TabNet: Attentive Interpretable Tabular Learning

Sercan Ö. Arık, Tomas Pfister
Google Cloud AI
Sunnyvale, CA
soarik@google.com, 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 variants on a wide range of non-performance-saturated tabular datasets and yields interpretable feature attributions plus insights into its global behavior. Finally, we demonstrate self-supervised learning for tabular data, significantly improving performance when unlabeled data is abundant.

Introduction

Deep neural networks (DNNs) have shown notable success with images (He et al. 2015), text (Lai et al. 2015) and audio (Amodei et al. 2015). For these, canonical architectures that efficiently encode the raw data into meaningful representations, fuel the rapid progress. One data type that has yet to see such success with a canonical architecture is tabular data.

Despite being the most common data type in real-world AI (as it is comprised of any categorical and numerical features), (Chui et al. 2018), deep learning for tabular data remains under-explored, with variants of ensemble decision trees (DTs) still dominating most applications (Kaggle 2019a). Why? First, because DT-based approaches have certain benefits: (i) they are representionally efficient 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 popular post-hoc explainability methods for their ensemble form, e.g. (Lundberg, Erion, and Lee 2018) – this is an important concern in many real-world applications; (iii) they are fast to train. Second, because previously-proposed DNN architectures are not well-suited for tabular data: e.g. 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 (Goodfellow, Bengio, and Courville 2016; Shavitt and Segal 2018; Xu et al. 2019).

Why is deep learning worth exploring for tabular data? One obvious motivation is expected performance improvements particularly for large datasets (Hestness et al. 2017). In addition, unlike tree learning, DNNs enable gradient descent-based end-to-end learning for tabular data which can have a multitude of benefits: (i) efficiently encoding multiple data types like images along with tabular data; (ii) alleviating the need for feature engineering, which is currently a key aspect in tree-based tabular data learning methods; (iii) learning from streaming data and perhaps most importantly (iv) end-to-end models allow representation learning which enables many valuable application scenarios including data-efficient domain adaptation (Goodfellow, Bengio, and Courville 2016), generative modeling (Radford, Metz, and Chintala 2015) and semi-supervised learning (Dai et al. 2017).

We propose a new canonical DNN architecture for tabular data, TabNet. The main contributions are summarized as:

1. *TabNet inputs raw tabular data without any preprocessing* and is *trained using gradient descent-based optimization*, enabling 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 (Chen et al. 2018) or (Yoon, Jordon, and van der Schaar 2019), TabNet employs a *single deep learning architecture for feature selection and reasoning*.
3. Above design choices lead to two valuable properties: (i) *TabNet outperforms or is on par with other tabular learning models* on various datasets for classification and regression problems from different domains; and (ii) *TabNet enables two kinds of interpretability*: local interpretability that visualizes the importance of features and how they are combined, and global interpretability which quantifies the contribution of each feature to the trained model.
4. Finally, *for the first time for tabular data*, we show significant performance improvements by using unsupervised pre-training to predict masked features (see Fig. 2).

Related Work

Feature selection: Feature selection broadly refers to judiciously picking a subset of features based on their usefulness for prediction. Commonly-used techniques such as for-

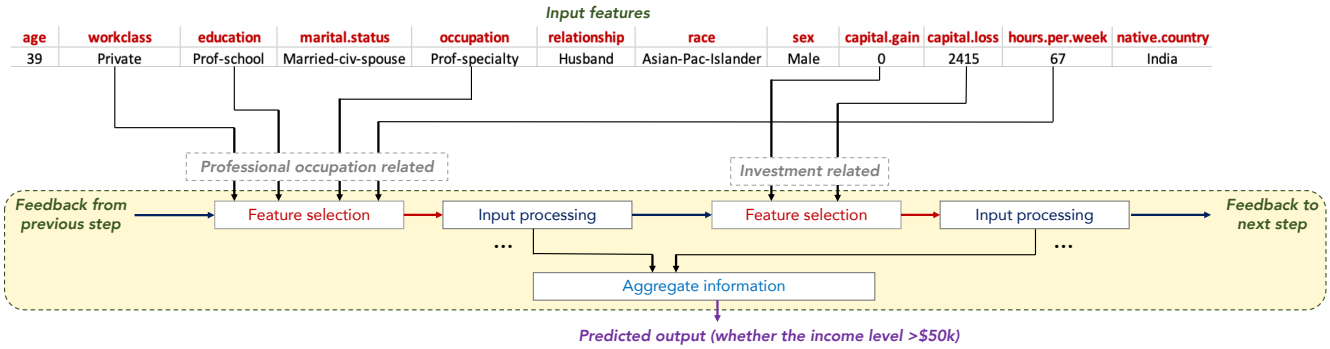


Figure 1: TabNet's sparse feature selection exemplified for Adult Census Income prediction (Dua and Graff 2017). Sparse feature selection enables interpretability and better learning as the capacity is used for the most salient features. TabNet employs multiple decision blocks that focus on processing a subset of input features for reasoning. Two decision blocks shown as examples process features that are related to professional occupation and investments, respectively, in order to predict the income level.

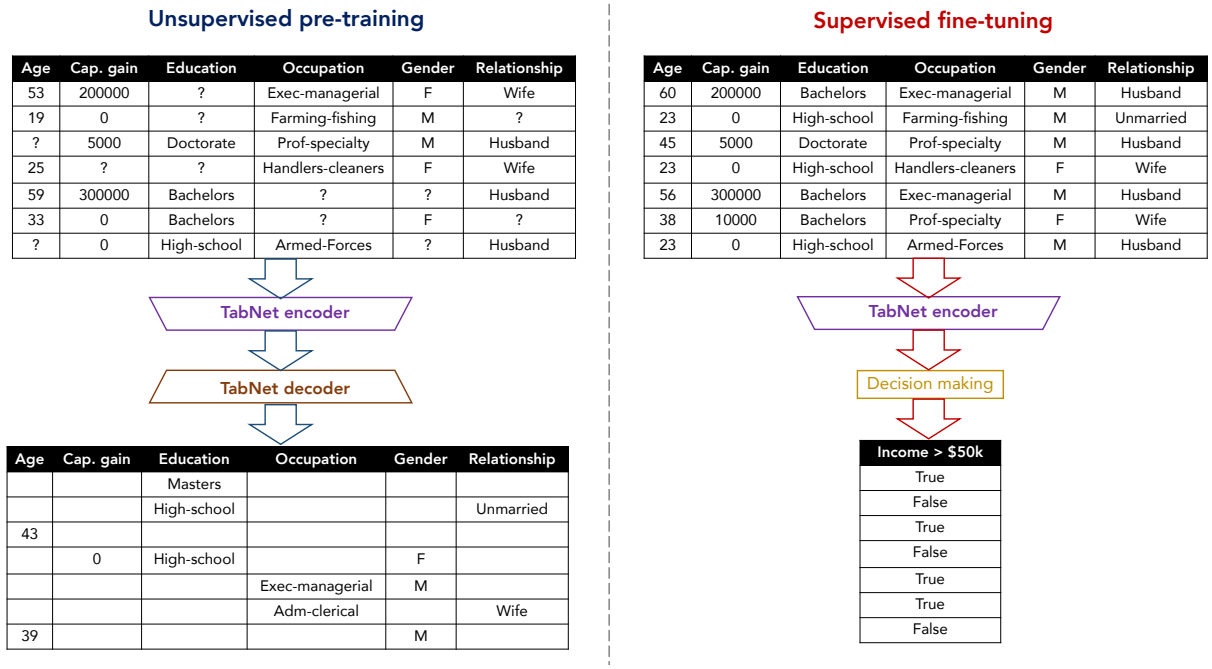


Figure 2: Self-supervised tabular learning. Real-world tabular datasets have interdependent feature columns, e.g., the education level can be guessed from the occupation, or the gender can be guessed from the relationship. Unsupervised representation learning by masked self-supervised learning results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

ward selection and Lasso regularization (Guyon and Elisseeff 2003) attribute feature importance based on the entire training data, and are referred as *global* methods. *Instance-wise* feature selection refers to picking features individually for each input, studied in (Chen et al. 2018) with an explainer model to maximize the mutual information between the selected features and the response variable, and in (Yoon, Jordon, and van der Schaar 2019) by using an actor-critic framework to mimic a baseline while optimizing the 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: DTs are commonly-used for tabular data learning. Their prominent strength is efficient picking of global features with the most statistical information gain (Grabczewski and Jankowski 2005). To improve the performance of standard DTs, one common approach is ensembling to reduce variance. Among ensembling methods, random forests (Ho 1998) use random subsets of data with randomly selected features to grow many trees. XGBoost (Chen and Guestrin 2016) and LightGBM (Ke et al. 2017) are the two recent ensemble DT approaches that dominate most of the recent data science competitions. Our experimental results

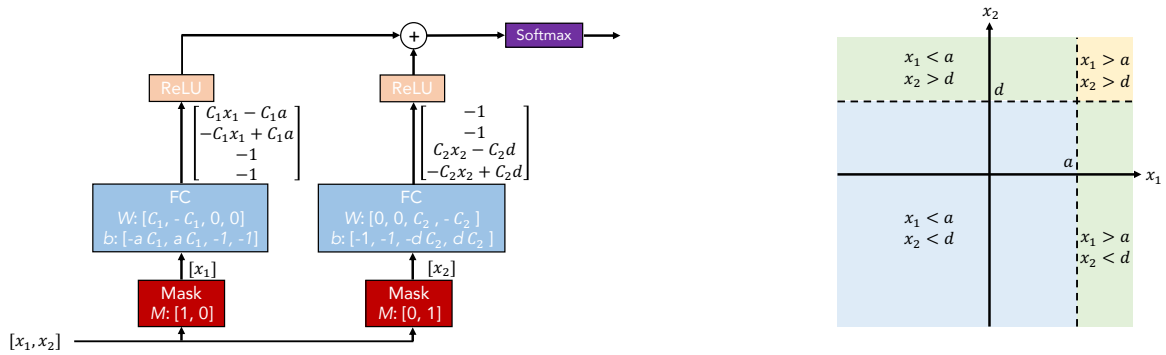


Figure 3: Illustration of DT-like classification using conventional DNN blocks (left) and the corresponding decision manifold (right). Relevant features are selected by using multiplicative sparse masks on inputs. The selected features are linearly transformed, and after a bias addition (to represent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

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 DTs: Representing DTs with DNN building blocks as in (Humbird, Peterson, and McClarren 2018) yields redundancy in representation and inefficient learning. Soft (neural) DTs (Wang, Aggarwal, and Liu 2017; Kontschieder et al. 2015) use differentiable decision functions, instead of non-differentiable axis-aligned splits. However, losing automatic feature selection often degrades performance. In (Yang, Morillo, and Hospedales 2018), a soft binning function is proposed to simulate DTs in DNNs, by inefficiently enumerating of all possible decisions. (Ke et al. 2019) proposes a DNN architecture by explicitly leveraging expressive feature combinations, however, learning is based on transferring knowledge from gradient-boosted DT. (Tanno et al. 2018) proposes a DNN architecture by adaptively growing from primitive blocks while representation learning into edges, routing functions and leaf nodes. TabNet differs from these as it embeds soft feature selection with controllable sparsity via sequential attention.

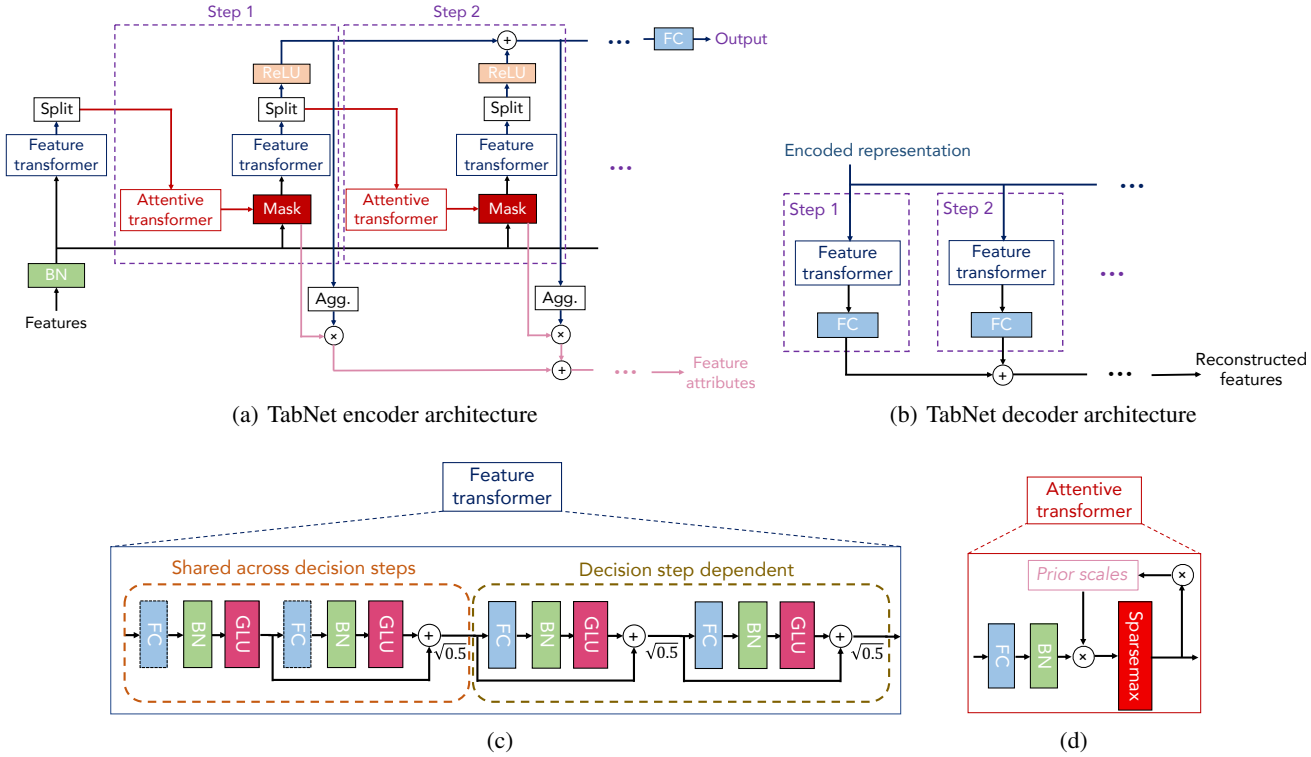
Self-supervised learning: Unsupervised representation learning improves supervised learning especially in small data regime (Raina et al. 2007). Recent work for text (Devlin et al. 2018) and image (Trinh, Luong, and Le 2019) data has shown significant advances – driven by the judicious choice of the unsupervised learning objective (masked input prediction) and attention-based deep learning.

TabNet for Tabular Learning

DTs are successful for learning from real-world tabular datasets. With a specific design, conventional DNN building blocks can be used to implement DT-like output manifold, e.g. see Fig. 3). In such a design, individual feature selection is key to obtain decision boundaries in hyperplane form, which can be generalized to a linear combination of features where coefficients determine the proportion of each feature. TabNet is based on such functionality and it outperforms DTs while reaping their benefits by careful design which: (i) uses sparse instance-wise feature selection learned from data; (ii) constructs a sequential multi-step architecture, where each step contributes to a portion of the decision based on the selected features; (iii) improves the learning capacity via nonlinear processing of the selected features; and (iv) mimics ensembling via higher dimensions and more steps.

Fig. 4 shows the TabNet architecture for encoding tabular data. We use the raw numerical features and consider mapping of categorical features with trainable embeddings. We do not consider any global feature normalization, 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 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 text data (Hudson and Manning 2018) and reinforcement learning (Mott et al. 2019) while searching for a small subset of relevant information in high dimensional input.

Feature selection: We employ a learnable mask $\mathbf{M}[\mathbf{i}] \in \Re^{B \times D}$ 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 ones, and thus the model becomes more parameter efficient. The masking is multiplicative, $\mathbf{M}[\mathbf{i}] \cdot \mathbf{f}$. We use an attentive transformer (see Fig. 4) to obtain the masks using the processed features from the preceding step, $\mathbf{a}[\mathbf{i}-\mathbf{1}]$: $\mathbf{M}[\mathbf{i}] = \text{sparsemax}(\mathbf{P}[\mathbf{i}-\mathbf{1}] \cdot \text{h}_i(\mathbf{a}[\mathbf{i}-\mathbf{1}]))$. Sparsemax normalization (Martins and Astudillo 2016) 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 explainability. Note that $\sum_{j=1}^{D} \mathbf{M}[\mathbf{i}]_{\mathbf{b},\mathbf{j}} = 1$. h_i is a trainable function, shown in Fig. 4 using a FC layer, followed by BN. $\mathbf{P}[\mathbf{i}]$ is the prior scale term, denoting how much a particular feature has been used previously: $\mathbf{P}[\mathbf{i}] = \prod_{j=1}^{i} (\gamma - \mathbf{M}[\mathbf{j}])$, where γ is a relaxation parameter – when $\gamma = 1$, a feature is enforced



(a) TabNet encoder architecture

(b) TabNet decoder architecture

(c)

(d)

Figure 4: (a) TabNet encoder, composed of a feature transformer, an attentive transformer and feature masking. A split block divides the processed representation to be used by the attentive transformer of the subsequent step as well as for the overall output. For each step, the feature selection mask provides 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 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. sparsemax (Martins and Astudillo 2016) is used for normalization of the coefficients, resulting in sparse selection of the salient features.

to be used only at one decision step and as γ increases, more flexibility is provided to use a feature at multiple decision steps. $\mathbf{P}[\mathbf{0}]$ 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 $\mathbf{P}[\mathbf{0}]$ 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 (Grandvalet and Bengio 2004), $L_{sparse} = \sum_{i=1}^{N_{steps}} \sum_{b=1}^{B} \sum_{j=1}^{D} \frac{-\mathbf{M_{b,j}}[\mathbf{i}] \log(\mathbf{M_{b,j}}[\mathbf{i}]+\epsilon)}{N_{steps}\cdot B}$, where ϵ is a small number for numerical stability. We add the sparsity regularization to the overall loss, with a coefficient λ_{sparse}. Sparsity provides a favorable inductive bias 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 (Dauphin et al. 2016), 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 (Gehring et al. 2017). For faster training, we use large batch sizes with BN. Thus, except the one applied to the input features, we use ghost BN (Hoffer, Hubara, and Soudry 2017) form, using 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.[1]

Interpretability: TabNet's feature selection masks can shed light on the selected features at each step. If $\mathbf{M_{b,j}}[\mathbf{i}] = 0$, 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 $\mathbf{M_{b,j}}[\mathbf{i}]$ 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. We aim 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 simply propose $\eta_{\mathbf{b}}[\mathbf{i}] = \sum_{c=1}^{N_d} \mathrm{ReLU}(\mathbf{d_{b,c}}[\mathbf{i}])$ to denote the aggregate decision contribution at i^{th} decision step for the b^{th} sample. Intuitively, if $\mathbf{d_{b,c}}[\mathbf{i}] < 0$, 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, $\mathbf{M_{agg-b,j}} = \sum_{i=1}^{N_{steps}} \eta_{\mathbf{b}}[\mathbf{i}]\mathbf{M_{b,j}}[\mathbf{i}] \Big/ \sum_{j=1}^{D}\sum_{i=1}^{N_{steps}} \eta_{\mathbf{b}}[\mathbf{i}]\mathbf{M_{b,j}}[\mathbf{i}]$.[2]

Tabular self-supervised learning: We propose a decoder architecture to reconstruct tabular features from the TabNet encoded representations. The decoder is composed of feature transformers, followed by FC layers at each decision step. The outputs are summed to obtain the reconstructed features. 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 decoder outputs the reconstructed features, $\mathbf{S}\cdot\hat{\mathbf{f}}$. We initialize $\mathbf{P}[\mathbf{0}] = (\mathbf{1}-\mathbf{S})$ in encoder so that the model emphasizes merely on the known features, and the decoder's last FC layer is multiplied with $\mathbf{S}$ to output the unknown features. We consider the reconstruction loss in self-supervised phase: $\sum_{b=1}^{B}\sum_{j=1}^{D} \left| \frac{(\hat{\mathbf{f}}_{\mathbf{b,j}} - \mathbf{f}_{\mathbf{b,j}})\cdot \mathbf{S}_{\mathbf{b,j}}}{\sqrt{\sum_{b=1}^{B}(\mathbf{f}_{\mathbf{b,j}} - 1/B\sum_{b=1}^{B}\mathbf{f}_{\mathbf{b,j}})^2}} \right|^2$. Normalization with the population standard deviation of the ground truth is beneficial, as the features may have different ranges. We sample $\mathbf{S_{b,j}}$ independently from a Bernoulli distribution with parameter p_s, at each iteration.

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[3] and numerical columns are input without and preprocessing.[4] 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 Appendix, we also present ablation studies on various design and guidelines on selection of the key hyperparameters. For all experiments we cite, we use the same training, validation and testing data split with the original work. Adam optimization algorithm (Kingma and Ba 2014) and Glorot uniform initialization are used for training of all models.[5]

Instance-wise feature selection

Selection of the salient features is crucial for high performance, especially for small datasets. We consider 6 tabular datasets from (Chen et al. 2018) (consisting 10k training samples). The datasets are constructed in such a way that only a subset of the features determine the output. For Syn1-Syn3, salient features are same for all instances (e.g., the

[1] For discrete outputs, we additionally employ softmax during training (and argmax during inference).

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

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

[4] 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.

[5] An open-source implementation will be released.

Table 1: Mean and std. of test area under the receiving operating characteristic curve (AUC) on 6 synthetic datasets from (Chen et al. 2018), for TabNet vs. other feature selection-based DNN models: No sel.: using all features without any feature selection, Global: using only globally-salient features, Tree Ensembles (Geurts, Ernst, and Wehenkel 2006), Lasso-regularized model, L2X (Chen et al. 2018) and INVASE (Yoon, Jordon, and van der Schaar 2019). Bold numbers denote the best 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-regularized	.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**

output of Syn2 depends on features X_3-X_6), and global feature selection, as if the salient features were known, would give high performance. For Syn4-Syn6, salient features are instance dependent (e.g., for Syn4, the output depends on either X_1-X_2 or X_3-X_6 depending on the value of X_{11}), which makes global feature selection suboptimal. Table 1 shows that TabNet outperforms others (Tree Ensembles (Geurts, Ernst, and Wehenkel 2006), LASSO regularization, L2X (Chen et al. 2018)) and is on par with INVASE (Yoon, Jordon, and van der Schaar 2019). For Syn1-Syn3, TabNet performance is close to global feature selection - *it can figure out what features are globally important*. For Syn4-Syn6, eliminating instance-wise redundant features, TabNet improves global feature selection. All other methods utilize a predictive model with 43k parameters, and the total number of parameters is 101k for INVASE due to the two other models in the actor-critic framework. TabNet is a single architecture, and its size is 26k for Syn1-Syn3 and 31k for Syn4-Syn6. The compact representation is one of TabNet's valuable properties.

Performance on real-world datasets

Table 2: Performance for Forest Cover Type dataset.

Model	*Test accuracy (%)*
XGBoost	89.34
LightGBM	89.28
CatBoost	85.14
AutoML Tables	94.95
TabNet	**96.99**

Forest Cover Type (Dua and Graff 2017): The task is classification of forest cover type from cartographic variables. Table 2 shows that TabNet outperforms ensemble tree based approaches that are known to achieve solid performance (Mitchell et al. 2018). We also consider AutoML Tables (AutoML 2019), an automated search framework based on ensemble of models including DNN, gradient boosted DT, AdaNet (Cortes et al. 2016) and ensembles (AutoML 2019) with very thorough hyperparameter search. A single TabNet without fine-grained hyperparameter search outperforms it.

Table 3: Performance for Poker Hand induction dataset.

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

Poker Hand (Dua and Graff 2017): The task is 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 can get 100% accuracy. Yet, conventional DNNs, DTs, and even their hybrid variant of deep neural DTs (Yang, Morillo, and Hospedales 2018) severely suffer from the imbalanced data and cannot learn the required sorting and ranking operations (Yang, Morillo, and Hospedales 2018). Tuned XGBoost, CatBoost, and LightGBM show very slight improvements over them. TabNet outperforms other methods, as it can perform highly-nonlinear processing with its depth, without overfitting thanks to instance-wise feature selection.

Table 4: Performance on Sarcos dataset. Three TabNet models of different sizes are considered.

Model	*Test MSE*	*Model size*
Random forest	2.39	16.7K
Stochastic DT	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 (Vijayakumar and Schaal 2000): The task is regressing inverse dynamics of an anthropomorphic robot arm.

(Tanno et al. 2018) shows that decent performance with a very small model is possible with a random forest. In the very small model size regime, TabNet's performance is on par with the best model from (Tanno et al. 2018) with 100x more parameters. When the model size is not constrained, TabNet achieves almost an order of magnitude lower test MSE.

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

Model	*Test acc. (%)*	*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**

Higgs Boson (Dua and Graff 2017): The task is distinguishing between a Higgs bosons process vs. background. Due to its much larger size (10.5M instances), DNNs outperform DT variants even with very large ensembles. TabNet outperforms MLPs with more compact representations. We also compare to the state-of-the-art evolutionary sparsification algorithm (Mocanu et al. 2018) that integrates non-structured sparsity into training. With its compact representation, TabNet yields almost similar performance to sparse evolutionary training for the same number of parameters. Unlike sparse evolutionary training, the sparsity of TabNet is structured – it does not degrade the operational intensity (Wen et al. 2016) and can efficiently utilize modern multi-core processors.

Table 6: Performance for Rossmann Store Sales dataset.

Model	*Test MSE*
MLP	512.62
XGBoost	490.83
LightGBM	504.76
CatBoost	489.75
TabNet	**485.12**

Rossmann Store Sales (Kaggle 2019b): The task is forecasting the store sales from static and time-varying features. We observe that TabNet outperforms commonly-used methods. The time features (e.g. day) obtain high importance, and the benefit of instance-wise feature selection is observed for cases like holidays where the sales dynamics are different.

Interpretability

Synthetic datasets: Fig. 5 shows the aggregate feature importance masks for the synthetic datasets from Table 1.[6] The output on Syn2 only depends on X_3-X_6 and we observe that the aggregate masks are almost all zero for irrelevant features and TabNet merely focuses on the relevant ones. For Syn4, the 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 the indicator X_{11}, and assigns almost all-zero weights to irrelevant features (the ones other than two feature groups).

Real-world datasets: We first consider the simple task of mushroom edibility prediction (Dua and Graff 2017). TabNet achieves 100% test accuracy on this dataset. It is indeed known (Dua and Graff 2017) that "Odor" is the most discriminative feature – with "Odor" only, a model can get $> 98.5\%$ test accuracy (Dua and Graff 2017). Thus, a high feature importance is expected for it. TabNet assigns an importance score ratio of 43% for it, while other methods like LIME (Ribeiro, Singh, and Guestrin 2016), Integrated Gradients (Sundararajan, Taly, and Yan 2017) and DeepLift (Shrikumar, Greenside, and Kundaje 2017) assign less than 30% (Ibrahim et al. 2019). Next, we consider Adult Census Income. TabNet yields feature importance rankings consistent with the well-known (Lundberg, Erion, and Lee 2018; Nbviewer 2019) (see Appendix) For the same problem, Fig. 6 shows the clear separation between age groups, as suggested by "Age" being the most important feature by TabNet.

Self-supervised learning

Table 7: 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	$\mathbf{61.37 \pm 0.88}$
10k	66.66 ± 0.88	$\mathbf{68.06 \pm 0.39}$
100k	72.92 ± 0.21	$\mathbf{73.19 \pm 0.15}$

Table 7 shows that unsupervised pre-training significantly improves performance on the supervised 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 useful for continual learning and domain adaptation.

Conclusions

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.

[6] For better illustration here, the models are trained with 10M samples rather than 10K as we obtain sharper selection masks.

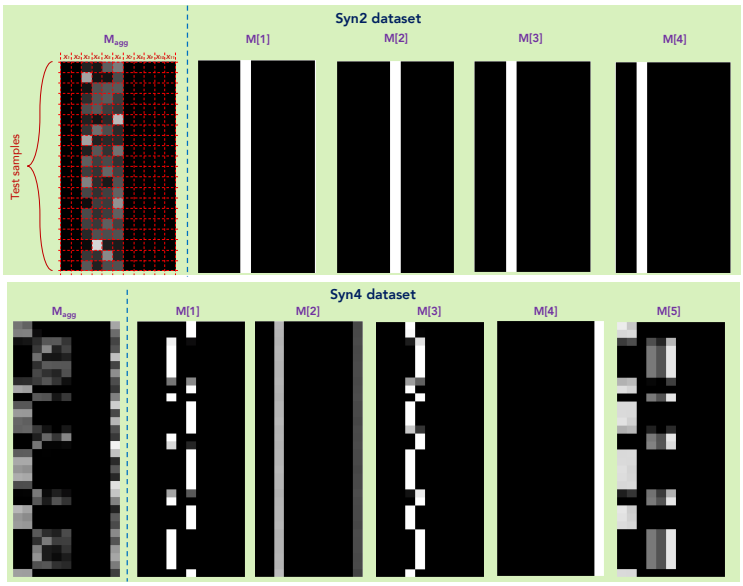


Figure 5: Feature importance masks $\mathbf{M[i]}$ (that indicate feature selection at i^{th} step) and the aggregate feature importance mask $\mathbf{M_{agg}}$ showing the global instance-wise feature selection, on Syn2 and Syn4 (Chen et al. 2018). Brighter colors show a higher value. E.g. for Syn2, only X_3-X_6 are used.

.

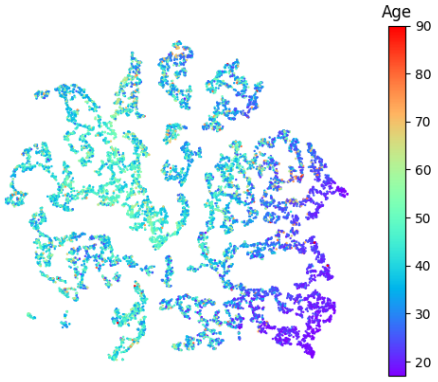


Figure 6: First two dimensions of the T-SNE of the decision manifold for Adult and the impact of the top feature 'Age'.

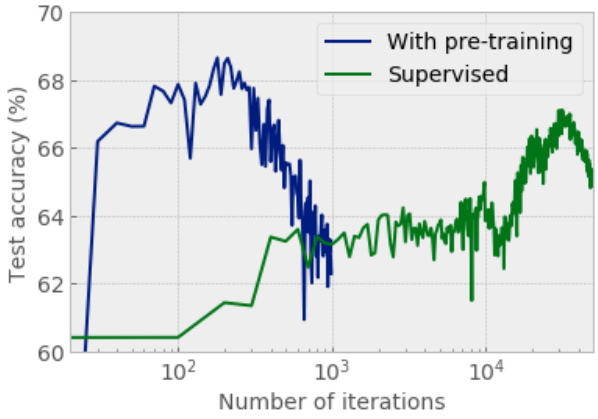


Figure 7: Training curves on Higgs dataset with 10k samples.

Acknowledgements

Discussions with Jinsung Yoon, Kihyuk Sohn, Long T. Le, Ariel Kleiner, Zizhao Zhang, Andrei Kouznetsov, Chen Xing, Ryan Takasugi and Andrew Moore are gratefully acknowledged.

Performance on KDD datasets

Table 8: 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

Appetency, Churn and Upselling datasets are classification tasks for customer relationship management, and KDD Census Income (Dua and Graff 2017) 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 8 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.

Comparison of feature importance ranking of TabNet

Table 9: 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

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").

Self-supervised learning on Forest Cover Type

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 $\{256, 512, 1024, 2048, 4096\}$, the learning rate is chosen from $\{0.005, 0.01.0.02, 0.025\}$, the decay rate is chosen from $\{0.4, 0.8, 0.9, 0.95\}$ and the decay iterations is chosen from $\{0.5k, 2k, 8k, 10k, 20k\}$, 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, 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 (Mitchell et al. 2018). 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. DT, MLP and deep neural DT models follow the same hyperparameters with (Yang, Morillo, and Hospedales 2018). 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-

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	$\mathbf{67.86 \pm 0.63}$
10k	78.85 ± 1.24	$\mathbf{79.22 \pm 0.78}$

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 (Tanno et al. 2018). 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 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 (Mocanu et al. 2018). For gradient boosted trees (Tensorflow 2019), 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 (Catboost 2019) – 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 (Catboost 2019). Obtained with hyperparameter tuning, the MLP is composed of 5 layers of FC (with a hidden unit size of 128), followed by BN and ReLU nonlinearity, trained with a batch size of 512 and a learning rate of 0.001. 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 (Prokhorenkova et al. 2018). The performance of the comparison models are from (Prokhorenkova et al. 2018). 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 (Oza 2005). 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.

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.

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

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 (Hoffer, Hubara, and Soudry 2017). The optimal batch size for TabNet seems considerably higher than the conventional batch sizes used for other data types, such as images or speech.

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.

References

Amodei, D.; Anubhai, R.; Battenberg, E.; Case, C.; Casper, J.; et al. 2015. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *arXiv:1512.02595* .

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

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

Chen, J.; Song, L.; Wainwright, M. J.; and Jordan, M. I. 2018. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. *arXiv:1802.07814* .

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*.

Chui, M.; Manyika, J.; Miremadi, M.; Henke, N.; Chung, R.; et al. 2018. Notes from the AI Frontier. *McKinsey Global Institute* .

Cortes, C.; Gonzalvo, X.; Kuznetsov, V.; Mohri, M.; and Yang, S. 2016. AdaNet: Adaptive Structural Learning of Artificial Neural Networks. *arXiv:1607.01097* .

Dai, Z.; Yang, Z.; Yang, F.; Cohen, W. W.; and Salakhutdinov, R. 2017. Good Semi-supervised Learning that Requires a Bad GAN. *arxiv:1705.09783* .

Dauphin, Y. N.; Fan, A.; Auli, M.; and Grangier, D. 2016. Language Modeling with Gated Convolutional Networks. *arXiv:1612.08083* .

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805* .

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository. URL http://archive.ics.uci.edu/ml.

Gehring, J.; Auli, M.; Grangier, D.; Yarats, D.; and Dauphin, Y. N. 2017. Convolutional Sequence to Sequence Learning. *arXiv:1705.03122* .

Geurts, P.; Ernst, D.; and Wehenkel, L. 2006. Extremely randomized trees. *Machine Learning* 63(1): 3–42. ISSN 1573-0565.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Grabczewski, K.; and Jankowski, N. 2005. Feature selection with decision tree criterion. In *HIS*.

Grandvalet, Y.; and Bengio, Y. 2004. Semi-supervised Learning by Entropy Minimization. In *NIPS*.

Guyon, I.; and Elisseeff, A. 2003. An Introduction to Variable and Feature Selection. *JMLR* 3: 1157–1182.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. *arXiv:1512.03385* .

Hestness, J.; Narang, S.; Ardalani, N.; Diamos, G. F.; Jun, H.; Kianinejad, H.; Patwary, M. M. A.; Yang, Y.; and Zhou, Y. 2017. Deep Learning Scaling is Predictable, Empirically. *arXiv:1712.00409* .

Ho, T. K. 1998. The random subspace method for constructing decision forests. *PAMI* 20(8): 832–844.

Hoffer, E.; Hubara, I.; and Soudry, D. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv:1705.08741* .

Hudson, D. A.; and Manning, C. D. 2018. Compositional Attention Networks for Machine Reasoning. *arXiv:1803.03067* .

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

Ibrahim, M.; Louie, M.; Modarres, C.; and Paisley, J. W. 2019. Global Explanations of Neural Networks: Mapping the Landscape of Predictions. *arxiv:1902.02384* .

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

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

Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; et al. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *NIPS*.

Ke, G.; Zhang, J.; Xu, Z.; Bian, J.; and Liu, T.-Y. 2019. TabNN: A Universal Neural Network Solution for Tabular Data. URL https://openreview.net/forum?id=r1eJssCqY7.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. In *ICLR*.

Kontschieder, P.; Fiterau, M.; Criminisi, A.; and Bulò, S. R. 2015. Deep Neural Decision Forests. In *ICCV*.

Lai, S.; Xu, L.; Liu, K.; and Zhao, J. 2015. Recurrent Convolutional Neural Networks for Text Classification. In *AAAI*.

Lundberg, S. M.; Erion, G. G.; and Lee, S. 2018. Consistent Individualized Feature Attribution for Tree Ensembles. *arXiv:1802.03888* .

Martins, A. F. T.; and Astudillo, R. F. 2016. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. *arXiv:1602.02068* .

Mitchell, R.; Adinets, A.; Rao, T.; and Frank, E. 2018. XGBoost: Scalable GPU Accelerated Learning. *arXiv:1806.11248* .

Mocanu, D.; Mocanu, E.; Stone, P.; Nguyen, P.; Gibescu, M.; and Liotta, A. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications* 9.

Mott, A.; Zoran, D.; Chrzanowski, M.; Wierstra, D.; and Rezende, D. J. 2019. S3TA: A Soft, Spatial, Sequential, Top-Down Attention Model. URL https://openreview.net/forum?id=B1gJOoRcYQ.

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

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

Prokhorenkova, L.; Gusev, G.; Vorobev, A.; Dorogush, A. V.; and Gulin, A. 2018. CatBoost: unbiased boosting with categorical features. In *NIPS*.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434* .

Raina, R.; Battle, A.; Lee, H.; Packer, B.; and Ng, A. Y. 2007. Self-Taught Learning: Transfer Learning from Unlabeled Data. In *ICML*.

Ribeiro, M.; Singh, S.; and Guestrin, C. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *KDD*.

Shavitt, I.; and Segal, E. 2018. Regularization Learning Networks: Deep Learning for Tabular Datasets.

Shrikumar, A.; Greenside, P.; and Kundaje, A. 2017. Learning Important Features Through Propagating Activation Differences. *arXiv:1704.02685* .

Sundararajan, M.; Taly, A.; and Yan, Q. 2017. Axiomatic Attribution for Deep Networks. *arXiv:1703.01365* .

Tanno, R.; Arulkumaran, K.; Alexander, D. C.; Criminisi, A.; and Nori, A. V. 2018. Adaptive Neural Trees. *arXiv:1807.06699* .

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

Trinh, T. H.; Luong, M.; and Le, Q. V. 2019. Selfie: Self-supervised Pretraining for Image Embedding. *arXiv:1906.02940* .

Vijayakumar, S.; and Schaal, S. 2000. Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space. In *ICML*.

Wang, S.; Aggarwal, C.; and Liu, H. 2017. Using a random forest to inspire a neural network and improving on it. In *SDM*.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning Structured Sparsity in Deep Neural Networks. *arXiv:1608.03665* .

Xu, L.; Skoularidou, M.; Cuesta-Infante, A.; and Veeramachaneni, K. 2019. Modeling Tabular data using Conditional GAN. *arXiv:1907.00503* .

Yang, Y.; Morillo, I. G.; and Hospedales, T. M. 2018. Deep Neural Decision Trees. *arXiv:1806.06988* .

Yoon, J.; Jordon, J.; and van der Schaar, M. 2019. INVASE: Instance-wise Variable Selection using Neural Networks. In *ICLR*.