(x) \approx \tanh(ax)$ and $x \approx c \tanh\left(\frac{x}{c}\right)$.

We cannot choose a arbitrarily large as this would make gradients vanish during the network optimization (the function being flat on most of the space), and hinder the training. We therefore introduce 4 hyper-parameters for the MLP encoding of any tree-based method that regulate the degree of approximation for the activation functions after the first, second and third layers of a decision tree translation, as well as for the identity mapping, respectively denoted by `strength01`, `strength12`, `strength23` and `strength_id`.

Hyperparameter choice The use of the tanh activation function involves extra hyperparameters. We study the influence of each one, by making them vary in some range (keeping the others fixed to 10^{10}, resulting in an almost perfect approximation of the sign and identity functions), see Appendix C.1 for details. Our analysis shows that increasing the hyperparameters beyond some limit value is no longer beneficial (as the activation functions are already perfectly approximated) and, across multiple data sets, these limit values are similar. We also exhibit relevant search spaces that will allow us to find optimal HP values for each application.

3 A new initialization method for MLP training

In this section, we study the impact of tree-based initialization methods for MLP training when dealing with tabular data. The latter empirically proves to be always preferable to standard random initialization and makes MLP a competitive predictor for tabular data.

3.1 Our proposal

Random initialization is the most common technique for initializing MLP before stochastic gradient training. It consists in setting all layer parameters to random values of small magnitude centered at 0. More precisely, all parameter values of the j-th layer are uniformly drawn in $[-1/\sqrt{d_j}, 1/\sqrt{d_j}]$ where d_j is the layer input dimension; this is the default behaviour of most MLP implementations such as `nn.Linear` in PyTorch (Paszke et al., 2019).

We introduce new ways of initializing an MLP for learning with tabular data, by leveraging the recasting of tree-based methods in a neural network fashion:

- **RF/GBDT initialization.** First, a RF/GBDT is fitted to the training data and transformed into a 3-layer neural network, following the procedure described in Section 2. The first two layers of this network are used to initialize the first two layers of the network of interest. Thus, upon initialization, these first two layers encode the RF partition. The parameters of the third and all subsequent layers are randomly initialized as described above.
- **DF initialization.** Similarly as above, a Deep Forest (DF) using ℓ forest layers is first fitted to the training data. The first $3\ell - 1$ layers of the MLP are then initialized using the first $3\ell - 1$ layers of the MLP encoding of this pre-trained DF. The parameters of the 3ℓ-th and all subsequent layers are randomly initialized as explained above.

The tree-based initialization techniques may seem far-fetched at first glance, but they are actually consistent with recent approaches for adapting Deep Learning models for tabular data. The key to interpreting them is to think of the first (tree-based initialized) layers of the MLP as a feature extractor that produces an abstract representation of the input data (in fact, this is a vector encoding the tree-based predictor space partition in which the observation lies). The subsequent randomly initialized layers, once trained, then perform the prediction task based on this abstract representation (see MLP sparsity in Section 3.5 for details).

3.2 Experimental setup

Datasets & learning tasks We compare prediction performances on a total of 10 datasets: 3 regression datasets (Airbnb, Diamonds and Housing), 5 binary classification datasets

(Adult, Bank, Blastchar, Heloc, Higgs) and 2 multi-class classification datasets (Covertype and Volkert). We mostly chose data sets that are used for benchmarking in relevent literature: Adult, Heloc, Housing, Higgs and Covertype are used by Borisov et al. (2021) and Bank, Blastchar and Volkert are used by Somepalli et al. (2021). Moreover, we add Airbnb and Diamonds to balance the different types of prediction tasks. The considered datasets are all medium-sized (10–60k observations) except for Covertype and Higgs (approx. 500k observations). Details about the datasets are given in Appendix D.1.

Predictors We consider the following tree-based predictors: Random Forest (RF), Deep Forest (DF, Zhou & Feng, 2017) and XGBoost (denoted by GBDT, Chen & Guestrin, 2016). The latter usually achieves state-of-the-art performances on tabular data sets (see, e.g., Shwartz-Ziv & Armon, 2022; Gorishniy et al., 2021; Borisov et al., 2021). We also consider deep learning approaches: MLP with default uniform initialization (MLP rand. init.) or tree-based initialization (resp. MLP RF init., MLP GBDT init. and MLP DF init.); and a transformer architecture SAINT Somepalli et al. (2021). This complex architecture is specifically designed for applications on tabular data and includes self-attention and inter-sample attention layers that extract feature correlations that are then passed on to an MLP. For regression and classification tasks, we use the mean-squared error (MSE) and cross-entropy loss for NN training, respectively. We choose SAINT as a baseline model as it is reported to outperform all other NN predictors on most of our data sets (all except Airbnb and Diamonds, see Borisov et al., 2021; Somepalli et al., 2021).

Parameter optimization All NN are trained using the Adam optimizer (see Kingma & Ba, 2014, and details in Appendix D.5.2). The MLP hyper-parameters (HP) are determined empirically using the optuna library (Akiba et al., 2019) for Bayesian optimization. For most HP, we use the default search spaces of Borisov et al. (2021), and for all HP tuning the translation between tree-base methods and MLP, we have identified relevant search spaces (see Appendix C.1). An overview of all search spaces used for each method can be found in Appendix D.5. The quantity minimized during HP tuning is the model's validation loss, and the smallest validation loss that occurred during training for all MLP-based models.

3.3 A BETTER MLP INITIALIZATION FOR A BETTER OPTIMIZATION

In this subsection, the optimization of standard MLP is shown to benefit from the proposed initialization technique. Experiments have been conducted on 6 out of the 10 data sets.

Experimental protocol 1 (P1) To obtain comparable optimization processes, we ensure that all purely MLP-related hyper-parameters, i.e. the MLP width, depth and learning rate, are identical for all MLP in play regardless of the initialization technique. These HP are chosen to maximize the predictive performance of the standard randomly initialized MLP. All HP related to the initialization technique (HP of the tree-based predictor, HP tuning the translation

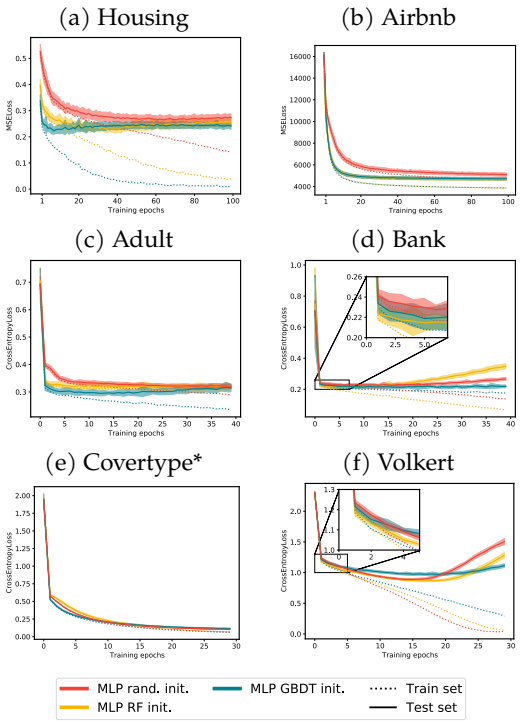

Figure 2: Optimization behaviour of randomly, RF and GBDT initialized MLP evaluated over a 5 times repeated (statisfied) 5-fold of each data set, according to Protocol P1. The lines and shaded areas report the mean and standard deviation. *evaluation on a single 5-fold cross validation.

Model \ Data set	Housing	Airbnb	Diamonds	Adult	Bank	Blastchar	Heloc	Higgs	Covertype	Volkert
	MSE ↓	MSE ↓ ($\times 10^3$)	MSE ↓ ($\times 10^{-3}$)	AUC ↑ (in %)	AUC ↑ (in %)	AUC ↑ (in %)	AUC ↑ (in %)	AUC ↑ (in %)	Acc. ↑ (in %)	Acc. ↑ (in %)
Random Forest	0.263±0.009	5.39±0.13	9.80±0.35	91.6±0.3	92.8±0.3	**84.5±1.2**	91.3±0.6	80.4±0.1	83.6±0.1	64.2±0.3
GBDT	**0.208±0.010**	**4.71±0.15**	**7.38±0.28**	**92.7±0.3**	**93.3±0.3**	**84.7±1.0**	**92.1±0.4**	82.8±0.1	**97.0±0.0**	71.3±0.4
Deep Forest	0.225±0.008	**4.68±0.16**	8.23±0.29	91.8±0.3	92.9±0.2	**83.7±1.2**	90.3±0.5	81.2±0.0*	92.4±0.1*	66.3±0.4
MLP rand. init.	0.258±0.011	5.07±0.16	15.5±12.5	90.5±0.4	91.0±0.3	81.4±1.2	80.1±0.1	83.2±0.3	96.7±0.0	72.2±0.4
MLP RF init.	0.222±0.009	**4.66±0.16**	7.93±0.22	92.1±0.3	92.4±0.4	**84.4±1.2**	**91.7±0.4**	**83.6±0.1**	96.7±0.0	**74.1±0.4**
MLP GBDT init.	**0.206±0.007**	**4.70±0.09**	8.15±0.35	92.2±0.3	92.5±0.3	**84.6±1.2**	91.5±0.6	83.0±0.0	96.2±0.0	73.5±0.5
MLP DF init.	0.234±0.016	**4.81±0.13**	8.28±0.24	91.9±0.4	92.2±0.3	**84.2±1.0**	91.4±0.6	83.3±0.1*	94.5±0.3*	71.3±0.5
SAINT	0.258±0.011	**4.81±0.15**	17.7±3.83	91.6±0.3	92.2±0.4	**84.0±0.8**	90.2±0.7	**83.7±0.1***	96.6±0.1*	70.1±0.4

Table 1: Best scores for Protocol 2. For each data set, predictors performing at least as well as the best (resp. DL) score up to its standard deviation are highlighted in **bold** (resp. underlined). The scores are based on 5 times repeated (stratified) 5-fold cross validation. For each model, HP have been chosen via the "optuna" library with 100 iterations. *score based on a simple 5-fold cross validation. See Appendix D.4.1 for a comparison with literature results.

between trees and MLP) are optimized independently for each tree-based initialization method.

Results Figure 2 shows that for most data sets, the use of tree-based initialization methods for MLP training provides a *faster convergence* towards *a better minimum* (in terms of generalization) than random initialization. This is all the more remarkable since Protocol P1 has been calibrated in favor of random initialization. Among tree-based initializers, GBDT initializations outperform or are on par with RF initializations in terms of the optimization behavior on all regression and binary classification problems. However, for multi-class classification problems, the advantages of tree-based initialization seem to be limited. This is probably due to the fact that the MLP architecture at play is tailored for random initialization, being thus too restrictive for tree-based initializers. Experiments presented in Appendix D.3 with fixed arbitrary widths corroborate this idea: in this case, the RF initialization is beneficial for the optimization process. For the Adult, Bank, and Volkert data sets, Figure 2 also shows the performance of each method at initialization. None of these procedures leads to a better MLP performance at initialization (due to both the non-exact translation from trees to MLP and to the additional randomly initialized layers), but rather help to guide the MLP in its learning process.

3.4 A better MLP initialization for a better generalization

In this subsection, tree-based initialization methods are shown to systematically improve the predictive power of neural networks compared to random initialization.

Experimental protocol 2 (P2) Each MLP is trained on 100 epochs, but with HP tuned depending on the initialization technique. For maximum comparability, the optimization budget is strictly the same for all methods (100 "optuna" iterations each, where one optuna iteration includes a hold-out validation). In particular, when using a tree-based initializer, we use 25 HP optimization iterations to find optimal HP for the tree-based predictor, fix these HPs, and then use the remaining 75 iterations to determine optimal HP for the MLP. For all NN approaches, the model with the best performance on the validation set during training is kept (using the classical *early-stopping* procedure). Performances are measured via the AUROC score for binary classification, accuracy for multi-class classification and MSE for regression (using 5 runs of 5-fold cross-validation).

Results Table 1 shows that RF or GBDT initialization strictly outperform random initialization, in terms of final generalization performance, for all data sets except Covertype (for which performances are similar). Additionally, MLP using both RF and GBDT initialization techniques outperform SAINT on all medium-sized data sets and fall short on large data sets (Higgs and Covertype). Despite its simplicity and the additional random/noisy layers, the proposed methods (based on RF or GBDT) are on par with GBDT half of the data sets, ranking MLP as relevant predictors for tabular data. Among the tree-based initializers,

GBDT initialization performs better than the RF one on regression and binary classification problems but not for multi-class classification, which may result from how GBDT deals with multiclass labels (see Appendix D.3 for explanation). DF initialization, for its part, cannot compete with RF and GBDT initialization, despite showing some improvement over the random one (except for Covertype and Volkert). This underlines that injecting prior information via tree-based methods into the first layers of a MLP is sufficient to improve its performance.

The interested reader may find a comparison of the optimization procedures of all MLP methods and SAINT (Figure 11) and a table summarizing all HP (Table 9) in Appendix D.5.2. We remark that tree-based initializers generally bring into play wider networks with similar depths (fixed width of 2048 and an optimized depth between 4 and 7) compared to MLP with default initialization. But, for most data sets, the overall procedure is computationally more efficient than state-of-the-art deep learning architectures such as SAINT. This hold for the number of parameters, the training time and the test time (see Table 5-7 in Appendix D.4).

3.5 Analyzing key elements of the new initialization methods

Influence of the MLP width We mainly use standard search spaces from (Borisov et al., 2021) to determine the optimal hyper-parameters for each model. However, the MLP width is an exception to this. The standard search spaces used in the literature usually involve MLP with a few hundred neurons per layer (e.g. up to 100 neurons in Borisov et al., 2021); yet, in this work, we consider MLP with a width up to 2048 neurons. Large MLP are actually very beneficial for tree-based initialization methods as they allow the use of more expressive tree-based models in the initialization step.

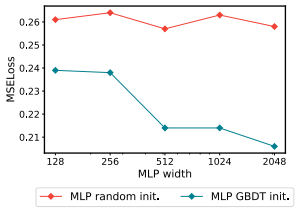

Figure 3: Influence of width on the final generalization performance for random and GBDT initializations. Mean values over 5 times repeated 5-fold cross-validation on the housing data set.

Figure 3 compares the performance of an MLP with random/GDBT initializations and different widths. There is no gain in prediction by using wider (and therefore more complex) NN, when randomly initialized. This is corroborated by the results of Table 4: for all regression and binary classification data sets, the performance of our (potentially much wider) MLP with random initialization is consistently very close to the literature values, and only increases for multi-class classification problems. However, an MLP initialized with GBDT significantly benefits from enlarging the NN width (justifying a width of 2048 for tree-based initialized MLP in Table D.4.1). This confirms the idea that tree-based initialization helps revealing relevant features to the MLP, all the more as the NN width increases, and by doing so, boost the MLP performance after training.

Performance of the initializer For the regression and binary classification problems considered here, the GBDT model trained before the NN initialization always performs significantly better than the RF model; for the multi-class classification problems, the differences are tenuous (see Figure 12 in Appendix D.6). Note, however, that a better performance of the tree-based predictor used for initialization does not always lead to a better performance after training (see the examples of Airbnb and Volkert data sets in Figure 12). This observation suggests that the predictive performance of the initializer (GBDT or RF) is not sufficient in itself to explain the predictive power of the trained network: other aspects, such as the expressiveness of the feature interactions captured by the initializer or the structure it induces on the MLP, must also play a significant role.

MLP sparsity Figure 4 shows the histograms of the weights of the three first and last layers before and after MLP training, for random, RF and GBDT initializations on the Housing data set (see Appendix D.7 for the Adult data set). The weight distribution on the first two layers changes significantly during training when the MLP parameters are randomly initialized: the weights are uniformly distributed at epoch 0 but appears

to be Gaussian (closely centered around zero) after training. When RF or GBDT initializers are used instead, the weights of the first two layers at epoch 0 are sparsely distributed by construction, and their distribution is preserved during training (notice the logarithmic y-axis for these plots in Figure 4). Note that the (uniform) distribution of the weights in other layers is also preserved through training (third and last lines of Figure 4). This means that our initialization technique, in combination with SGD optimization strategies, introduces an *implicit regularization* of the NN optimization: the obtained MLP is structured and consists of sparse feature extraction layers (with a structure that is particularly suited for capturing features in tabular data), followed by dense prediction layers. This is very similar to the architecture of CNN (constrained by design), a very successful class of NN designed specifically for image processing.

It has been argued (see, e.g., Section 2.2 in Neal, 2012) that "with Gaussian priors the contributions of individual hidden [neurons] are all negligible, and consequently, these [neurons] do not represent 'hidden features' that capture important aspects of the data." The argument further states that a layer can only capture patterns in the data if it has a sufficient number of individual neurons with non-negligible weights. Indeed, empirical evaluations show that MLP whose weights are tightly centered around zero have worse predictive abilities than those that use a wider range of weights (Blundell et al., 2015). In this line of reasoning, the sparse weight distribution of the first two layers we observe before and after training of tree-based-initialized MLP shows how well these layers are adapted to detect features in the data.

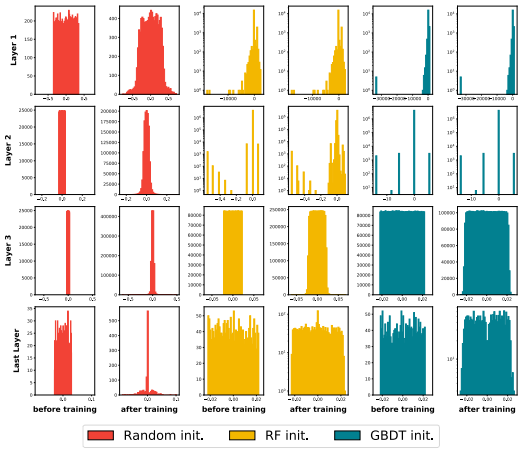

Figure 4: Histograms of the first three and the last layers' weights before and after the MLP training on the Housing data set. Comparison between random, RF and GBDT initializations.

4 Conclusion and Future work

This work builds upon the permeability that can exist between tree methods and neural networks, in particular how the former can help the latter during training, especially with tabular inputs. More precisely, we have proposed new methods for smartly initializing standard MLP using pre-trained tree-based methods. The sparsity of this initialization is preserved during training which shows that it encodes relevant correlations between the data features. Among deep learning methods, such initializations (via GBDT or RF) of MLP always improve the performance compared to the widely used random initialization, and provide an easy-to-use and more efficient alternative to SAINT, the state-of-the-art attention-based deep learning method for tabular data. Beyond deep learning models, the performance of this wisely-initialized MLP is remarkably approaching that of XGBoost, which so far reigns supreme for learning tasks on tabular data.

There is still undoubtedly room for improvement to democratize such a network initialization. For example, one could be interested in extending this method to MLP with arbitrary activation functions (other than $\tanh$). A first step would be to code sign and identity functions, for instance using the ReLU activation.

Our approach also yields a default method for choosing the MLP width, which results from the number of trees and their depth used in the initialization technique. Optimizing DF naturally leads to a default choice of MLP depth. However DF performances are not sufficient to legitimate such procedures for now. Other ways of translating tree structures into deep networks should be considered to build upon our work and propose new data-driven choices of MLP depth.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 6679–6687, 2021.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):1–9, 2014.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Gérard Biau, Erwan Scornet, and Johannes Welbl. Neural random forests. *Sankhya A*, 81 (2):347–386, 2019.

C.M. Bishop, P.N.C.C.M. Bishop, G. Hinton, and Oxford University Press. *Neural Networks for Pattern Recognition*. Advanced Texts in Econometrics. Clarendon Press, 1995. ISBN 9780198538646. URL https://books.google.fr/books?id=T0S0BgAAQBAJ.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International conference on machine learning*, pp. 1613–1622. PMLR, 2015.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey, 2021. URL https://arxiv.org/abs/2110.01889.

L Breiman, JH Friedman, R Olshen, and CJ Stone. Classification and regression trees. 1984.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001a. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001b.

Richard P Brent. Fast training algorithms for multilayer neural nets. *IEEE Transactions on Neural Networks*, 2(3):346–354, 1991.

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, 2016.

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

Wei Fan, Haixun Wang, Philip S Yu, and Sheng Ma. Is random model better? on its accuracy and efficiency. In *Third IEEE International Conference on Data Mining*, pp. 51–58. IEEE, 2003.

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

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data, 2021. URL https://arxiv.org/abs/2106.11959.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815*, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

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

R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997. ISSN 0167-7152. doi: https://doi.org/10.1016/S0167-7152(96)00140-X. URL https://www.sciencedirect.com/science/article/pii/S016771529600140X.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.

Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

David L Richmond, Dagmar Kainmueller, Michael Y Yang, Eugene W Myers, and Carsten Rother. Relating cascaded random forests to deep convolutional neural networks for semantic segmentation. *arXiv preprint arXiv:1507.07583*, 2015.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(45-76):26, 1986.

Ishwar Krishnan Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, 1990.

Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2021.11.011. URL https://www.sciencedirect.com/science/article/pii/S1566253521002360.

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training, 2021. URL https://arxiv.org/abs/2106.01342.

Ruo-Yu Sun. Optimization for deep learning: An overview. *Journal of the Operations Research Society of China*, 8(2):249–294, 2020.

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020, 2021. URL https://arxiv.org/abs/2104.10201.

Johannes Welbl. Casting random forests as artificial neural networks (and profiting from it). In *German Conference on Pattern Recognition*, pp. 765–771. Springer, 2014.

Z. Zhou and J. Feng. Deep forest. *National Science Review*, 6(1):74–86, 2019.

Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 3553–3559, 2017. doi: 10.24963/ijcai.2017/497. URL https://doi.org/10.24963/ijcai.2017/497.