TabNet: Attentive Interpretable Tabular Learning

Sercan Ö. Arık
Google Cloud AI
soarik@google.com

Tomas Pfister
Google Cloud AI
tpfister@google.com

ABSTRACT

We propose a novel high-performance and interpretable canonical deep tabular data learning architecture, TabNet. TabNet uses sequential attention to choose which features to reason from at each decision step, enabling interpretability and more efficient learning as the learning capacity is used for the most salient features. We demonstrate that TabNet outperforms other neural network and decision tree variants on a wide range of non-performance-saturated tabular datasets and yields interpretable feature attributions plus insights into the global model behavior. Finally, for the first time to our knowledge, we demonstrate self-supervised learning for tabular data, significantly improving performance with unsupervised representation learning when unlabeled data is abundant.

KEYWORDS

Interpretable deep learning, tabular data, attention, self-supervised.

1 INTRODUCTION

Deep neural networks (DNNs) have shown notable success with images [21, 50], text [9, 34] and audio [1, 56]. For these data types, a major enabler of the progress is the availability of canonical DNN architectures that efficiently encode the raw data into meaningful representations, resulting in high performance on new datasets and related tasks with minor effort. For example, in image understanding, variants of residual convolutional networks (e.g. ResNet [21]) provide reasonably-well performance on new image datasets or slightly different visual recognition problems (e.g. segmentation).

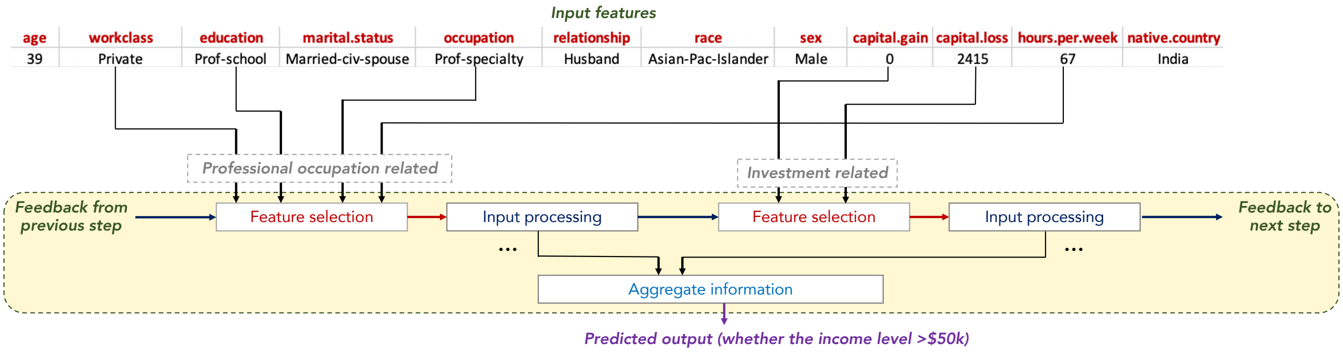
One data type that has yet to see such success with a canonical DNN architecture is tabular data. Despite being the most common data type in real-world AI[1] [8], deep learning for tabular data remains under-explored, with variants of ensemble decision trees still dominating most applications [28]. Why is this? First, because tree-based approaches have certain benefits that make them popular: (i) they are representionally efficient (and thus often high-performing) for decision manifolds with approximately hyperplane boundaries which are common in tabular data; and (ii) they are highly interpretable in their basic form (e.g. by tracking decision nodes) and there are effective post-hoc explainability methods for their ensemble form, e.g. [36] – this is an important concern in many real-world applications (e.g. in financial services, where trust behind a high-risk action is crucial); (iii) they are fast to train. Second, because previously-proposed DNN architectures are not well-suited for tabular data: conventional DNNs based on stacked convolutional layers or multi-layer perceptrons (MLPs) are vastly overparametrized – the lack of appropriate inductive bias often causes them to fail to find optimal solutions for tabular decision manifolds [17].

Why is deep learning worth exploring for tabular data? One obvious motivation is that, similarly to other domains, one would expect performance improvements from DNN-based architectures particularly for large datasets [22]. In addition, unlike tree learning which does not use back-propagation into their inputs to guide efficient learning from the error signal, DNNs enable gradient descent-based end-to-end learning for tabular data which can have a multitude of benefits just like in the other domains it has already shown success: (i) it could efficiently encode multiple data types like images along with tabular data; (ii) it would alleviate or eliminate the need for feature engineering, which is currently a key aspect in tree-based tabular data learning methods; (iii) it would enable learning from streaming data – tree learning needs global statistics to select split points and straightforward modifications such as [4] typically yield lower accuracy compared to learning from entire data – in contrast, DNNs show great potential for continual learning[44]; and perhaps most importantly (iv) end-to-end models allow representation learning which enables many valuable new application scenarios including data-efficient domain adaptation [17], generative modeling [46] and semi-supervised learning [11]. Clearly there are significant benefits in both tree-based and DNN-based methods. Is there a way to design a method that has the most beneficial aspects of both?

In this paper, we propose a new canonical DNN architecture for tabular data, TabNet, that is designed to learn a 'decision-tree-like' mapping in order to inherit the valuable benefits of tree-based methods (interpretability and sparse feature selection), while providing the key benefits of DNN-based methods (representation learning and end-to-end training). In particular, TabNet's design considers two key needs: high performance and interpretability. As mentioned, high performance alone is often not enough – a DNN needs to be interpretable to substitute tree-based methods. Overall, we make the following contributions in the design of our method:

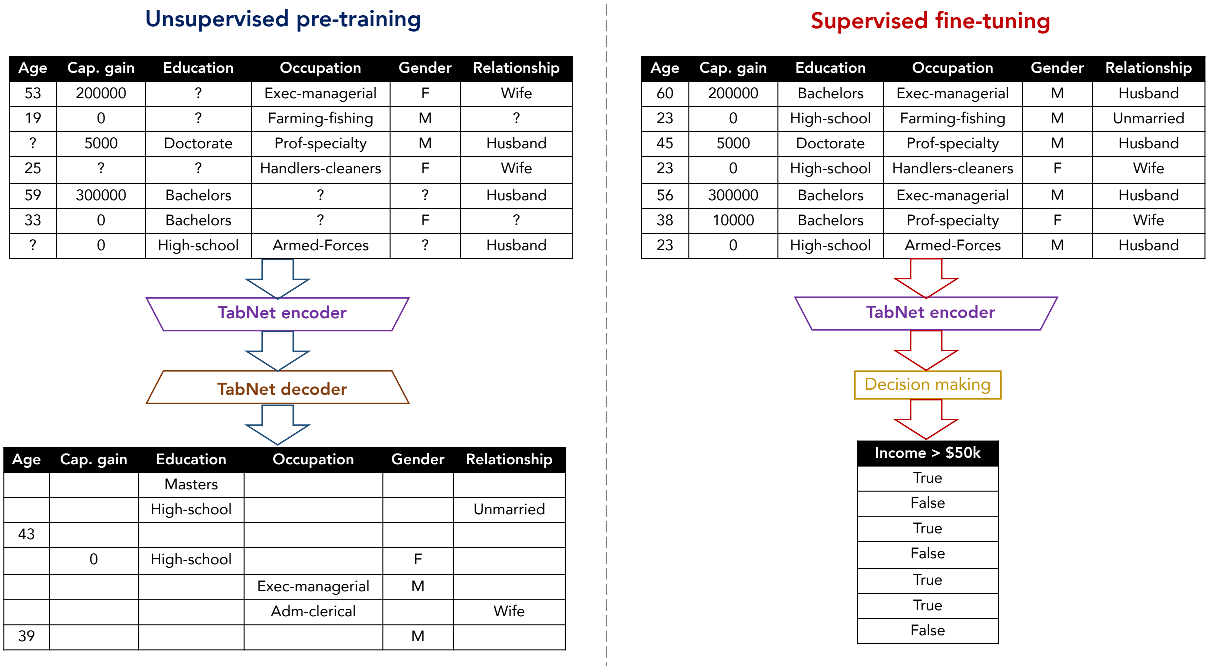
(1) Unlike tree-based methods, *TabNet inputs raw tabular data without any feature preprocessing* and is *trained using gradient descent-based optimization* to learn flexible representations and enable flexible integration into end-to-end learning.
(2) *TabNet uses sequential attention to choose which features to reason from at each decision step*, enabling interpretability and better learning as the learning capacity is used for the most salient features (see Fig. 1). This feature selection is instance-wise, e.g. it can be different for each input, and unlike other instance-wise feature selection methods like [6] or [61], TabNet employs a *single deep learning architecture with end-to-end learning*.
(3) We show that the above design choices lead to two valuable properties: (1) *TabNet outperforms or is on par with other tabular learning models* on various datasets for classification and regression problems from different domains; and (2) *TabNet enables*

[1] As it corresponds to a combination of any unrelated categorical and numerical feature.



age	workclass	education	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country
39	Private	Prof-school	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Islander	Male	0	2415	67	India

Figure 1: TabNet's sparse feature selection exemplified for Adult Census Income prediction [14]. Sparse feature selection enables interpretability and better learning as the capacity is used for the most salient features. TabNet employs multiple decision blocks that focus on processing a subset of input features for reasoning. Feature selection is based on feedback flowing from the preceding decision step. Two decision blocks shown as examples process features that are related to professional occupation and investments, respectively, in order to predict the income level.



Age	Cap. gain	Education	Occupation	Gender	Relationship
53	200000	?	Exec-managerial	F	Wife
19	0	?	Farming-fishing	M	?
?	5000	Doctorate	Prof-specialty	M	Husband
25	?	?	Handlers-cleaners	F	Wife
59	300000	Bachelors	?	?	Husband
33	0	Bachelors	?	F	?
?	0	High-school	Armed-Forces	?	Husband

Age	Cap. gain	Education	Occupation	Gender	Relationship
		Masters			
		High-school			Unmarried
43					
	0	High-school		F	
			Exec-managerial	M	
			Adm-clerical		Wife
39				M	

Age	Cap. gain	Education	Occupation	Gender	Relationship
60	200000	Bachelors	Exec-managerial	M	Husband
23	0	High-school	Farming-fishing	M	Unmarried
45	5000	Doctorate	Prof-specialty	M	Husband
23	0	High-school	Handlers-cleaners	F	Wife
56	300000	Bachelors	Exec-managerial	M	Husband
38	10000	Bachelors	Prof-specialty	F	Wife
23	0	High-school	Armed-Forces	M	Husband

Income > $50k
True
False
True
False
True
True
False

Figure 2: Self-supervised tabular learning. Real-world tabular datasets have interdependent feature columns, e.g., the education level can be guessed from the occupation, or the gender can be guessed from the relationship. Unsupervised representation learning by masked self-supervised learning results in an improved encoder model for the supervised learning task.

two kinds of interpretability: local interpretability that visualizes the importance of input features and how they are combined, and global interpretability which quantifies the contribution of each input feature to the trained model.

(4) Finally, we show that our canonical DNN design achieves significant performance improvements by using unsupervised pre-training to predict masked features (see Fig. 2). *Our work is the first demonstration of self-supervised learning for tabular data.*

2 RELATED WORK

Feature selection: Feature selection in machine learning broadly refers to judiciously picking a subset of features based on their usefulness for prediction. Commonly-used techniques such as forward selection and LASSO regularization [20] attribute feature importance based on the entire training data set, and are referred as *global* methods. *Instance-wise* feature selection refers to picking features individually for each input, studied in [6] by training an explainer model to maximize the mutual information between the selected features and the response variable, and in [61] by using an

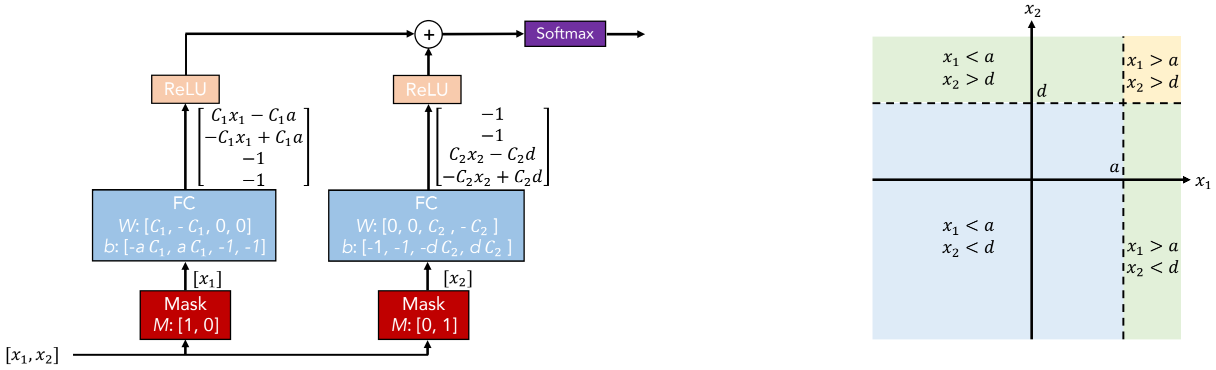


Figure 3: Illustration of decision tree-like classification using conventional DNN blocks (left) and the corresponding decision manifold (right). Relevant features are selected by using multiplicative sparse masks on inputs. The selected features are linearly transformed, and after a bias addition (to represent boundaries) ReLU performs region selection by zeroing the regions that are on the negative side of the boundary. Aggregation of multiple regions is based on addition. As C_1 and C_2 get larger, the decision boundary gets sharper due to the softmax.

actor-critic framework to mimic a baseline while optimizing the feature selection. Unlike these, TabNet employs *soft feature selection with controllable sparsity in end-to-end learning* – a single model jointly performs feature selection and output mapping, resulting in superior performance with compact representations.

Tree-based learning: Tree-based models are the most common approaches for tabular data learning. The prominent strength of tree-based models is their efficacy in picking global features with the most statistical information gain [18]. To improve the performance of standard tree-based models by reducing the model variance, one common approach is ensembling. Among ensembling methods, random forests [23] use random subsets of data with randomly selected features to grow many trees. XGBoost [7] and LightGBM [30] are the two recent ensemble decision tree approaches that dominate most of the recent data science competitions. Our experimental results for various datasets show that tree-based models can be outperformed when the representation capacity is improved with deep learning while retaining their feature selecting property.

Integration of DNNs into decision trees: Representing decision trees with canonical DNN building blocks as in [26] yields redundancy in representation and inefficient learning. Soft (neural) decision trees [33, 58] are proposed with differentiable decision functions, instead of non-differentiable axis-aligned splits. However, abandoning trees loses their automatic feature selection ability which is important for tabular data. In [60], a soft binning function is proposed to simulate decision trees in DNNs, which needs to enumerate all possible decisions and is inefficient. [31] proposes a DNN architecture by explicitly leveraging expressive feature combinations, however, learning is based on transferring knowledge from a gradient-boosted decision tree and limited performance improvements are observed. [53] proposes a DNN architecture by adaptively growing from primitive blocks while representation learning into edges, routing functions and leaf nodes of a decision tree. TabNet differs from these methods as it embeds the soft feature selection ability with controllable sparsity via sequential attention.

Attentive table-to-text models: Table-to-text models extract textual information from tabular data, for which recent works [3, 35] propose a sequential mechanism for field-level attention. Unlike these, we demonstrate the application of sequential attention for supervised or self-supervised learning instead of mapping tabular data to a different data type.

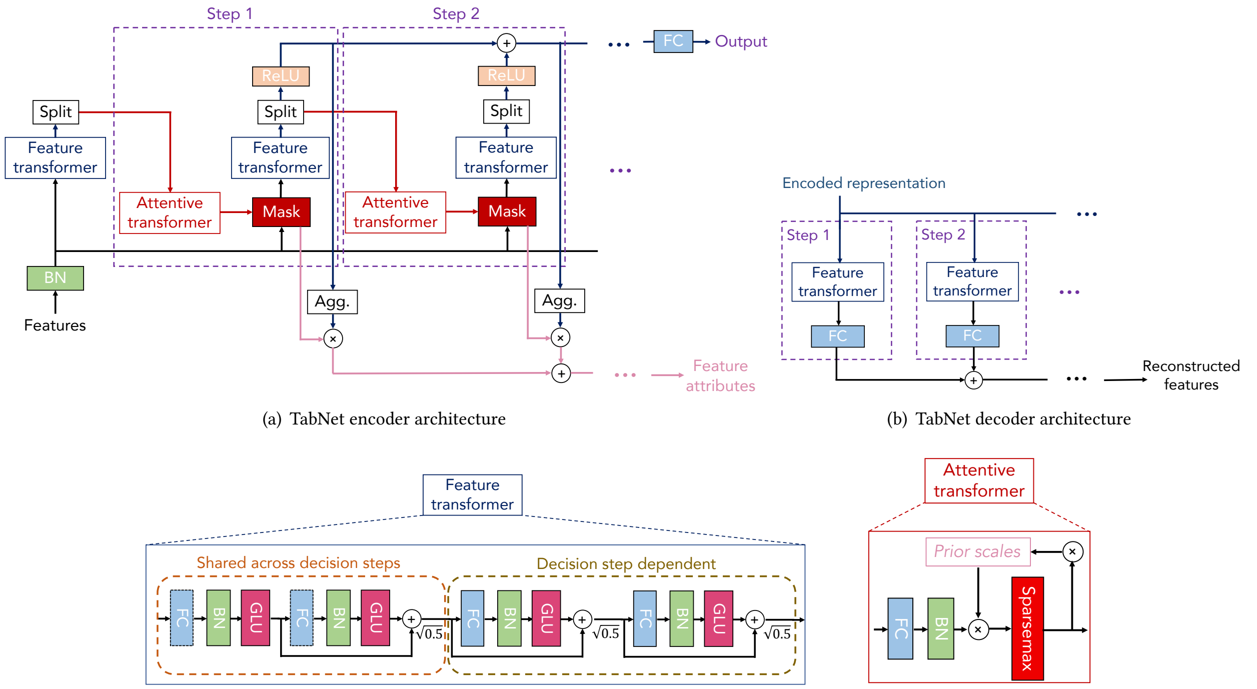
Self-supervised learning: Unsupervised representation learning is shown to benefit the supervised learning task especially in small data regime [47]. Recent work for language [13] and image [55] data has shown significant advances – specifically careful choice of the unsupervised learning objective (masked input prediction) and attention-based deep learning architecture is important.

3 TABNET FOR TABULAR LEARNING

Decision trees are successful for learning from real-world tabular datasets. However, even conventional DNN building blocks can be used to implement decision tree-like output manifold – see Fig. 3) for an example. In such a design, individual feature selection is key to obtain decision boundaries in hyperplane form. This idea can be generalized for a linear combination of features where constituent coefficients determine the proportion of each feature. TabNet is based on such a tree-like functionality. We show that it outperforms decision trees while reaping many of their benefits by careful design which: (i) uses sparse instance-wise feature selection learned based on the training dataset; (ii) constructs a sequential multi-step architecture, where each decision step can contribute to a portion of the decision that is based on the selected features; (iii) improves the learning capacity by non-linear processing of the selected features; and (iv) mimics an ensemble via higher dimensions and more steps.

Fig. 4 shows the TabNet architecture for encoding tabular data. Tabular data are comprised of numerical and categorical features. We use the raw numerical features and consider mapping of categorical features with trainable embeddings[2]. We do not consider any global normalization features, but merely apply batch normalization (BN). We pass the same D-dimensional features $\mathbf{f} \in \Re^{B \times D}$ to each decision step, where B is the batch size. TabNet's encoding is based on sequential multi-step processing with N_{steps} decision steps. The i^{th} step inputs the processed information from the $(i-1)^{th}$ step

[2] E.g., the three possible categories A, B and C for a particular feature can be learned to be mapped to scalars 0.4, 0.1, and -0.2.



(a) TabNet encoder architecture

(b) TabNet decoder architecture

(c) Feature transformer

(d) Attentive transformer

Figure 4: (a) TabNet encoder for classification or regression, composed of a feature transformer, an attentive transformer and feature masking at each decision step. A split block divides the processed representation into two, to be used by the attentive transformer of the subsequent step as well as for constructing the overall output. At each decision step, the feature selection mask can provide interpretable information about the model's functionality, and the masks can be aggregated to obtain global feature important attribution. (b) TabNet decoder, composed of a feature transformer block at each step. (c) A feature transformer block example – 4-layer network is shown, where 2 of the blocks are shared across all decision steps and 2 are decision step-dependent. Each layer is composed of a fully-connected (FC) layer, BN and GLU nonlinearity. (d) An attentive transformer block example – a single layer mapping is modulated with a prior scale information which aggregates how much each feature has been used before the current decision step. Normalization of the coefficients is done using sparsemax [37] for sparse selection of the most salient features at each decision step.

to decide which features to use and outputs the processed feature representation to be aggregated into the overall decision. The idea of top-down attention in the sequential form is inspired by its applications in processing visual and language data (e.g. in visual question answering [25]) and reinforcement learning [40] while searching for a small subset of relevant information in high dimensional input. Ablation studies in the Appendix focus on the impact of various design choices which are explained next. Guidelines on selection of the important hyperparameters are also provided in the Appendix.

Feature selection: We employ a learnable mask $\mathbf{M[i]} \in \Re^{B \times D}$ for soft selection of the salient features. Through sparse selection of the most salient features, the learning capacity of a decision step is not wasted on irrelevant features, and thus the model becomes more parameter efficient. The masking is in multiplicative form, $\mathbf{M[i]} \cdot \mathbf{f}$. We use an attentive transformer (see Fig. 4) to obtain the masks using the processed features from the preceding step, $\mathbf{a[i-1]}$:

$$\mathbf{M[i]} = \text{sparsemax}(\mathbf{P[i-1]} \cdot \text{h}_i(\mathbf{a[i-1]})). \qquad (1)$$

Sparsemax normalization [37] encourages sparsity by mapping the Euclidean projection onto the probabilistic simplex, which is observed to be superior in performance and aligned with the goal of sparse feature selection for most real-world datasets. Note that Eq. 1 ensures $\sum_{j=1}^{D} \mathbf{M[i]_{b,j}} = 1$. h_i is a trainable function, shown in Fig. 4 using a FC layer, followed by BN. $\mathbf{P[i]}$ is the prior scale term, denoting how much a particular feature has been used previously:

$$\mathbf{P[i]} = \prod\nolimits_{j=1}^{i} (\gamma - \mathbf{M[j]}), \qquad (2)$$

where γ is a relaxation parameter – when $\gamma = 1$, a feature is enforced to be used only at one decision step and as γ increases, more flexibility is provided to use a feature at multiple decision steps. $\mathbf{P[0]}$ is initialized as all ones, $\mathbf{1}^{B \times D}$, without any prior on the masked features. If some features are unused (as in self-supervised learning), corresponding $\mathbf{P[0]}$ entries are made 0 to help model's learning. To further control the sparsity of the selected features, we propose sparsity regularization in the form of entropy [19]:

$$L_{sparse} = \sum\nolimits_{i=1}^{N_{steps}} \sum\nolimits_{b=1}^{B} \sum\nolimits_{j=1}^{D} \frac{-\mathbf{M_{b,j}[i]}}{N_{steps} \cdot B} \log(\mathbf{M_{b,j}[i]} + \epsilon),$$

where ϵ is a small number for numerical stability. We add the sparsity regularization to the overall loss, with a coefficient λ_{sparse}.

Sparsity may provide a favorable inductive bias for convergence to higher accuracy for datasets where most features are redundant.

Feature processing: We process the filtered features using a feature transformer (see Fig. 4) and then split for the decision step output and information for the subsequent step, $[\mathbf{d}[\mathbf{i}], \mathbf{a}[\mathbf{i}]] = \mathrm{f}_i(\mathbf{M}[\mathbf{i}]\cdot\mathbf{f})$, where $\mathbf{d}[\mathbf{i}] \in \Re^{B \times N_d}$ and $\mathbf{a}[\mathbf{i}] \in \Re^{B \times N_a}$. For parameter-efficient and robust learning with high capacity, a feature transformer should comprise layers that are shared across all decision steps (as the same features are input across different decision steps), as well as decision step-dependent layers. Fig. 4 shows the implementation as concatenation of two shared layers and two decision step-dependent layers. Each FC layer is followed by BN and gated linear unit (GLU) nonlinearity [12][3], eventually connected to a normalized residual connection with normalization. Normalization with $\sqrt{0.5}$ helps to stabilize learning by ensuring that the variance throughout the network does not change dramatically [15]. For faster training, we aim for large batch sizes. To improve performance with large batch sizes, all BN operations, except the one applied to the input features, are implemented in ghost BN [24] form, with a virtual batch size B_V and momentum m_B. For the input features, we observe the benefit of low-variance averaging and hence avoid ghost BN. Finally, inspired by decision-tree like aggregation as in Fig. 3, we construct the overall decision embedding as $\mathbf{d_{out}} = \sum_{i=1}^{N_{steps}} \mathrm{ReLU}(\mathbf{d}[\mathbf{i}])$. We apply a linear mapping $\mathbf{W_{final}}\mathbf{d_{out}}$ to get the output mapping. For discrete outputs, we additionally employ softmax during training (and argmax during inference).

4 TABULAR SELF-SUPERVISED LEARNING

Decoding tabular features: We propose a decoder architecture to reconstruct tabular features from the encoded representations, obtained from the TabNet encoder. The decoder is composed of feature transformer blocks, followed by FC layers at each decision step. The outputs are summed to obtain the reconstructed features.[4]

Self-supervised objective: We propose the task of prediction of missing feature columns from the others. Consider a binary mask $\mathbf{S} \in \{0,1\}^{B \times D}$. The TabNet encoder inputs $(\mathbf{1}-\mathbf{S})\cdot\hat{\mathbf{f}}$ and the TabNet decoder outputs the reconstructed features, $\mathbf{S}\cdot\hat{\mathbf{f}}$. We initialize $\mathbf{P}[\mathbf{0}] = (\mathbf{1}-\mathbf{S})$ in the encoder so that the model emphasizes merely on the known features, and the decoder's last FC layer is multiplied with $\mathbf{S}$ to merely output the unknown features. We consider the reconstruction loss to optimize in self-supervised phase:

$$\sum_{b=1}^{B}\sum_{j=1}^{D}\left|(\hat{\mathbf{f}}_{\mathbf{b,j}}-\mathbf{f}_{\mathbf{b,j}})\cdot\mathbf{S}_{\mathbf{b,j}}\Big/\sqrt{\sum_{b=1}^{B}(\mathbf{f}_{\mathbf{b,j}}-1/B\sum_{b=1}^{B}\mathbf{f}_{\mathbf{b,j}})^2}\right|^2 .$$

Normalization with the population standard deviation of the ground truth data is important, as the features may have very different ranges. We sample the entries $\mathbf{S_{b,j}}$ independently from a Bernolli distribution with parameter p_s, at each iteration.

5 EXPERIMENTS

We study TabNet in wide range of problems, that contain regression or classification tasks, *particularly with published benchmarks*. For all datasets, categorical inputs are mapped to a single-dimensional trainable scalar with a learnable embedding[5] and numerical columns are input without and preprocessing.[6] We use standard classification (softmax cross entropy) and regression (mean squared error) loss functions and we train until convergence. Hyperparameters of the TabNet models are optimized on a validation set and listed in Appendix. TabNet performance is not very sensitive to most hyperparameters as shown with ablation studies in Appendix. In all of the experiments where we cite results from other papers, we use the same training, validation and testing data split with the original work. Adam optimization algorithm [32] and Glorot uniform initialization are used for training of all models. An open-source implementation can be found on https://github.com/google-research/google-research/tree/master/tabnet.

5.1 Instance-wise feature selection

Selection of the most salient features can be crucial for high performance, especially for small datasets. We consider the 6 synthetic tabular datasets from [6] (consisting 10k training samples). The synthetic datasets are constructed in such a way that only a subset of the features determine the output. For Syn1, Syn2 and Syn3 datasets, the 'salient' features are the same for all instances, so that an accurate global feature selection mechanism should be optimal. E.g., the ground truth output of the Syn2 dataset only depends on features X_3-X_6. For Syn4, Syn5 and Syn6 datasets, the salient features are instance dependent. E.g., for Syn4 dataset, X_{11} is the indicator, and the ground truth output depends on either X_1-X_2 or X_3-X_6 depending on the value of X_{11}. This instance dependence makes global feature selection suboptimal, as the globally-salient features would be redundant for some instances.

Table 1 shows the performance of TabNet encoder vs. other techniques, including no selection, using only globally-salient features, Tree Ensembles [16], LASSO regularization, L2X [6] and INVASE [61]. We observe that TabNet outperforms all other methods and is on par with INVASE. For Syn1, Syn2 and Syn3 datasets, we observe that the TabNet performance is very close to global feature selection. For Syn4, Syn5 and Syn6 datasets, we observe that TabNet improves global feature selection, which would contain redundant features. (Feature selection is visualized in Sec. 5.3.) All other methods utilize a predictive model with 43k parameters, and the total number of trainable parameters is 101k for INVASE due to the two other networks in the actor-critic framework. On the other hand, TabNet is a single DNN architecture, and its model size is 26k for Syn1-Syn3 datasets and 31k for Syn4-Syn6 datasets. This compact end-to-end representation is one of TabNet's valuable properties.

5.2 Performance on real-world datasets

Forest Cover Type [14]: This dataset corresponds to the task of classification of forest cover type from cartographic variables. Table 2 shows that TabNet significantly outperforms ensemble tree based approaches that are known to achieve solid performance on this task [38]. In addition, we consider AutoInt[51] for this task given

[3] In GLU, first a linear mapping is applied and the dimensionality is doubled, and then second half of the output is used to determine nonlinear processing on the first half.

[4] We have also experimented sequential decoding of features with attentive transformer at each decision step, but did not observe significant benefits for the purpose of self-supervised learning and thus chose this simpler architecture.

[5] In some cases, higher dimensional embeddings may slightly improve the performance, but interpretation of individual dimensions may become challenging.

[6] Specially-designed feature engineering, e.g. logarithmic transformation of variables highly-skewed distributions, may further improve the results but we leave it out of the scope of this paper.

Table 1: Mean and std. of test area under the receiving operating characteristic curve (AUC) on 6 synthetic datasets from [6], for TabNet vs. other feature selection-based DNN models: No selection: using all features without any feature selection, Global: using only globally-salient features, Tree: Tree Ensembles [16], LASSO: LASSO-regularized model, L2X [6] and INVASE [61]. Bold numbers are the best method for each dataset.

Model	*Test AUC*					
	Syn1	Syn2	Syn3	Syn4	Syn5	Syn6
No selection	.578 ± .004	.789 ± .003	.854 ± .004	.558 ± .021	.662 ± .013	.692 ± .015
Tree	.574 ± .101	.872 ± .003	.899 ± .001	.684 ± .017	.741 ± .004	.771 ± .031
Lasso	.498 ± .006	.555 ± .061	.886 ± .003	.512 ± .031	.691 ± .024	.727 ± .025
L2X	.498 ± .005	.823 ± .029	.862 ± .009	.678 ± .024	.709 ± .008	.827 ± .017
INVASE	**.690 ± .006**	.877 ± .003	**.902 ± .003**	**.787 ± .004**	.784 ± .005	.877 ± .003
Global	.686 ± .005	.873 ± .003	.900 ± .003	.774 ± .006	.784 ± .005	.858 ± .004
TabNet	.682 ± .005	**.892 ± .004**	.897 ± .003	.776 ± .017	**.789 ± .009**	**.878 ± .004**

Table 2: Performance for Forest Cover Type dataset.

Model	*Test accuracy (%)*
XGBoost	89.34
LightGBM	89.28
CatBoost	85.14
AutoInt	90.24
AutoML Tables (2 node hours)	94.56
AutoML Tables (10 node hours)	96.67
AutoML Tables (30 node hours)	96.93
TabNet	**96.99**

its strength for problems with high feature dimensionality. AutoInt models pairwise feature interactions with an attention-based DNN [51] and significantly underperforms TabNet that employs instance-wise feature selection, and considers the interaction between different features if the model infers that it is the appropriate processing to apply. Lastly, we consider AutoML Tables [2], an automated search framework based on ensemble of models including linear feed-forward DNN, gradient boosted decision tree, AdaNet [10] and ensembles [2]. For AutoML Tables, the amount of node hours reflects the measure of the count of searched models for the ensemble and their complexity.[7] A single TabNet model without fine-grained hyperparameter search outperforms the accuracy of ensemble models with very thorough hyperparameter search.

Table 3: Performance for Poker Hand induction dataset.

Model	*Test accuracy (%)*
Decision tree	50.0
MLP	50.0
Deep neural decision tree	65.1
XGBoost	71.1
LightGBM	70.0
CatBoost	66.6
TabNet	**99.2**
Rule-based	100.0

Poker Hand [14]: This dataset corresponds to the task of classification of the poker hand from the raw suit and rank attributes of the cards. The input-output relationship is deterministic and hand-crafted rules implemented with several lines of code can get 100% accuracy. Yet, conventional DNNs, decision trees, and even their hybrid variant of deep neural decision tree models [60] severely suffer from the imbalanced data and cannot learn the required sorting and ranking operations with the raw input features [60]. Tuned XGBoost, CatBoost, and LightGBM show very slight improvements over them. On the other hand, TabNet significantly outperforms the other methods and approaches to deterministic rule accuracy, as it can perform highly-nonlinear processing with high depth, without overfitting thanks to instance-wise feature selection.

Table 4: Performance for Sarcos Robotics Arm Inverse Dynamics dataset. Three TabNet models of different sizes are considered (denoted with -S, -M and -L).

Model	*Test MSE*	*Model size*
Random forest	2.39	16.7K
Stochastic decision tree	2.11	28K
MLP	2.13	0.14M
Adaptive neural tree	1.23	0.60M
Gradient boosted tree	1.44	0.99M
TabNet-S	**1.25**	**6.3K**
TabNet-M	**0.28**	**0.59M**
TabNet-L	**0.14**	**1.75M**

Sarcos Robotics Arm Inverse Dynamics [57]: This dataset corresponds to the task of regression of inverse dynamics of seven degrees-of-freedom of an anthropomorphic robot arm. [53] shows that decent performance with a very small model is possible with a random forest, but the best performance is achieved with their adaptive neural tree, which slightly outperforms gradient boosted tree. In the very small model size regime, TabNet's performance is on par with the best proposed model with 100x more parameters. TabNet allocates its capacity to salient features, and yields a more compact model. When the model size is not constrained, TabNet achieves almost an order of magnitude lower test MSE.

Higgs Boson [14]: This dataset corresponds to the task of distinguishing between a signal process which produces Higgs bosons and a background process. Due to its much larger size (10.5M training examples), DNNs outperform decision tree variants on this task

[7] 10 node hours is well above the suggested exploration time [2] for this dataset.

Table 5: Performance on Higgs Boson dataset. Two TabNet models are considered (denoted with -S and -M).

Model	*Test accuracy (%)*	*Model size*
Sparse evolutionary MLP	**78.47**	**81K**
Gradient boosted tree-S	74.22	0.12M
Gradient boosted tree-M	75.97	0.69M
MLP	78.44	2.04M
Gradient boosted tree-L	76.98	6.96M
TabNet-S	78.25	81K
TabNet-M	**78.84**	**0.66M**

even with very large ensembles. We show that the TabNet outperforms MLPs with more compact representations. We also compare to the state-of-the-art evolutionary sparsification algorithm [39] that applies non-structured sparsity integrated into training, yielding a low number of parameters. With its compact representation TabNet yields almost similar performance to sparse evolutionary training for the same number of parameters. This sparsity learned by TabNet is structured differently from alternative approaches – it does not degrade the operational intensity of the model [59] and can efficiently utilize modern multi-core processors.[8]

Table 6: Performance for Rossmann Store Sales dataset.

Model	*Test MSE*
XGBoost	490.83
LightGBM	504.76
CatBoost	489.75
TabNet	**485.12**

Rossmann Store Sales [29]: This dataset corresponds to the task of forecasting the store sales from static and time-varying features. We observe that TabNet outperforms XGBoost, LightGBM and CatBoost that are commonly-used for such problems. The time features (e.g. day) obtain high importance, and the benefit of instance-wise feature selection is particularly observed for cases like holidays where the sales dynamics are different.

Table 7: Performance on KDD datasets.

Model	*Test accuracy (%)*			
	Appetency	*Churn*	*Upselling*	*Census*
XGBoost	**98.2**	92.7	**95.1**	**95.8**
CatBoost	**98.2**	**92.8**	**95.1**	95.7
TabNet	**98.2**	92.7	95.0	95.5

KDD datasets: Appetency, Churn and Upselling datasets are classification tasks for customer relationship management, and KDD Census Income [14] dataset is for income prediction from demographic and employment related variables. These datasets show saturated behavior in performance (even simple models yield similar results). Table 7 shows that TabNet achieves very similar or slightly worse performance than XGBoost and CatBoost, that are known to be robust as they contain high amount of ensembles.

5.3 Interpretability

The feature selection masks in TabNet can be understand selected features at each step. Such a capability is not available for conventional DNNs such as MLPs, as each subsequent layer jointly processes all features without a sparsity-controlled selection mechanism. For feature selection masks, if $\mathbf{M_{b,j}[i]} = 0$, then j^{th} feature of the b^{th} sample should have no contribution to the decision. If f_i were a linear function, the coefficient $\mathbf{M_{b,j}[i]}$ would correspond to the feature importance of $\mathbf{f_{b,j}}$. Although each decision step employs non-linear processing, their outputs are combined later in a linear way. Our goal is to quantify an aggregate feature importance in addition to analysis of each step. Combining the masks at different steps requires a coefficient that can weigh the relative importance of each step in the decision. We use $\eta_\mathbf{b}[\mathbf{i}] = \sum_{c=1}^{N_d} \mathrm{ReLU}(\mathbf{d_{b,c}[i]})$ to denote the aggregate decision contribution at i^{th} decision step for the b^{th} sample. Intuitively, if $\mathbf{d_{b,c}[i]} < 0$, then all features at i^{th} decision step should have 0 contribution to the overall decision. As its value increases, it plays a higher role in the overall linear combination. Scaling the decision mask at each decision step with $\eta_\mathbf{b}[\mathbf{i}]$, we propose the aggregate feature importance mask,

$$\mathbf{M_{agg-b,j}} = \sum_{i=1}^{N_{steps}} \eta_\mathbf{b}[\mathbf{i}]\mathbf{M_{b,j}[i]} \Big/ \sum_{j=1}^{D} \sum_{i=1}^{N_{steps}} \eta_\mathbf{b}[\mathbf{i}]\mathbf{M_{b,j}[i]}.$$

Normalization is used to ensure $\sum_{j=1}^{D} \mathbf{M_{agg-b,j}} = 1$.

Synthetic datasets: Fig. 5 shows the aggregate feature importance masks for the synthetic datasets discussed in Sec. 5.1.[9] The ground truth output of the Syn2 dataset only depends on features X_3-X_6. We observe that the aggregate masks are almost all zero for irrelevant features and they merely focus on relevant ones. For Syn4 dataset, X_{11} is the indicator, and the ground truth output depends on either X_1-X_2 or X_3-X_6 depending on the value of X_{11}. TabNet yields accurate instance-wise feature selection – it allocates a mask to focus on X_{11}, and assigns almost all-zero weights to irrelevant features (the ones other than one of the two feature groups).

Table 8: Importance ranking of features for Adult Census Income. TabNet yields feature importance rankings consistent with the well-known methods.

Feature	*SHAP*	*Skater*	*XGBoost*	*TabNet*
Age	1	1	1	1
Capital gain	3	3	4	6
Capital loss	9	9	6	4
Education	5	2	3	2
Gender	8	10	12	8
Hours per week	7	7	2	7
Marital status	2	8	10	9
Native country	11	11	9	12
Occupation	6	5	5	3
Race	12	12	11	11
Relationship	4	4	8	5
Work class	10	8	7	10

Real-world datasets: We first consider the simple real-world task

[8] Matrix sparsification techniques such as adaptive pruning [41] in TabNet could further improve the parameter-efficiency.

[9] For better illustration here, unlike Sec. 5.1, the models are trained with 10M training samples rather than 10K as we obtain sharper feature selection masks

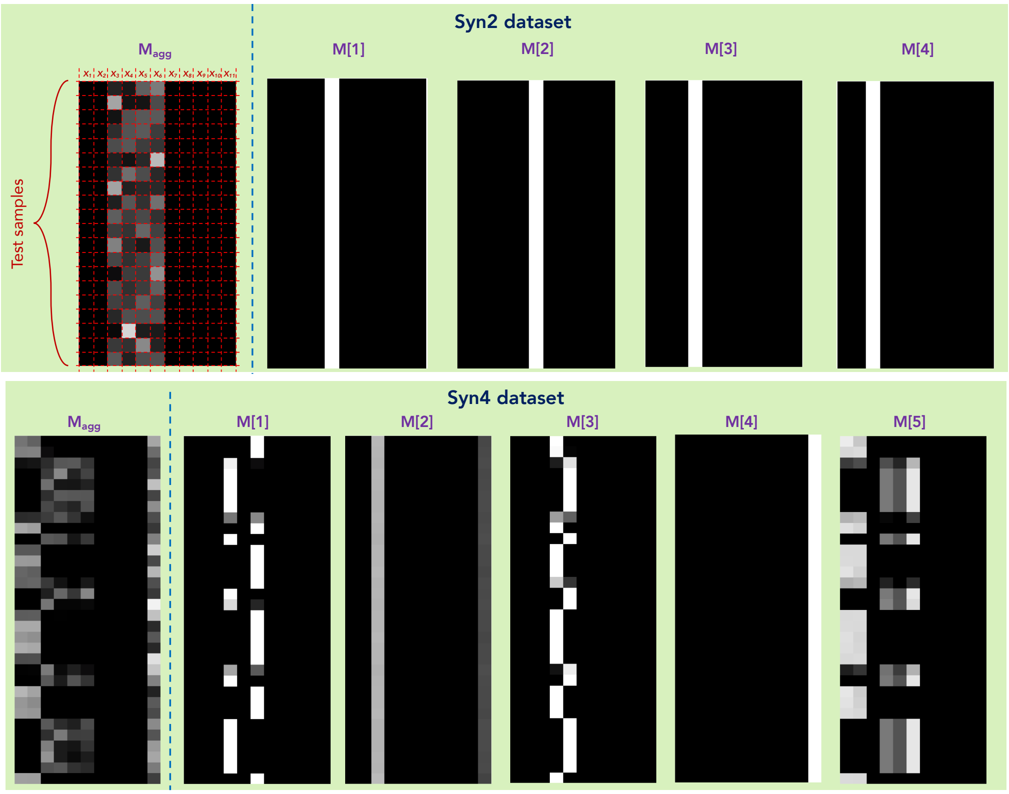


Figure 5: Feature importance masks M[i] (that indicate which features are selected at i^{th} step) and the aggregate feature importance mask M_{agg} showing the global instance-wise feature selection for Syn2 and Syn6 datasets from [6]. Brighter colors show a higher value. E.g. for Syn2 dataset, only four features (X_3-X_6) are used.

.

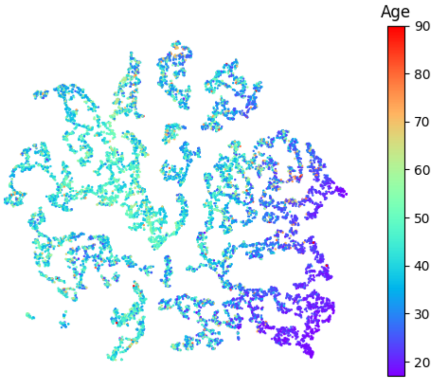


Figure 6: T-SNE of the decision manifold for Adult Census Income test samples and the impact of the top feature 'Age'.

of mushroom edibility prediction [14]. TabNet achieves 100% test accuracy on this dataset. It is indeed known [14] that "Odor" is the most discriminative feature for this task, with "Odor" feature only, model can get > 98.5% test accuracy [14]. Thus, a high feature importance is expected for it. TabNet assigns an importance score ratio of 43% for it, while other notable methods like LIME [48], Integrated Gradients [52] and DeepLift [49] assign importance score ratios of less than 30% [27].

Next, we consider Adult Census Income, where the task is to distinguish whether a personfis income is above $50,000. Table 8 shows the importance ranking of features for TabNet vs. other explainability techniques from [36] [42]. We observe the commonality of the most important features ("Age", "Capital gain/loss", "Education number", "Relationship") and the least important features ("Native country", "Race", "Gender", "Work class"). For the same problem, Fig. 6(c) shows the impact of the most important feature on the output decision by visualizing the T-SNE of the decision manifold. A clear separation between age groups is observed, as suggested by "Age" being the most important feature by TabNet.

5.4 Self-supervised learning

We study self-supervised learning on Higgs and Forest Cover Type datasets. For the pre-training task of guessing the missing columns, we use the masking parameter $p_s = 0.8$ and train for 1M iterations. We use a subset of the labeled dataset for supervised fine-tuning with a validation set to determine the number of iterations for early stopping. A large validation dataset would be unrealistic for small training datasets, so in these experiments we assume its size is equal to the training dataset. Table 9 shows that unsupervised pre-training significantly improves performance on the supervised

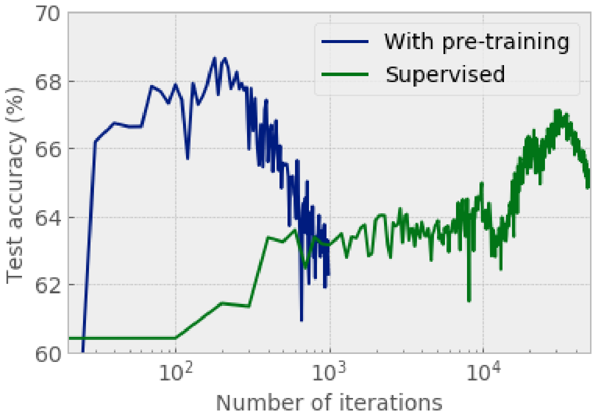


Figure 7: Convergence with unsupervised pre-training is much faster, shown for Higgs dataset with 10k samples.

Table 9: Self-supervised tabular learning results. Mean and std. of accuracy (over 15 runs) on Higgs with Tabnet-M model, varying the size of the training dataset for supervised fine-tuning.

Training dataset size	*Test accuracy (%)*	
	Supervised	*With pre-training*
1k	57.47 ± 1.78	**61.37 ± 0.88**
10k	66.66 ± 0.88	**68.06 ± 0.39**
100k	72.92 ± 0.21	**73.19 ± 0.15**

Table 10: Self-supervised tabular learning results. Mean and std. of accuracy (over 15 runs) on Forest Cover Type, varying the size of the training dataset for supervised fine-tuning.

Training dataset size	*Test accuracy (%)*	
	Supervised	*With pre-training*
1k	65.91 ± 1.02	**67.86 ± 0.63**
10k	78.85 ± 1.24	**79.22 ± 0.78**

classification task, especially in the regime where the unlabeled dataset is much larger than the labeled dataset. As exemplified in Fig. 7 the model convergence is much faster with unsupervised pre-training. Very fast convergence can be highly beneficial particularly in scenarios like continual learning and domain adaptation.

6 CONCLUSION

We have proposed TabNet, a novel deep learning architecture for tabular learning. TabNet uses a sequential attention mechanism to choose a subset of semantically meaningful features to process at each decision step. Instance-wise feature selection enables efficient learning as the model capacity is fully used for the most salient features, and also yields more interpretable decision making via visualization of selection masks. We demonstrate that TabNet outperforms previous work across tabular datasets from different domains. Lastly, we demonstrate significant benefits of unsupervised pre-training for fast adaptation and improved performance.

REFERENCES

[1] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, et al. 2015. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *arXiv:1512.02595* (2015).

[2] AutoML. 2019. AutoML Tables – Google Cloud. https://cloud.google.com/automl-tables/

[3] J. Bao, D. Tang, N. Duan, Z. Yan, M. Zhou, and T. Zhao. 2019. Text Generation From Tables. *IEEE Trans Audio, Speech, and Language Processing* 27, 2 (Feb 2019), 311–320.

[4] Yael Ben-Haim and Elad Tom-Tov. 2010. A Streaming Parallel Decision Tree Algorithm. *JMLR* 11 (March 2010), 849–872.

[5] Catboost. 2019. Benchmarks. https://github.com/catboost/benchmarks. Accessed: 2019-11-10.

[6] Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. 2018. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. *arXiv:1802.07814* (2018).

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*.

[8] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, et al. 2018. Notes from the AI Frontier. *McKinsey Global Institute* (4 2018).

[9] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. 2016. Very Deep Convolutional Networks for Natural Language Processing. *arXiv:1606.01781* (2016).

[10] Corinna Cortes, Xavi Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. 2016. AdaNet: Adaptive Structural Learning of Artificial Neural Networks. *arXiv:1607.01097* (2016).

[11] Zihang Dai, Zhilin Yang, Fan Yang, William W. Cohen, and Ruslan Salakhutdinov. 2017. Good Semi-supervised Learning that Requires a Bad GAN. *arxiv:1705.09783* (2017).

[12] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language Modeling with Gated Convolutional Networks. *arXiv:1612.08083* (2016).

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805* (2018).

[14] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[15] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. *arXiv:1705.03122* (2017).

[16] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine Learning* 63, 1 (01 Apr 2006), 3–42.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[18] K. Grabczewski and N. Jankowski. 2005. Feature selection with decision tree criterion. In *HIS*.

[19] Yves Grandvalet and Yoshua Bengio. 2004. Semi-supervised Learning by Entropy Minimization. In *NIPS*.

[20] Isabelle Guyon and André Elisseeff. 2003. An Introduction to Variable and Feature Selection. *JMLR* 3 (March 2003), 1157–1182.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv:1512.03385* (2015).

[22] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory F. Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. 2017. Deep Learning Scaling is Predictable, Empirically. *arXiv:1712.00409* (2017).

[23] Tin Kam Ho. 1998. The random subspace method for constructing decision forests. *PAMI* 20, 8 (Aug 1998), 832–844.

[24] Elad Hoffer, Itay Hubara, and Daniel Soudry. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv:1705.08741* (2017).

[25] Drew A. Hudson and Christopher D. Manning. 2018. Compositional Attention Networks for Machine Reasoning. *arXiv:1803.03067* (2018).

[26] K. D. Humbird, J. L. Peterson, and R. G. McClarren. 2018. Deep Neural Network Initialization With Decision Trees. *IEEE Trans Neural Networks and Learning Systems* (2018).

[27] Mark Ibrahim, Melissa Louie, Ceena Modarres, and John W. Paisley. 2019. Global Explanations of Neural Networks: Mapping the Landscape of Predictions. *arxiv:1902.02384* (2019).

[28] Kaggle. 2019. Historical Data Science Trends on Kaggle. https://www.kaggle.com/shivamb/data-science-trends-on-kaggle. Accessed: 2019-04-20.

[29] Kaggle. 2019. Rossmann Store Sales. https://www.kaggle.com/c/rossmann-store-sales. Accessed: 2019-11-10.

[30] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, et al. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *NIPS*.

[31] Guolin Ke, Jia Zhang, Zhenhui Xu, Jiang Bian, and Tie-Yan Liu. 2019. TabNN: A Universal Neural Network Solution for Tabular Data. https://openreview.net/forum?id=r1eJssCqY7

[32] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *ICLR*.

[33] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bul. 2015. Deep Neural Decision Forests. In *ICCV*.

[34] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent Convolutional Neural Networks for Text Classification. In *AAAI*.

[35] Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. 2017. Table-to-text Generation by Structure-aware Seq2seq Learning. *arXiv:1711.09724* (2017).

[36] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. 2018. Consistent Individualized Feature Attribution for Tree Ensembles. *arXiv:1802.03888* (2018).

[37] André F. T. Martins and Ramón Fernández Astudillo. 2016. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. *arXiv:1602.02068* (2016).

[38] Rory Mitchell, Andrey Adinets, Thejaswi Rao, and Eibe Frank. 2018. XGBoost: Scalable GPU Accelerated Learning. *arXiv:1806.11248* (2018).

[39] Decebal Mocanu, Elena Mocanu, Peter Stone, Phuong Nguyen, Madeleine Gibescu, and Antonio Liotta. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications* 9 (12 2018).

[40] Alex Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo J. Rezende. 2019. S3TA: A Soft, Spatial, Sequential, Top-Down Attention Model. https://openreview.net/forum?id=B1gJOoRcYQ

[41] Sharan Narang, Gregory F. Diamos, Shubho Sengupta, and Erich Elsen. 2017. Exploring Sparsity in Recurrent Neural Networks. *arXiv:1704.05119* (2017).

[42] Nbviewer. 2019. Notebook on Nbviewer. https://nbviewer.jupyter.org/github/dipanjanS/data_science_for_all/blob/master/tds_model_interpretation_xai/Human-interpretableMachineLearning-DS.ipynb#

[43] N. C. Oza. 2005. Online bagging and boosting. In *IEEE Trans Conference on Systems, Man and Cybernetics*.

[44] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2018. Continual Lifelong Learning with Neural Networks: A Review. *arXiv:1802.07569* (2018).

[45] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In *NIPS*.

[46] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434* (2015).

[47] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. 2007. Self-Taught Learning: Transfer Learning from Unlabeled Data. In *ICML*.

[48] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. fiWhy Should I Trust You?fi: Explaining the Predictions of Any Classifier. In *KDD*.

[49] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning Important Features Through Propagating Activation Differences. *arXiv:1704.02685* (2017).

[50] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556* (2014).

[51] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2018. AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *arxiv:1810.11921* (2018).

[52] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. *arXiv:1703.01365* (2017).

[53] Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya V. Nori. 2018. Adaptive Neural Trees. *arXiv:1807.06699* (2018).

[54] Tensorflow. 2019. Classifying Higgs boson processes in the HIGGS Data Set. https://github.com/tensorflow/models/tree/master/official/boosted_trees

[55] Trieu H. Trinh, Minh-Thang Luong, and Quoc V. Le. 2019. Selfie: Self-supervised Pretraining for Image Embedding. *arXiv:1906.02940* (2019).

[56] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, et al. 2016. WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499* (2016).

[57] Sethu Vijayakumar and Stefan Schaal. 2000. Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space. In *ICML*.

[58] Suhang Wang, Charu Aggarwal, and Huan Liu. 2017. Using a random forest to inspire a neural network and improving on it. In *SDM*.

[59] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning Structured Sparsity in Deep Neural Networks. *arXiv:1608.03665* (2016).

[60] Yongxin Yang, Irene Garcia Morillo, and Timothy M. Hospedales. 2018. Deep Neural Decision Trees. *arXiv:1806.06988* (2018).

[61] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2019. INVASE: Instance-wise Variable Selection using Neural Networks. In *ICLR*.

APPENDIX

A EXPERIMENT HYPERPARAMETERS

For all datasets, we use a pre-defined hyperparameter search space. N_d and N_a are chosen from $\{8, 16, 24, 32, 64, 128\}$, N_{steps} is chosen from $\{3, 4, 5, 6, 7, 8, 9, 10\}$, γ is chosen from $\{1.0, 1.2, 1.5, 2.0\}$, λ_{sparse} is chosen from $\{0, 0.000001, 0.0001, 0.001, 0.01, 0.1\}$, B is chosen from $\{256, 512, 1024, 2048, 4096, 8192, 16384, 32768\}$, B_V is chosen from $\{256, 512, 1024, 2048, 4096\}$ and m_B is chosen from $\{0.6, 0.7, 0.8, 0.9, 0.95, 0.98\}$. If the model size is not under the desired cutoff, we decrease the value to satisfy the size constraint. For all the comparison models, we run a hyperparameter tuning with the same number of search steps.

Synthetic: All TabNet models use N_d=N_a=16, B=3000, B_V=100, m_B=0.7. For Syn1 we use λ_{sparse}=0.02, N_{steps}=4 and γ=2.0; for Syn2 and Syn3 we use λ_{sparse}=0.01, N_{steps}=4 and γ=2.0; and for Syn4, Syn5 and Syn6 we use λ_{sparse}=0.005, N_{steps}=5 and γ=1.5. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. All models use Adam with a learning rate of 0.02 (decayed 0.7 every 200 iterations with an exponential decay) for 4k iterations. For visualizations in Sec. 5.3, we also train TabNet models with datasets of size 10M samples. For this case, we choose $N_d = N_a = 32$, λ_{sparse}=0.001, B=10000, B_V=100, m_B=0.9. Adam is used with a learning rate of 0.02 (decayed 0.9 every 2k iterations with an exponential decay) for 15k iterations. For Syn2 and Syn3, N_{steps}=4 and γ=2. For Syn4 and Syn6, N_{steps}=5 and γ=1.5.

Forest Cover Type: The dataset partition details, and the hyperparameters of XGBoost, LigthGBM, and CatBoost are from [38]. We re-optimize AutoInt hyperparameters. TabNet model uses N_d=N_a=64, λ_{sparse}=0.0001, B=16384, B_V=512, m_B=0.7, N_{steps}=5 and γ=1.5. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.02 (decayed 0.95 every 0.5k iterations with an exponential decay) for 130k iterations. For unsupervised pre-training, the decoder model uses N_d=N_a=64, B=16384, B_V=512, m_B=0.7, and N_{steps}=10. For supervised fine-tuning, we use the batch size B=B_V as the training datasets are small.

Poker Hands: We split 6k samples for validation from the training dataset, and after optimization of the hyperparameters, we retrain with the entire training dataset. Decision tree, MLP and deep neural decision tree models follow the same hyperparameters with [60]. We tune the hyperparameters of XGBoost, LigthGBM, and CatBoost. TabNet uses N_d=N_a=16, λ_{sparse}=0.000001, B=4096, B_V=1024, $m_B = 0.95$, N_{steps}=4 and γ=1.5. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.01 (decayed 0.95 every 500 iterations with an exponential decay) for 50k iterations.

Sarcos: We split 4.5k samples for validation from the training dataset, and after optimization of the hyperparameters, we retrain with the entire training dataset. All comparison models follow the hyperparameters from [53]. TabNet-S model uses N_d=N_a=8, λ_{sparse}=0.0001, B=4096, B_V=256, m_B=0.9, N_{steps}=3 and γ=1.2. Each feature transformer block uses one shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.01 (decayed 0.95 every 8k iterations with

Table 11: Ablation studies for the TabNet encoder model for the forest cover type dataset.

Ablation cases	*Test accuracy* % (difference)	*Model size*
Base ($N_d = N_a = 64$, $\gamma = 1.5$, $N_{steps} = 5$, $\lambda_{sparse} = 0.0001$, feature transformer block composed of two shared and two decision step-dependent layers, $B = 16384$)	96.99	470k
Decreasing capacity via number of units (with $N_d = N_a = 32$)	94.99 (-2.00)	129k
Decreasing capacity via number of decision steps (with $N_{steps} = 3$)	96.22 (-0.77)	328k
Increasing capacity via number of decision steps (with $N_{steps} = 9$)	95.48 (-1.51)	755k
Decreasing capacity via all-shared feature transformer blocks	96.74 (-0.25)	143k
Increasing capacity via decision step-dependent feature transformer blocks	96.76 (-0.23)	703k
Feature transformer block as a single shared layer	95.32 (-1.67)	35k
Feature transformer block as a single shared layer, with ReLU instead of GLU	93.92 (-3.07)	27k
Feature transformer block as two shared layers	96.34 (-0.66)	71k
Feature transformer block as two shared layers and 1 decision step-dependent layer	96.54 (-0.45)	271k
Feature transformer block as a single decision-step dependent layer	94.71 (-0.28)	105k
Feature transformer block as a single decision-step dependent layer, with N_d=N_a=128	96.24 (-0.75)	208k
Feature transformer block as a single decision-step dependent layer, with N_d=N_a=128 and replacing GLU with ReLU	95.67 (-1.32)	139k
Feature transformer block as a single decision-step dependent layer, with N_d=N_a=256 and replacing GLU with ReLU	96.41 (-0.58)	278k
Reducing the impact of prior scale (with $\gamma = 3.0$)	96.49 (-0.50)	470k
Increasing the impact of prior scale (with $\gamma = 1.0$)	96.67 (-0.32)	470k
No sparsity regularization (with $\lambda_{sparse} = 0$)	96.50 (-0.49)	470k
High sparsity regularization (with $\lambda_{sparse} = 0.01$)	93.87 (-3.12)	470k
Small batch size ($B = 4096$)	96.42 (-0.57)	470k

an exponential decay) for 600k iterations. TabNet-M model uses N_d=N_a=64, λ_{sparse}=0.0001, B=4096, B_V=128, m_B=0.8, N_{steps}=7 and γ=1.5. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.01 (decayed 0.95 every 8k iterations with an exponential decay) for 600k iterations. The TabNet-L model uses N_d=N_a=128, λ_{sparse}=0.0001, B=4096, B_V=128, m_B=0.8, N_{steps}=5 and γ=1.5. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.02 (decayed 0.9 every 8k iterations with an exponential decay) for 600k iterations.

Higgs: We split 500k samples for validation from the training dataset, and after optimization of the hyperparameters, we retrain with the entire training dataset. MLP models are from [39]. For gradient boosted trees [54], we tune the learning rate and depth – the gradient boosted tree-S, -M, and -L models use 50, 300 and 3000 trees respectively. TabNet-S model uses N_d=24, N_a=26, λ_{sparse}=0.000001, B=16384, B_V=512, m_B=0.6, N_{steps}=5 and γ=1.5. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.02 (decayed 0.9 every 20k iterations with an exponential decay) for 870k iterations. TabNet-M model uses N_d=96, N_a=32, λ_{sparse}=0.000001, B=8192, B_V=256, m_B=0.9, N_{steps}=8 and γ=2.0. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.025 (decayed 0.9 every 10k iterations with an exponential decay) for 370k iterations. For unsupervised pre-training, the decoder model uses N_d=N_a=128, B=8192, B_V=256, m_B=0.9, and N_{steps}=20. For supervised fine-tuning, we use the batch size B=B_V as the training datasets are small.

Rossmann: We use the same preprocessing and data split with [5] – data from 2014 is used for training and validation, whereas 2015 is used for testing. We split 100k samples for validation from the training dataset, and after optimization of the hyperparameters, we retrain with the entire training dataset. The performance of the comparison models are from [5]. TabNet model uses N_d=N_a=32, λ_{sparse}=0.001, B=4096, B_V=512, m_B=0.8, N_{steps}=5 and γ=1.2. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.002 (decayed 0.95 every 2000 iterations with an exponential decay) for 15k iterations.

KDD: For Appetency, Churn and Upselling datasets, we apply the similar preprocessing and split as [45]. The performance of the comparison models are from [45]. TabNet models use N_d=N_a=32, λ_{sparse}=0.001, B=8192, B_V=256, m_B=0.9, N_{steps}=7 and γ=1.2. Each feature transformer block uses two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.01 (decayed 0.9 every 1000 iterations with an exponential decay) for 10k iterations. For Census Income, the dataset and comparison model specifications follow [43]. TabNet model uses N_d=N_a=48, λ_{sparse}=0.001, B=8192, B_V=256, m_B=0.9, N_{steps}=5 and γ=1.5. Feature transformers use two shared and two decision step-dependent FC layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.02 (decayed 0.7 every 2000 iterations with an exponential decay) for 4k iterations.

Mushroom edibility: TabNet model uses N_d=N_a=8, λ_{sparse}=0.001, B=2048, B_V=128, m_B=0.9, N_{steps}=3 and γ=1.5. Feature transformers use two shared and two decision step-dependent FC layer, ghost

BN and GLU blocks. Adam is used with a learning rate of 0.01 (decayed 0.8 every 400 iterations with an exponential decay) for 10k iterations.

Adult Census Income: TabNet model uses N_d=N_a=16, λ_{sparse} = 0.0001, B=4096, B_V=128, m_B=0.98, N_{steps}=5 and γ=1.5. Feature transformers use two shared and two decision step-dependent layer, ghost BN and GLU blocks. Adam is used with a learning rate of 0.02 (decayed 0.4 every 2.5k iterations with an exponential decay) for 7.7k iterations. 85.7% test accuracy is achieved.

B ABLATION STUDIES

Table 11 shows the impact of ablation cases. For all cases, the number of iterations is optimized on the validation set.

Obtaining high performance necessitates appropriately-adjusted model capacity based on the characteristics of the dataset. Decreasing the number of units N_d, N_a or the number of decision steps N_{steps} are efficient ways of gradually decreasing the capacity without significant degradation in performance. On the other hand, increasing these parameters beyond some value causes optimization issues and do not yield performance benefits. Replacing the feature transformer block with a simpler alternative, such as a single shared layer, can still give strong performance while yielding a very compact model architecture. This shows the importance of the inductive bias introduced with feature selection and sequential attention. To push the performance, increasing the depth of the feature transformer is an effective approach. While increasing the depth, parameter sharing between feature transformer blocks across decision steps is an efficient way to decrease model size without degradation in performance. We indeed observe the benefit of partial parameter sharing, compared to fully decision step-dependent blocks or fully shared blocks. We also observe the empirical benefit of GLU, compared to conventional nonlinearities like ReLU.

The strength of sparse feature selection depends on the two parameters we introduce: γ and λ_{sparse}. We show that optimal choice of these two is important for performance. A γ close to 1, or a high λ_{sparse} may yield too tight constraints on the strength of sparsity and may hurt performance. On the other hand, there is still the benefit of a sufficient low γ and sufficiently high λ_{sparse}, to aid learning of the model via a favorable inductive bias.

Lastly, given the fixed model architecture, we show the benefit of large-batch training, enabled by ghost BN [24]. The optimal batch size for TabNet seems considerably higher than the conventional batch sizes used for other data types, such as images or speech.

C GUIDELINES FOR HYPERPARAMETERS

We consider datasets ranging from ~10K to ~10M samples, with varying degrees of fitting difficulty. TabNet obtains high performance on all with a few general principles on hyperparameters:

- For most datasets, $N_{steps} \in [3, 10]$ is optimal. Typically, when there are more information-bearing features, the optimal value of N_{steps} tends to be higher. On the other hand, increasing it beyond some value may adversely affect training dynamics as some paths in the network becomes deeper and there are more potentially-problematic ill-conditioned matrices. A very high value of N_{steps} may suffer from overfitting and yield poor generalization.
- Adjustment of N_d and N_a is an efficient way of obtaining a trade-off between performance and complexity. $N_d = N_a$ is a reasonable choice for most datasets. A very high value of N_d and N_a may suffer from overfitting and yield poor generalization.
- An optimal choice of γ can have a major role on the performance. Typically a larger N_{steps} value favors for a larger γ.
- A large batch size is beneficial – if the memory constraints permit, as large as 1-10 % of the total training dataset size can help performance. The virtual batch size is typically much smaller.
- Initially large learning rate is important, which should be gradually decayed until convergence.