Tree-Regularized Tabular Embeddings

Xuan Li
Amazon
milanlx@amazon.com

Yun Wang
Amazon
yunwng@amazon.com

Bo Li
Amazon
booli@amazon.com

Abstract

Tabular neural network (NN) has attracted remarkable attentions and its recent advances have gradually narrowed the performance gap with respect to tree-based models on many public datasets. While the mainstreams focus on calibrating NN to fit tabular data, we emphasize the importance of homogeneous embeddings and alternately concentrate on regularizing tabular inputs through supervised pretraining. Specifically, we extend a recent work coined as DeepTLF [5], and utilize the structure of pretrained tree ensembles to transform raw variables into a single vector (T2V), or an array of tokens (T2T). Without loss of space efficiency, these binarized embeddings can be directly consumed by canonical tabular NN with full-connected or attention-based building blocks. Through quantitative experiments on 88 OpenML datasets with binary classification task, we validated that the proposed tree-regularized representation not only tapers the difference with respect to tree-based models, but also achieves on-par and better performance when compared with advanced NN models. Most importantly, it possesses better robustness and can be easily scaled and generalized as standalone encoder for tabular modality.

1 Introduction

Neural Network has achieved exceptional breakthroughs in the unstructured data regimes including image [11, 31], text [6, 33], video [27, 36] and speech [3, 42], whereas its performance is still capped by tree-based approaches when applied to structured tabular datasets [19, 30]. As there are growing demands on leveraging NN's capability to incorporate tabular modality for broader use cases such as multimodal learning [13, 14, 20, 35, 44], it is critical to further boost tabular NN to its upper limit to better support these expansions.

Many recent works have attempted to bridge this gap by applying techniques that have demonstrated superior performance on other modalities to tabular learning. For example, a majority of the approaches follow a model-centric paradigm of applying simple feature transformation yet sophisticated customization on NN frameworks to fit tabular input. However, the underemphasis on feature quality could overshadow the efficacy of NN. Essentially, unlike image, text and speech data which have basic units (pixel, word, phoneme) that formulate a homogeneous representation space, tabular features are heterogeneous in nature as the columns possess different data sources, scales and distributions [1, 16, 28]. Likewise, simple feature transformations such as min-max normalization might be incapable to make tabular input homogeneous enough to be consumed by NN backbones. Subsequently, we follow the data-centric scenario and seek data transformation strategies to acquire dedicated tabular embeddings.

Precisely, in this work we revisit the underexplored rationale on calibrating tabular data to fit NN. As visioned in Figure 1, we leverage supervised pretraining to learn tree-regularized representations through an embedder module. In a snapshot, the proposed methodology exploits the structure of pretrained tree ensembles to generate binarized embeddings through a pairwise comparison between value in raw variable and the corresponding thresholds in tree node. Spanning the latent space of

Table Representation Learning Workshop at NeurIPS 2023

trees, the enriched representations can be fed into tabular NN directly and finetuned for different downstream tasks. In terms of implementation, we optimized and extended DeepTLF [5], an overlooked advancement in boosting tabular NN with tree-transformed vector, to make it scalable for larger datasets and generalizable for vaster frameworks. On one hand, instead of transforming the data and storing the vectors all at once, we deploy it on-the-fly for each mini-batch during model training and inference, thus requesting no exhaustive memory usage. To compensate for the ensuing time complexity, we reformulate the pairwise comparison with matrix manipulation, which maintains the forward evaluation time at a similar scale. These two optimizations are essential for industrial tabular applications where the datasets might contain hundreds of columns and millions of rows. On the other hand, beyond generating embeddings as a single vector, we also treat each tree as tokenizer and further support tree-level transformation to obtain embeddings as an array of tokens. Essentially, it enables the representations to be compatible with attention-based models [22, 37] that have received increasing attentions in the tabular learning communities. For evaluation, we leverage the TabZilla framework [30] and compare with a variety of state-of-the-art (SOTA) methods on 88 OpenML datasets with binary classification tasks.

In summary, the contributions and novelties of this work are as follows:

- We approach tabular representation learning from a data-centric perspective. Through a toy synthetic experiment, we reveal that simple NN model can always outperform well-tuned tree-based model in a homogeneous space, and therefore highlight the desideratum of tabular-specific transformations.
- We improve a recent approach, DeepTLF [5], and further implement scalable algorithms to obtain tree-regularized tabular embeddings as a single vector (T2V), or an array of vectors (T2T). In essence, the transformed representations can be directly integrated with advanced tabular NN models with multi-layered perception (MLP) or multi-head attention (MHA) as building blocks.
- We run comprehensive evaluations with a collection of 88 OpenML datasets on binary classification tasks. We validate that T2T with MHA backbones can narrow the performance gap with respect to tree-based models and achieve comparable or better performance compared to SOTA tabular NN models. More importantly, our methods show better robustness, and support generalizations at scale.

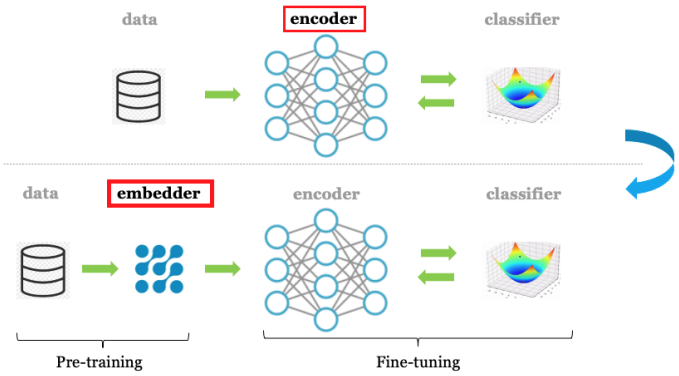


Figure 1: An overview of data-centric tabular learning

2 Related Work

Heterogeneity in tabular emebddings Unlike image, text and speech data that are composed of homogeneous units such as pixel, character and spectral band, tabular data are usually gathered from various information sources which made it heterogeneous by design. For example, tabular variables have different distributions [28], locate in irregular spaces [28], and contain different types including categorical, numerical and ordinal [1, 38] format. Although several researchers [16, 28] have pointed out heterogeneity to be the fundamental blocker that restricts NN's generalization on tabular data, qualitative definitions and quantitative metrics are still missing for rigorous evaluations. However,

the t-SNE plots [40] can be utilized as a qualitative proxy to visualize the level of heterogeneity for different tabular representations [5].

Tabular NN models and pretraining Inspired by the recent advance of NN in other fields, many researchers have customized these techniques for tabular modality from two perspectives including modeling architectures and pretraining frameworks.

In terms of modeling architectures, MLP [16, 18, 17, 23], MHA [7, 18, 17, 22, 37], CNN [45] and GNN [12] have been modified and found effective to boost performance over tree models on different public datasets. Although there is still no single option that dominates the rest, there are growing interests of adapting MHA in recent progress such as multimodal learning [13] and reasoning with language models [21]. Intuitively, the self-attention mechanism in MHA is designed to discover relational pattern among the input features, i.e., understanding the context between words, which is similar to the conditional split mechanism utilized in tree-based models.

Besides, unsupervised, self-supervised and supervised pretraining have been leveraged by many works to obtain tabular-specific embeddings. For unsupervised scenario, quantile binning and periodic activation have been explored to independently encode each feature without interactions [17]. For self-supervised pretext tasks, contrastive learning [4, 8, 10, 20, 34, 37, 43] and masked reconstruction [2, 22, 29, 34, 39, 43] are commonly adopted and the latter is reported to have better performance. For the supervised counterpart, knowledge distillation from ensembles of pretrained NNs [26] or boosting trees [5, 24, 41] are implemented and reported to outperform tree models. However, this array of research is not well-explored, which is probably due to the concerns of overfitting [15] and scalability.

3 Towards Data-Centric Tabular Learning

In contrast to model-centric approaches that focus on calibrating NN models to fit with tabular data, we highlight the coupling effect between homogeneous features and NN models, and instead leverage pretraining to regularize the input latent space. As showed in Figure 1, we first utilize an embedder at pretraining stage to learn representations through supervised pretraining. Specifically, we implement tree-to-vector (T2V) to support fully-connected encoders, and tree-to-tokens (T2T) to support attention-based encoders. Before diving into the technical details, we first introduce a synthetic experiment that motivates us towards doubling down on data-centric approaches.

notations Let $\mathbb{R}^n$ be the n-dimensional Euclidean space and $||\cdot||_2$ be the Euclidean norm (L2 norm). We denote the unit hypersphere in $\mathbb{R}^d$ by $\mathbb{S}^{d-1} := \{\mathrm{x} \in \mathbb{R}^d : ||\mathrm{x}||_2 = 1\}$. We use $f_\theta(\cdot)$ to denote function $\{f(\cdot) : \mathbb{R}^d \to \mathbb{R}^c\}$ parameterized by θ. With loss of generality, we use x, x, X to represent scalar, vector and matrix respectively. For matrix X, we use X_i^j to index the element in the i-th row and j-th column.

3.1 Synthetic Experiments

To validate the coupling effects between homogeneous latent space and neural models, we conduct a toy experiment with synthetic data which simulates homogeneous feature spaces. For this homogeneous scenario, we generate balanced 100-dimensional data that are uniformly pinpointed on a unit hypersphere around two central points c_0 and c_1, where the two centers are diagonal to each other and also are located on that unit hypersphere, i.e., $\mathrm{c}_0 = -\mathrm{c}_1$. We use the term β to control the maximum distance between a sample (x, y) and its central point, i.e., $P(y = i \mid ||\mathrm{x} - \mathrm{c}_\mathrm{i}||_2 \leq \beta) = 1$. Intuitively, a small β indicates the data are tightly clustered around centers, while a large β indicates patterned overlapping on the boundaries. An illustrative visualization of the synthetic data in 2-dimensional scenario can be found in Figure 3.

Through uniform sampling with rejection, we generate 10k balanced samples and split them into training, validation and testing bucket with 60%, 20% and 20% in proportion. For comparison, we train a two-layer MLP ($100 \to 100 \to 2$) as NN model, a XGBoost (XGB) with default hyperparameter, and a XGB with well-tuned hyperparameter as tree-based models. We run 5 trials of experiment per β and report the average of accuracy in Figure 2. By varying β between 1.85 and 2.20 with a 0.05 interval, we found that NN can always outperform the default as well as the well-tuned XGB in

this hyperspherical feature space. With different features regularized within the same scale, we posit NN might have superiority over tree-based models in this homogeneous latent space, and therefore introduce tree-regularized embeddings that are aligned with this observation.

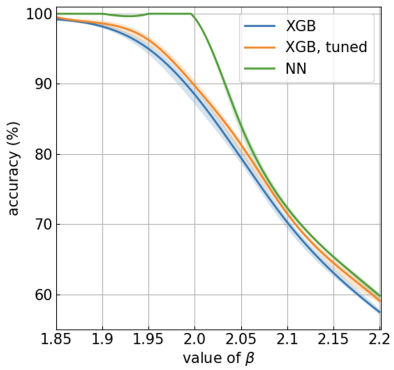


Figure 2: Comparison between MLP and XGB with varying β in terms of accuracy

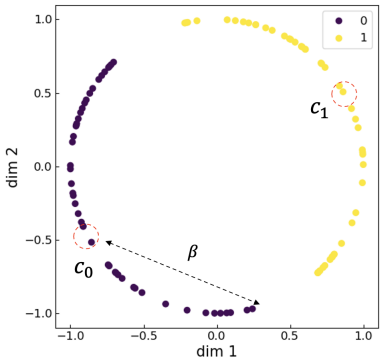


Figure 3: A visualization of the synthetic data in 2D scenario

3.2 Tree-regularized Embedding

supervised tree-regularized embeddings As a realization of supervised pretraining, the tree-regularized approach takes advantages of tree information from XGB to formulate new embeddings with feature interactions. Ideally, this procedure will transform the heterogeneous tabular data into homogeneous format by distilling knowledge from nodes of trained decision trees [5]. As showed in Figure 4, it will firstly extracts node information - a tuple of variable index and threshold - from each tree as a map, and then binarizes each data by comparing the corresponding variable value with respect to the threshold given the index. Interested readers can refer to Figure 11 for an illustrative example. To make the embedder compatible with different NN encoders and scalable with large datasets, we extend this simple setup from work [5] and introduce T2V and T2T to support fully-connected and attention-based models.

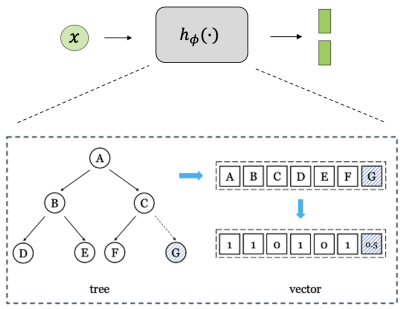


Figure 4: Overview of tree-to-vector (T2V) embedding

T2V: With the embedding vectors extracted from each tree, we perform a preprocessing on the collection of $\{\text{variable\_index} : \text{threshold}\}$ map to remove duplicated instances based on rounded threshold, concatenate the vectors to form a single one-dimensional vector, and finally integrate the embedding with MLP encoders during model training. To make the embedder scalable, we reformulate the pairwise ({value, threshold}) comparison with matrix manipulation, and only employ this operation within each mini-batch on the fly, which we denote as in-batch transformation. Specifically, assume we have a data matrix $X \in \mathbb{R}^{n \times m}$ with n instances and m variables, and a corresponding collection $M \in \mathbb{R}^{k \times 2}$ with k pairs of the $\{\text{variable\_index}, \text{threshold}\}$ map extracted from tree ensembles (XGB). According to Eq (1), we can construct a matrix $U \in \mathbb{R}^{m \times k}$, and a matrix $V \in \mathbb{R}^{m \times k}$

composed of m stacked vector v ($\mathrm{v} \in \mathbb{R}^k, \mathrm{v}_i = M_i^2$), so that the operation of $\mathrm{sign}(XU - V)$ is equivalent to the iterative pairwise comparison of {value, threshold}. Most importantly, the in-batch transformation makes the algorithm generalizable to much larger datasets with hundreds of columns and millions of rows. We provide the details in Algorithm (1) and a PyTorch-like pseudocode in Figure 5.

$$U^i_{M_i^1} = \begin{cases} 1, & \forall\, i \in \{1, 2, ..., k\} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

T2T: To make it compatible with MHA backbone, we treat the embeddings from each tree as token and apply paddings to ensure every token are aligned in dimension. The final embeddings for each data instance have a dimension of $\mathbb{R}^{d \times k}$, where d is the number of tree ensembles in XGB and k is the maximum number of nodes in these trees. Precisely, we pad 0.5 to non-splitting nodes (to make tree complete) and -1.0 at the tail of the embedding vector to make it aligned with dimension k. To ensure the semantics of token are consistent, we preserve the topological order of each tree through level order traversal when extracting tree nodes. The details of these operations can be found in Algorithm (2) and Figure 11a. Matrix manipulations and in-batch transformation are applied similarly as T2V to account for scalability. Intuitively, the final output X ($X \in \mathbb{R}^{n \times d \times k}$) can be regarded as an array of tokens and directly consumed by transformers with attention block.

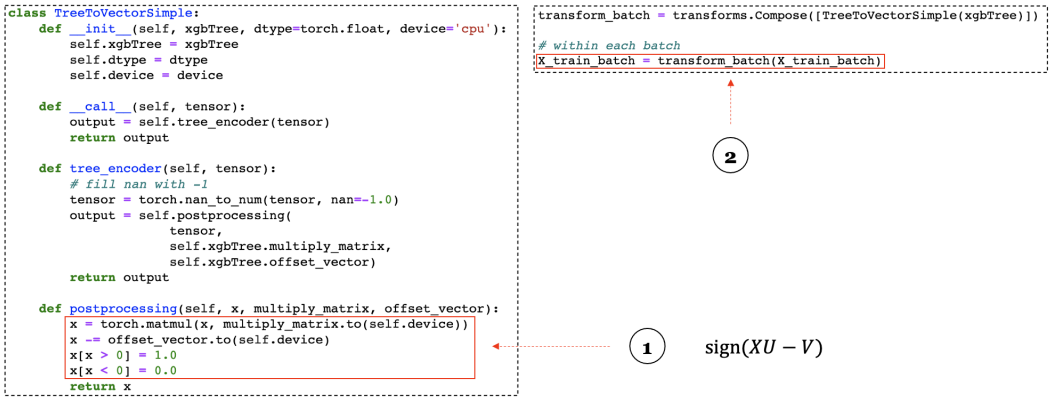


Figure 5: Pseudocode of in-batch transformation for T2V in a PyTorch-like style. Step 1 replaces pairwise comparison with matrix manipulation, while Step 2 showcases on-the-fly transformations for mini-batch implemented through the tranforms.Compose module in PyTorch.

4 Experiments

4.1 Datasets, models, and training details

We leverage a subset of the benckmark datasets provided in TabZilla [30] repository to evaluate the effectiveness, generalizability and scalability of the proposed methods. Specifically, we select 91 OpenML [1] datasets with binary classification task and utilize the Area Under the Curve (AUC) in percentage as evaluation metrics. We apply light preprocessing to fill missing value with zero and convert categorical variables to ordinal values through label encoding.

We keep model framework consistent throughout the experiments. For T2V, we use two-layered MLP with ReLU activation and fix the hidden dimensions as $m \to 256 \to 128 \to 2$, where m is the dimension of T2V embeddings. For T2T, we use MHA encoder configured with 2 identical building blocks, where each block consists of 4 heads with embedding dimension as 8. An one-layered MLP ($m \to 128 \to 2$) is connected with the concatenated output of MHA as classification head. For comprehensive comparisons, we select CatBoost [32], XGBoost [9] and LightGBM [25] as tree-based baselines. In addition, we use SAINT [37] and the ResNet-like model [18] as SOTA NN baselines

[1] https://www.openml.org/

given the rankings reported in [30]. Finally, we include a two-layered MLP ($m \to 128 \to 2$, denoted as MLP) with min-max normalization applied on raw variables as a vanilla NN baseline.

For evaluation, we leverage the default 10 training/testing splits provided by OpenML and report the mean AUC over the 10 runs for each dataset. Similar to TabZilla, for each split we further extract a fixed validation set from the training set to make the training/validation/testing proportion as 80%, 10% and 10% respectively. Additionally, we fix the hyperparameters for each model with their default values for generalization purpose. Specifically, for all NN-based models we apply Adam as default optimizer with learning rate as 0.001 and batch size as 64. Early stopping with 10 epochs and 600 seconds timeout is applied to both tree-based and NN-based models. All experiments are run on an A10G GPU with approximately 3 GPU days.

4.2 Performance Evaluation

We summarize the experiment results in this section. In terms of robustness, we find most of the NN models cannot generalize to the entire datasets, and therefore compare models in full-scale and partial-scale scenarios based on their dataset coverage. Precisely, we compare T2V, T2T, MLP with tree-based models on 88 datasets as full-scale scenario. For partial-scale case, we compare T2V with SAINT and ResNet on 59 and 73 datasets respectively. Also, we provide a heuristic analysis on the time complexity of in-batch transformation by varying batch size and number of tree ensembles.

robustness We report the number of datasets that can be evaluated by each method in Table 1. In general, we find tree-based models achieve the best robustness while NN models, such as SAINT and ResNet, suffer from numerical and timeout issue on a variety of datasets. Notably, T2V and T2T have better robustness as they can generalize to 88/91 of the cases.

CatBoost	XGBoost	LightGBM	T2V	T2T	MLP	SAINT	ResNet
91	91	91	88	88	88	59	73

Table 1: Number of datasets can be evaluated by tree-based and NN-based models

full-scale comparison Given the availability of data coverage, we first compare T2V, T2T and the vanilla MLP with respect to tree-based models. The results are reported in Table 2 where the methods are ranked by the mean AUC taken over across the 88 overlapped datasets. The distribution of AUC attained by different method is showed in Figure 9 in Appendix. Firstly, while T2T outperforms the vanilla MLP, it still has a 3.43% gap in percentaged AUC with respect to the best tree-based model. Second, T2V underperforms MLP, probably because a shallow NN backbone is not sufficient for the high-dimensional embeddings. Moreover, we point out the diversity existed in the datasets as each method can achieve the highest as well as the lowest ranking. This observation is aligned with the results reported in TabZilla [30], where the authors found no single approach can consistently dominate the rest and the difference in performance was insignificant in many of the cases.

Algorithm	Rank ↓				AUC (%) ↑
	min	max	mean	median	mean
CatBoost	1	6	2.38	2	88.06
XGBoost	1	6	2.83	2	87.70
LightGBM	1	6	3.16	3	86.37
T2T	1	6	4.07	4	84.63
MLP	1	6	4.22	4	84.42
T2V	1	6	4.45	5	83.15

Table 2: Comparison between T2V, T2T, MLP and tree-based models on 88 datasets

partial-scale comparison Given the results from full-scale comparison, we also conduct pairwise comparison between T2T, SAINT and ResNet on the intersected datasets. For comparison, we check the difference in percentaged AUC between two methods and define a win on a dataset if the former method achieves a high AUC. The histogram of difference in AUC between {T2T, SAINT} and {T2T, ResNet} are showed in Figure 6 and 7 respectively. Comparing T2T and SAINT, we find the former win 39 out of 59 of the datasets (66.10%) and achieve a 3.74% absolute lift in percentaged AUC. When compared with ResNet, however, we find T2T can win 36 of the 73 cases (49.31%) with a 0.13% difference in percentaged AUC on average. From the histogram it is found the majority of the differences are within 0% – 10% range, and each method has generalization issue on several datasets. The distribution of the AUC can be found in Figure 10.

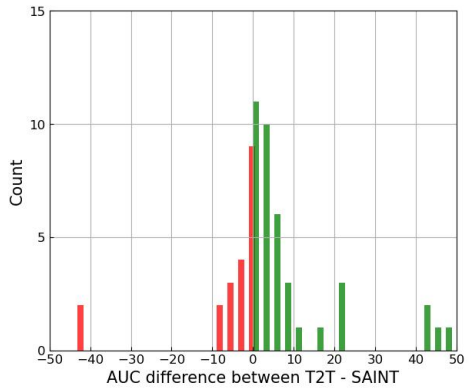


Figure 6: Histogram of difference in AUC between T2T and SAINT

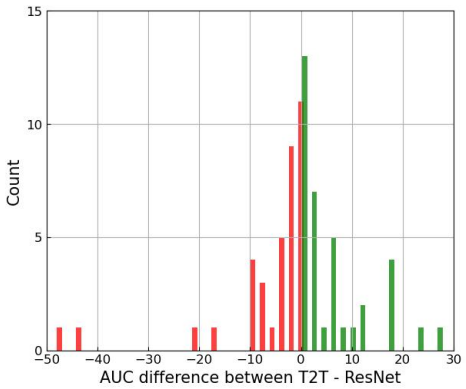


Figure 7: Histogram of difference in AUC between T2T and ResNet

time complexity analysis As our methods made a trade-off between time and space complexity, we further conduct an analysis to evaluate the computational overhead with the synthetic datasets introduced in the previous section. Basically, we compare the forward-pass time between T2V with MLP and vanilla MLP for mini-batch evaluations. The results are showed in Figure 8, where the execution time is reported as the average over 10 runs per scenario. By varying the batch size and number of tree ensembles, we find T2V scales well with respect to number of tree ensembles. However, for each mini-batch it takes 3x - 5x evaluation time when compared to the vanilla MLP for batch size up to 512.

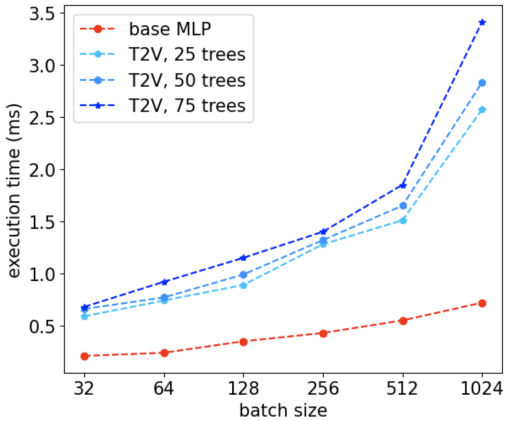


Figure 8: Comparison of time complexity between T2V and vanilla MLP on synthetic datasets

5 Conclusions and Future Works

We follow a data-centric perspective and propose two methods to obtain tree-regularized embeddings with efficient in-batch transformation. Our improved tabular embeddings, T2V and T2T, can be simply consumed by many tabular NN frameworks with MLP and MHA as building block. Through comprehensive evaluations on 88 OpenML datasets, we show strong robustness and on-par performance with respect to SOTA NN models on binary classification tasks. These results demonstrate the potential of generalizing and scaling our approaches as tabular encoder for broader applications that require tabular modality.

We plan to explore several directions to further improve the effectiveness and scalability of the proposed methods. Firstly, we will conduct architecture search to explore consonant NN designs that works with tree-regularized embeddings. In addition, for T2T we will try to further encode each tree as discrete token and utilize self-supervised pretraining to learn embeddings with customizable dimension through contrastive or reconstruction task. Finally, we point out a lack of quantitative metric on homogeneity and benchmark datasets at industrial scale, which are worth exploring in the next sprint.

Acknowledgements

We would like to thank Ege Beyazit, Jonathan Kozaczuk, Mihir Pendse, Pankaj Rajak, Jiajian Lu and Vanessa Wallace for valuable discussions, feedback and support.

References

[1] Rishabh Agarwal et al. "Neural additive models: Interpretable machine learning with neural nets". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 4699–4711.

[2] Sercan Ö Arik and Tomas Pfister. "Tabnet: Attentive interpretable tabular learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8. 2021, pp. 6679–6687.

[3] Alexei Baevski et al. "wav2vec 2.0: A framework for self-supervised learning of speech representations". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12449–12460.

[4] Dara Bahri et al. "Scarf: Self-supervised contrastive learning using random feature corruption". In: *arXiv preprint arXiv:2106.15147* (2021).

[5] Vadim Borisov et al. "DeepTLF: robust deep neural networks for heterogeneous tabular data". In: *International Journal of Data Science and Analytics* (2022), pp. 1–16.

[6] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[7] Kuan-Yu Chen et al. "Trompt: Towards a Better Deep Neural Network for Tabular Data". In: *arXiv preprint arXiv:2305.18446* (2023).

[8] Suiyao Chen et al. "ReConTab: Regularized Contrastive Representation Learning for Tabular Data". In: *arXiv preprint arXiv:2310.18541* (2023).

[9] Tianqi Chen et al. "Xgboost: extreme gradient boosting". In: *R package version 0.4-2* 1.4 (2015), pp. 1–4.

[10] Sajad Darabi et al. "Contrastive Mixup: Self-and Semi-Supervised learning for Tabular Domain". In: *arXiv preprint arXiv:2108.12296* (2021).

[11] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[12] Lun Du et al. "TabularNet: A neural network architecture for understanding semantic structures of tabular data". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 322–331.

[13] Sayna Ebrahimi et al. "LANISTR: Multimodal Learning from Structured and Unstructured Data". In: *arXiv preprint arXiv:2305.16556* (2023).

[14] Nick Erickson et al. "Multimodal automl for image, text and tabular data". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 4786–4787.

[15] Yutong Feng et al. “Rethinking supervised pre-training for better downstream transferring”. In: *arXiv preprint arXiv:2110.06014* (2021).

[16] James Fiedler. “Simple modifications to improve tabular neural networks”. In: *arXiv preprint arXiv:2108.03214* (2021).

[17] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. “On embeddings for numerical features in tabular deep learning”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24991–25004.

[18] Yury Gorishniy et al. “Revisiting deep learning models for tabular data”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[19] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. “Why do tree-based models still outperform deep learning on tabular data?” In: *arXiv preprint arXiv:2207.08815* (2022).

[20] Paul Hager, Martin J Menten, and Daniel Rueckert. “Best of Both Worlds: Multimodal Contrastive Learning with Tabular and Imaging Data”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 23924–23935.

[21] Stefan Hegselmann et al. “Tabllm: Few-shot classification of tabular data with large language models”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2023, pp. 5549–5581.

[22] Xin Huang et al. “Tabtransformer: Tabular data modeling using contextual embeddings”. In: *arXiv preprint arXiv:2012.06678* (2020).

[23] Arlind Kadra et al. “Well-tuned simple nets excel on tabular datasets”. In: *Advances in neural information processing systems* 34 (2021), pp. 23928–23941.

[24] Guolin Ke et al. “DeepGBM: A deep learning framework distilled by GBDT for online prediction tasks”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 384–394.

[25] Guolin Ke et al. “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems* 30 (2017).

[26] Chung-Wei Lee, Pavlos Anastasios Apostolopulos, and Igor L Markov. “Practical Knowledge Distillation: Using DNNs to Beat DNNs”. In: *arXiv preprint arXiv:2302.12360* (2023).

[27] Ze Liu et al. “Video swin transformer”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 3202–3211.

[28] Chao Ma et al. “VAEM: a deep generative model for heterogeneous mixed type data”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 11237–11247.

[29] Kushal Majmundar et al. “Met: Masked encoding for tabular data”. In: *arXiv preprint arXiv:2206.08564* (2022).

[30] Duncan McElfresh et al. “When Do Neural Nets Outperform Boosted Trees on Tabular Data?” In: *arXiv preprint arXiv:2305.02997* (2023).

[31] Maxime Oquab et al. “DINOv2: Learning Robust Visual Features without Supervision”. In: *arXiv preprint arXiv:2304.07193* (2023).

[32] Liudmila Prokhorenkova et al. “CatBoost: unbiased boosting with categorical features”. In: *Advances in neural information processing systems* 31 (2018).

[33] Alec Radford et al. “Learning transferable visual models from natural language supervision”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8748–8763.

[34] Ivan Rubachev et al. “Revisiting pretraining objectives for tabular deep learning”. In: *arXiv preprint arXiv:2207.03208* (2022).

[35] Xingjian Shi et al. “Benchmarking multimodal automl for tabular data with text fields”. In: *arXiv preprint arXiv:2111.02705* (2021).

[36] Uriel Singer et al. “Make-a-video: Text-to-video generation without text-video data”. In: *arXiv preprint arXiv:2209.14792* (2022).

[37] Gowthami Somepalli et al. “SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training”. In: *arXiv preprint arXiv:2106.01342* (2021).

[38] Matthew Tancik et al. “Fourier features let networks learn high frequency functions in low dimensional domains”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7537–7547.

[39] Talip Ucar, Ehsan Hajiramezanali, and Lindsay Edwards. “Subtab: Subsetting features of tabular data for self-supervised representation learning”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 18853–18865.

[40] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[41] Xiang Wang et al. “Tem: Tree-enhanced embedding model for explainable recommendation”. In: *Proceedings of the 2018 world wide web conference*. 2018, pp. 1543–1552.

[42] Dongchao Yang et al. “UniAudio: An Audio Foundation Model Toward Universal Audio Generation”. In: *arXiv preprint arXiv:2310.00704* (2023).

[43] Jinsung Yoon et al. “Vime: Extending the success of self-and semi-supervised learning to tabular domain”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 11033–11043.

[44] Yiyuan Zhang et al. “Meta-transformer: A unified framework for multimodal learning”. In: *arXiv preprint arXiv:2307.10802* (2023).

[45] Yitan Zhu et al. “Converting tabular data into images for deep learning with convolutional neural networks”. In: *Scientific reports* 11.1 (2021), p. 11325.

Appendix

More Results on partial-scale comparisons between NN Models

We present the comparison of T2V, T2T, SAINT and ResNet on 59 intersected datasets in Table 3. Similar to the observations reported in the partial-scale comparison, we find T2V outperforms SAINT but slightly underperforms ResNet. As showed in Figure 10, T2T does not generalize well on several datasets which limit its performance on average.

Algorithm	Rank ↓				AUC (%) ↑
	min	max	mean	median	mean
ResNet	1	4	2.15	2	84.87
T2T	1	4	2.29	2	84.72
T2V	1	4	2.61	3	83.92
SAINT	1	4	3.01	3	81.46

Table 3: Comparison between NN models on intersection datasets

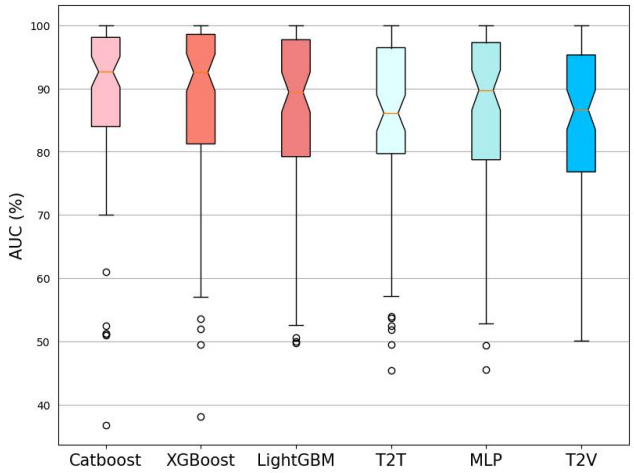


Figure 9: Distribution of AUC (%) for full-scale comparison

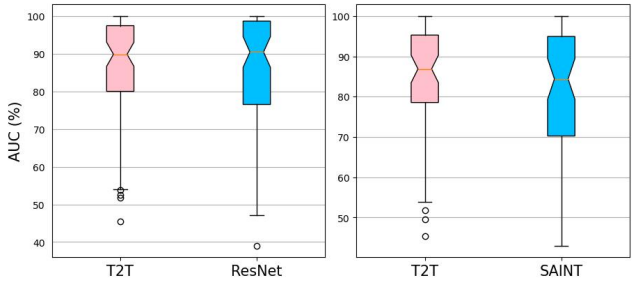


Figure 10: Distribution of AUC (%) for partial-scale comparison

Tree-to-Vector algorithms

We introduce T2V and T2T in Algorithm 1 and 2 respectively. For T2V, we set $\epsilon = 4$, i.e., the thresholds are rounded with 4 digit of decimals. For T2T, we set $\tau = 0.5$ and $\eta = -1.0$, where the

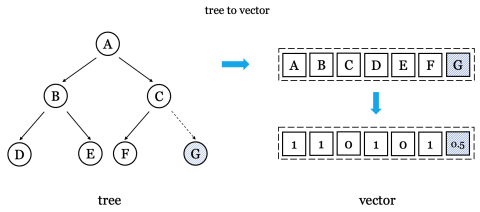
former is the default value to fill the complete tree and the later the default value to pad each token. The flowchart of T2V with an illustrative example is showed in Figure 11.

Algorithm 1: Tree to Vector (T2V)

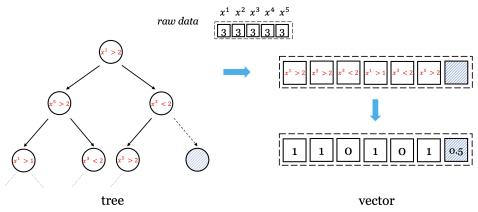
Input: xgb_trees, ϵ
Output: emb_map
Init: emb_map = {}
for tree $\in$ xgb_trees **do**
  for node $\in$ tree **do**
    {var_key, var_val} = node;
    var_val.round(ϵ);
    if {var_key, var_val} $\notin$ emb_map **then**
      emb_map[var_key].append(var_val);
    end
  end
end

Algorithm 2: Tree to Tokens (T2T)

Input: xgb_trees, τ, η
Output: emb_vec
Init: vec_len = 0, emb_vec = []
for tree $\in$ xgb_trees **do**
  l = tree.count_node() ;
  vec_len = max(vec_len, l)
end
for tree $\in$ xgb_trees **do**
  vec = tree.to_vec(τ);
  vec.pad(vec_len, η);
  emb_vec.append(vec);
end



(a) T2T: extract node. The nodes are traversed in level order to maintain tree structure.



(b) T2T: binary encode. A pseudo node G is added to make the tree complete and infilled with 0.5 by default.

Figure 11: An illustrative example of T2T embedding generation

OpenML Datasets

task id: 7592, 9946, 49, 3797, 168911, 190410, 14951, 168912, 146606, 9977, 125920, 146607, 3903, 24, 3735, 3891, 3711, 9971, 167141, 27, 10089, 9965, 146820, 145984, 3485, 146065, 10101, 146047, 146819, 10093, 168338, 9952, 167125, 3731, 3561, 189354, 3917, 43, 3602, 4, 167211, 48, 3954, 9976, 9978, 3779, 3543, 219, 3953, 50, 9957, 168335, 3904, 3620, 3647, 3913, 14954, 146210, 29, 3896, 37, 3739, 145847, 189356, 39, 42, 3902, 3950, 3889, 3918, 145799, 3540, 31, 9910, 9984, 168337, 168868, 167120, 34539, 25, 15, 146206, 14952, 3748, 3686, 3, 54, 190408, 14965, 146818, 168908.