ITC 2/54
Information Technology and Control
Vol. 54 / No. 2/ 2025
pp. 662-681
DOI 10.5755/j01.itc.54.2.39134

Embedding Numerical Features and Meta-Features in Tabular Deep Learning

Received 2024/10/16 | Accepted after revision 2024/12/02

HOW TO CITE: Ma, X., Yao, B. (2025). Embedding Numerical Features and Meta-Features in Tabular Deep Learning. *Information Technology and Control*, 54(2), 662-681. https://doi.org/10.5755/j01.itc.54.2.39134

Embedding Numerical Features and Meta-Features in Tabular Deep Learning

Xingyu Ma, Bin Yao

Department of Computer Science and Engineering, Shanghai Jiao Tong University, Dongchuan Road 800, 200240, Shanghai, China; e-mail: strive4g8@sjtu.edu.cn, yaobin@cs.sjtu.edu.cn

Corresponding author: yaobin@cs.sjtu.edu.cn

Tabular data is ubiquitous in real-world applications, and an increasing number of deep learning approaches have been developed for tabular data prediction. Among these approaches, embedding techniques serve as both a common and essential component. However, the design of tabular embedding paradigms remains relatively limited, and there is a lack of systematic evaluation regarding the performance of many existing methods in specific scenarios. In this paper, we focus on embedding numerical features and meta-features. To enrich the embedding methods for numerical features, we propose an ordering-oriented regularization technique applicable to piecewise linear embeddings, along with an unsupervised feature grouping method to facilitate partial embedding sharing. We demonstrate that these methods contribute to building more efficient and lightweight embedding modules. Importantly, we highlight ordering and sharing as two promising directions in the design of embeddings for numerical features. Additionally, we address several evaluation gaps: we assess the robustness of existing embeddings for numerical features and evaluate a set of general designs separately for data type embeddings and positional embeddings, providing insights into their practical applications and further developments.

KEYWORDS: Embedding, Deep Learning, Tabular Data, Benchmark, Feature Engineering.

1.Introduction

Tabular data, structured with rows as samples and columns as features, constitutes the foundation of many salient industrial applications such as click-through rate prediction [12], online patient monitoring [47], and credit scoring [6]. In the context of tabular data inference, deep learning methods have yet to achieve the remarkable success exhibited in unstructured data domains such as computer vision and natural language processing, often lagging behind tree-based ensemble models [4, 18, 32] in terms

of both predictive accuracy and computational efficiency [3, 10, 36]. However, due to their inherent differentiability and their capability for representation learning, neural networks exhibit numerous advantages over non-deep models including multi-modal learning [24, 35], transfer learning [23, 28], and continual learning [29, 40]. Therefore, there has been an ongoing effort in developing deep learning approaches to break through their performance limits on tabular tasks [9, 14, 16, 42, 49].

Embedding techniques are a common design element in tabular deep learning workflows. As illustrated in Figure 1, the raw tabular data is first transformed into dense, feature-independent embeddings, which are then fed into the primary model to predict the target variable. Applying appropriate embedding methods is effective in enhancing the performance of tabular deep models. Unlike unstructured data such as images and text, tabular data is characterized by feature heterogeneity, with each feature having its distinct source, type, and distribution. This heterogeneity can create an intricate input space that often results in non-smooth target functions, making it particularly challenging for deep models, especially on irregular datasets with long-tail or skewed distributions [11, 27]. In such cases, a sophisticated embedding scheme can be utilized to map the raw feature space into a more homogeneous embedding space, which better aligns with the inductive bias of deep models [2].

Existing literature mostly adopts a prevalent embedding paradigm, which embeds categorical features using lookup tables and numerical features using linear transformations, while few studies have explored dedicated designs for embedding modules. For example, concerning embeddings for numerical features, Gorishniy et al. [8] propose piecewise linear encoding and periodic encoding for application prior to the linear transformation, while Yan et al. [43] use a discretization algorithm to map continuous values into specific magnitude tokens. Similarly, several tabular works [43, 45-46] on large models explore how to combine feature value embeddings with additional embeddings targeting meta-features, such as feature names and data types. However, there is still a lack of established benchmark across different design dimensions, which can confuse practitioners when considering practical applications and further development in tabular embedding domain. Concretely, the two main limitations are identified as follows: (1) There are several underemphasized paradigms in embeddings for numerical features, such as ordering-based regularization and embedding sharing strategies; however, their effectiveness has yet to be convincingly evaluated. Furthermore, this underemphasis can result in certain valuable designs remaining underexplored, leading to suboptimal performance. (2) For each type of meta-features, different studies employ their own fusion designs across various application scenarios. However,

Figure 1

An example of using a deep model with embedding layers for tabular data prediction. The figure illustrates the Titanic dataset, with certain features omitted for simplicity in the presentation.

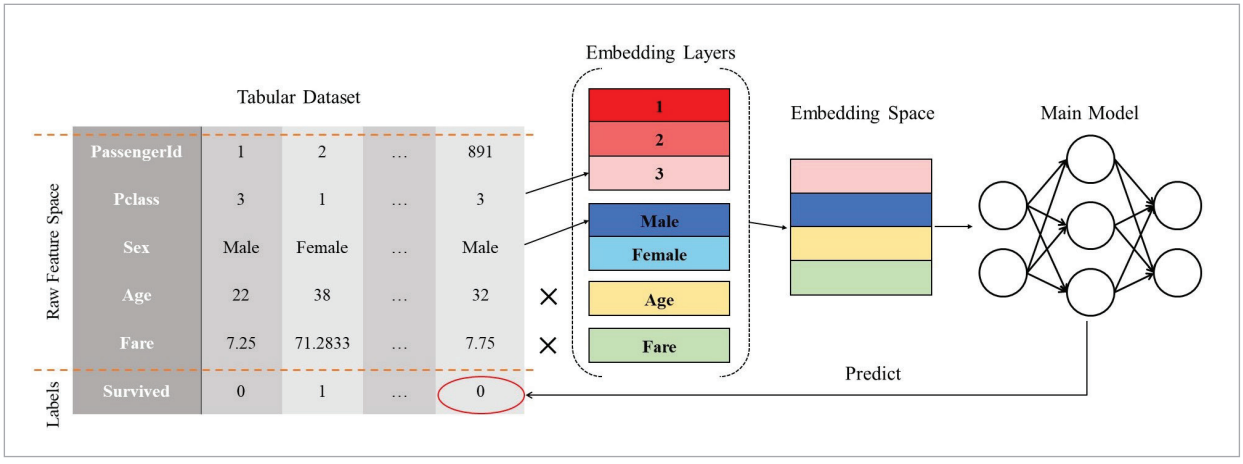
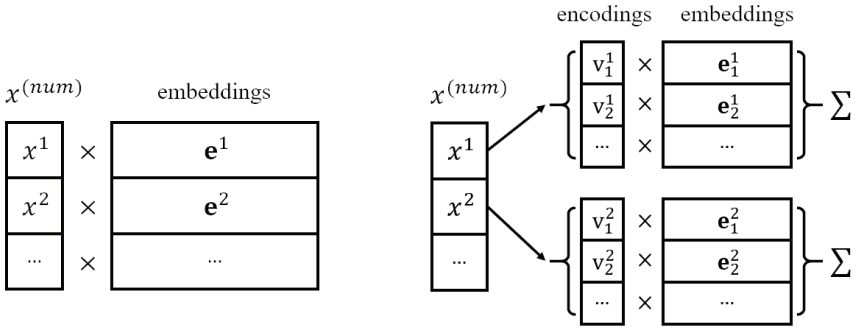


Figure 2
Categorization of embedding techniques for numerical features. Vectorization-based embeddings apply individual linear transformations to the encodings of each numerical feature. x^j represents the j-th numerical feature value from sample $\mathbf{x}$, $\mathbf{v}^j = \left(v_1^j, v_2^j, \cdots\right)$ denotes the encoding vector of x^j, and $\boldsymbol{e}^j$ denotes the embedding vector of x^j.



(a) Vanilla Embeddings (b) Vectorization-Based Embeddings

under the supervised learning setting, there is a lack of comprehensive summaries and systematic evaluations of general fusion choices, making it difficult for researchers to make informed decisions when constructing embedding layers.

To fill these gaps, we conduct an experimental study on embeddings for numerical features and meta-features. (1) Regarding embeddings for numerical features, we evaluate different ordering designs and sharing strategies. First, we introduce the concept of ordering in embedding techniques and provide a formal definition. Based on this, we quantitatively analyze certain embedding techniques that can preserve a degree of ordering. In particular, we propose a cosine-based regularization method applicable to piecewise linear embeddings [8], which can improve the performance of lightweight embeddings that struggle to learn proper ordering. Next, we treat shared embeddings not only as a necessity for specific application requirements but also as a regularization approach that encodes inductive biases. From this perspective, we identify three different embedding-sharing strategies according to varying degrees of sharing. We also propose an unsupervised feature grouping method to realize partial embedding sharing. Our experiments demonstrate that more efficient and lightweight embeddings modules can be obtained through partial or full sharing strategies. Moreover, given that noise is an inevitable and non-negligible perturbation in tabular tasks [37, 44, 50], we address the lack of robustness evaluations for existing embedding techniques for numerical features. We show that piecewise linear embeddings exhibit the strongest noise resistance, whereas vanilla and periodic embeddings [8], despite their initially superior performance, deteriorate significantly as noise levels increase. (2) For embeddings for meta-features, we focus on the designs and evaluations of data type embeddings and positional embeddings. For data type embeddings, we consider two fusion methods: an additive mechanism and a gating mechanism. Likewise, we present a framework that enables positional embeddings to model feature-specific information in either the embedding space or the latent representation space for encoder-like models. Through fair comparisons of different variants, we demonstrate that various positional embedding designs can lead to significant performance gains, while existing data type embeddings offer no clear benefit.

In summary, we outline our main contributions as follows:

1 We demonstrate that applying ordering-oriented regularization techniques to ordering-deficient embeddings for numerical features can improve model performance.

2 We show that creating separate embeddings for each numerical feature is not always the optimal

approach; partial or full sharing of embeddings can often result in more efficient and lightweight embedding modules.

3 We present the performance of different embeddings for numerical features under varying noise conditions, highlighting their robustness.

4 We demonstrate that, in general tabular tasks, data type embeddings with additive or gating mechanisms provide limited improvements, while positional embeddings yield noticeable gains.

2. Related Works

2.1. Tabular Deep Learning

In the early stages, a substantial body of tabular deep learning research concentrated on using neural networks to mimic the inference process of tree-based models to achieve better accuracy [17, 19, 31]. Over time, inspired by the tremendous success of deep learning in unstructured data domains, researchers have introduced many powerful model architectures and training paradigms to expand the arsenal of tabular deep learning. Notably, the Transformer has emerged as a popular architecture choice [3]. For instance, Gorishniy et al. [10] propose the FT-Transformer, which employs Transformer as the backbone and has been demonstrated competitive performance across various datasets. Similarly, Kossen et al. [22] introduce NPT, which leverages self-attention mechanism to capture dependencies both between features and between samples, enabling the model to make use of other samples for inference. These Transformer-based models rely on embedding techniques, which transform scalar feature values into learnable high-dimensional vectors. Moreover, inspired by the success of language large models (LLMs), many works have sought to pretrain a general-purpose large model across multiple datasets [7, 13, 46, 49]. To this end, they must consider how to establish a featurization protocol that is both flexible in handling diverse features and effective in practice. Specifically, they need to address how to embed and serialize features with semantics and quantities specific to each dataset before feeding them into the Transformer-stacked backbone. The challenges include, but are not limited to: (1) How to embed numerical features, since existing LLMs are notoriously poor at understanding numerical concepts [33]. (2) How to design a tokenization schema that is more aligned with the tabular structure [46]. Overall, optimizing embedding techniques can benefit the current research in tabular deep learning.

2.2. Embeddings for Tabular Data

Embedding methods are employed in many deep learning works within the tabular data domain. For most of these works, the go-to solution is embedding categorical features using lookup tables and numerical features using linear transformations. TabTransformer [15] opts to exclude continuous features and focuses solely on embedding categorical features. In contrast, Gorishniy et al. [8] show that embedding numerical features can benefit both Transformer-based and MLP-based models, and they also introduce two competitive embedding methods for numerical features. Borisov et al. [2] take a unique path by mapping the original features into the latent space of pretrained tree-based ensemble models, allowing deep models to improve predictive accuracy by utilizing these more homogeneous feature embeddings. In addition to the conventional embedding of feature values, many works have also proposed embedding designs targeting certain additional data information. For example, some works embed data types and then fuse them with feature value embeddings [22, 45]. Similarly, some works use "positional embeddings" to model feature-specific information, such as the additive bias embeddings [10, 49]. In the context of large tabular models, a common practice is to combine feature values with feature descriptions for embeddings. CM2 [46], Unitab [45], and TP-BERTa [43] aggregate the embedding sequences within each sample and feature, rather than simply forming sequences as in many other works, thereby aiding the model to understand tabular data at the cell level. In particular, TP-BERTa introduces a discretization-based token embedding method for numerical features, and these embeddings are constrained to preserve the relative magnitude of the feature values via a specialized regularization approach. The method is also applicable to non-LLMs. However, in the context of supervised tabular tasks without the use of large models, there remains considerable scope for further exploration of available embedding techniques. It is also worth noting that embedding techniques are a relatively resource-in-

tensive featurization method, and in some cases, their cost can outweigh the benefits, particularly when dealing with a large number of features.

3. Embedding Approaches for Tabular Data

In this section, we begin by describing standard supervised learning on tabular data and the role of embedding techniques in addressing these tasks. Next, we summarize existing embedding methods for numerical features. Then we analyze various ordering designs and explore different embedding-sharing strategies. Finally, we present various designs for data type embeddings and positional embeddings.

3.1. Preliminaries

We address tabular problems in the standard supervised learning setting. Given a tabular task with ℓ-dimensional input variables $X \in \mathrm{X} \subseteq \mathrm{R}^{\ell}$ and target variable $Y \in \mathrm{Y} \subseteq \mathrm{R}$, the goal is to develop a model θ to map from the input space to the target space, i.e., $f_{\theta}: \mathrm{X} \to \mathrm{Y}$. In the context of supervised learning, we have a labeled dataset $\mathrm{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, consisting of n instances i.i.d. sampled from the joint distribution $P(X, Y)$. In practice, the dataset is usually split into three disjoint subsets: $\mathrm{D} = \mathrm{D}_{\text{train}} \cup \mathrm{D}_{\text{val}} \cup \mathrm{D}_{\text{test}}$, where $\mathrm{D}_{\text{train}}$ is used for training, D_{val} for hyperparameter tuning and early stopping, and D_{test} for final evaluation on unseen data. With a loss function L guiding the optimization, the learning process of model can be formulated in Equation (1).

$$\theta^{*} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{train}}} \mathcal{L}\left(f_{\theta}(\mathbf{x}), y\right). \tag{1}$$

Let x^j denote the j-th feature of sample $\mathbf{x}(1 \le j \le \ell)$. For those solutions using embeddings, the predictive function can be rewritten as $f_{\theta}(\mathbf{x}) = g\left(\mathbf{z}^1, \mathbf{z}^2, \cdots, \mathbf{z}^{\ell}\right)$, where $\mathbf{z}^j = \phi\left(x^j\right) \in \mathrm{R}^d$ is the d-dimensional vector transformed from x^j via the embedding function $\phi(\cdot)$, and $g(\cdot)$ denotes the decision function of subsequent layers. In a nutshell, these models follow the "embedding-backbone" paradigm, with the embeddings of each feature computed independently before mixing them up in the backbone. Currently, the majority of backbone architectures can be categorized into three types: MLP-like, ResNet-like, and Transformer-like architectures [10].

In this work, we focus on conventional supervised learning for tabular data, where a single model is trained from scratch on a single labeled dataset for inference. This indicates that we exclude large models and embeddings pretrained on cross-datasets; however, we adopt some general embedding designs proposed in tabular large model studies, which are simple enough to be affordable for standard models. Note that, for the sake of simplicity, we use a uniform embedding dimensionality for all features, even though customizing the dimensionality for each feature could further enhance performance and reduce memory costs [21, 34].

3.2. On Embeddings for Numerical Features

3.2.1. Revisiting Existing Methods

Numerical features often serve as a crucial component of tabular data, and existing research has proposed various embedding methods that significantly boost model performance. Let J_{num} denote the index set of numerical features. As shown in Figure 2, we group existing embedding schemes for numerical features into two distinct categories: vanilla embeddings and vectorization-based embeddings. The first category is adopted in most works, where the scalar numerical feature is element-wise multiplied with an embedding vector. The second category first vectorizes the scalar through a multi-dimensional encoding strategy and then multiplies the resulting vector by a learnable parameter matrix. For both categories, a bias term can optionally be added depending on implementation choices. Essentially, the fundamental difference between the two approaches lies in whether a multi-dimensional encoding is applied before performing the linear transformation. Furthermore, based on different encoding strategies, we list all available embeddings belonging to the second category, including periodic embeddings, piecewise linear embeddings (PLE), and token embeddings [43].

For a numerical feature indexed by j, $\mathbf{e}^j$ denotes the embedding parameters and $\mathbf{E}^j$ denotes its final embedding. If necessary, the bias term $\mathbf{e}_0^j$, the encodings $\mathbf{v}^j = \left(v_1^j, v_2^j, \cdots, v_T^j\right)$, and the T+1 pre-specified bin

edges $\{b_k^j\}_{k=1}^{T+1}$ are provided. Formally, these embedding methods can be described as follows.

1 Vanilla embedding. The embedding process is presented in Equation (2).

$$\mathbf{E}^j = x^j \cdot \mathbf{e}^j + \mathbf{e}_0^j \,. \tag{2}$$

2 Piecewise linear embedding (PLE). The encoding process is shown in Equation (3), and the embedding process is presented in Equation (4).

$$v_t^j = \begin{cases} 0, \text{ if } x^j < b_t^j \text{ and } t > 1, \\ 1, \text{ if } x^j \geq b_t^j \text{ and } t < T, \\ \dfrac{x^j - b_{t-1}^j}{b_t^j - b_{t-1}^j} \qquad \text{otherwise.} \end{cases} \tag{3}$$

$$\mathbf{E}^j = \sum_{t=1}^{T} v_t^j \mathbf{e}_t^j + \mathbf{e}_0^j \,. \tag{4}$$

3 Periodic embedding. The encoding process is shown in Equation (5), and the embedding process is the same with PLE (see in Equation (4)).

$$\begin{cases} \mathbf{p}^j = \left(2\pi c_1^j x^j, 2\pi c_2^j x^j, \cdots, 2\pi c_{T/2}^j x^j\right) \\ \mathbf{v}^j = \text{concat}\left[\sin\left(\mathbf{p}^j\right), \cos\left(\mathbf{p}^j\right)\right] \end{cases}, \tag{5}$$

where T is always even, and c_i^j is initialized from a normal distribution $\mathrm{N}(0,\sigma^2)$ with hyperparameter σ.

4 Token Embedding. The encoding process is shown in Equation (6), and the embedding process is written in Equation (7).

$$v_t^j = \begin{cases} 1, \text{ if } b_t^j \leq x^j < b_{t+1}^j, \\ 0, \qquad \text{otherwise.} \end{cases} \tag{6}$$

$$\mathbf{E}^j = x^j \left(\sum_{t=1}^{T} v_t^j \mathbf{e}_t^j \right). \tag{7}$$

Importantly, both PLE and token embedding necessitate a prior application of a binning algorithm, typically using bin edges derived either from quantiles or the "C4.5 Discretization" algorithm [20], to segment the continuous values. Concretely, to obtain T bins for a numerical feature indexed by $j\left(j \in J_{\text{num}}\right)$, we need to generate a sequence of $T+1$ edges $\{b_k^j\}_{k=1}^{T+1}$ such that the t-th bin can be represented as $B_t^j = \left[b_t^j, b_{t+1}^j\right)\left(1 \leq t \leq T\right)$.

$$b_k^j = \text{Quantile}_{\frac{k}{T+1}}\left(\{x^j\}_{\mathbf{x} \in \mathcal{D}_{\text{train}}}\right) \tag{8}$$

$$b_k^j = \begin{cases} \min\limits_{\mathbf{x} \in \mathcal{D}_{\text{train}}} x^j, \text{ if } k = 1, \\ \tau_{k-1}^j, \text{ if } 1 < k \leq T, \\ \max\limits_{\mathbf{x} \in \mathcal{D}_{\text{train}}} x^j, \text{ if } k = T+1. \end{cases} \,. \tag{9}$$

To achieve this, the former (quantile-based binning) computes $T+1$ equally spaced quantiles between 0 and 1 as in Equation (8), while the latter (target-aware binning) selects thresholds by recursively maximizing the impurity gain of splitting ranges for target values in a greedy manner, i.e., the splitting values of internal nodes in a decision tree trained on $\left\{\left(x^j, y\right)\right\}_{\mathbf{x} \in \mathrm{D}_{\text{train}}}$. As a supervised approach, the target-aware binning only ensures that the number of thresholds does not exceed $T-1$ by limiting the maximum number of leaf nodes to T. Let $\left(\tau_1^j, \tau_2^j, \cdots, \tau_{T-1}^j\right)$ represent the thresholds in ascending order and the target-aware binning method is formalized in Equation (9). In addition, token embeddings set $b_1^j = -\infty$ and $b_{T+1}^j = +\infty$ to ensure that any value can fall into a bin. Notably, the "token embeddings" concept is not directly introduced in the original paper. We decompose its proposed Relative Magnitude Tokenization (RMT) technique into two parts: the first part is the embedding approach, which we refer to as "token embeddings" since RMT first discretizes the numerical values then embeds the resulting indices as tokens; the other part is the magnitude-aware regularization term, which will be analyzed in the following discussion about ordering. In our settings, we make adjustments to allow for per-feature embeddings and quantile-based binning, and determine the recommended configuration for standard tabular tasks in Section 4.2.

3.2.2. Ordering Analysis

Numerical features possess an inherent order, which forms the foundation for ranking and arithmetic operations. For instance, in the task of house price prediction, it is often observed as a common heuristic that larger houses tend to have higher prices. For the embedding-based solutions, if the relative magnitude of embeddings for "house area" feature can be modeled explicitly in a certain way, it might help models better understand the concept of house size and thus capture the underlying rule. Therefore, the effectiveness of designing embeddings for numerical features to preserve their inherent ordering is worth evaluating. It is also worth noting that the ordering property is not exclusive to numerical features but also applies to certain categorical features, such as education level and satisfaction.

To further investigate this, we perform a quantitative analysis. Mathematically, we can define the ordering of embeddings for numerical features as the consistent relative magnitude relationships such that if any triplet of the original feature values follows $x_1 < x_2 < x_3$, the corresponding embedding vectors should also satisfy $\Delta\rho = \rho\left(\phi(x_1), \phi(x_3)\right) - \rho\left(\phi(x_1), \phi(x_2)\right) > 0$, where $\rho(\cdot)$ is an arbitrary distance function. Note that we omit the feature index j for simplicity. For a more in-depth quantitative analysis of certain ordering designs, we use the Euclidean distance as the default distance function unless otherwise specified, and denote t_1, t_2, t_3 as the bin indices corresponding to x_1, x_2, x_3 for binning-based embeddings $\left(t_1 \le t_2 \le t_3\right)$.

1 Vanilla Embeddings.
$$\Delta\rho = \left(x_1 - x_3\right)\mathbf{e} - \left(x_1 - x_2\right)\mathbf{e} = \left(x_3 - x_2\right)\mathbf{e} > 0$$
where $\mathbf{e}$ is a non-zero embedding. This ordering also holds for other distance functions, such as norm-based distance metrics.

2 RMT Regularization. RMT introduces a regularization loss as in Equation (10), where the distance function computes the Euclidean distance between the outputs of $\mathbf{e}_{t_1}$ and $\mathbf{e}_{t_2}$ via a linear layer followed by layer normalization. If $t_1 = t_2 = t_3$, the case of distance comparison is the same as vanilla embedding. However, since the regularization maintains ordering at the token level, the ordering between more fine-grained embeddings for numerical features, which are scaled by the original feature values, cannot be guaranteed.

3 Piecewise Cosine Regularization (PCR). For the original PLE, the case is the same as vanilla embedding if $t_1 = t_2 = t_3$, otherwise extra constraints are required to ensure the strict ordering. Specifically, this method introduce a sufficient constraint that the cosine value between each pair of bin embedding vectors must be positive. To elucidate the principles behind the cosine-based penalties, we begin by presenting two foundational propositions.

Proposition 1: Given two vector sets $\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n\}$ and $\{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_m\}$, where $\cos\left(\mathbf{p}_i, \mathbf{q}_j\right) > 0$ for all i and j, the cosine between their respective positive linear combinations remains positive. Formally,
$$\operatorname{sign}\left(\cos\left(\sum_{i=1}^{n}\alpha_i\mathbf{p}_i, \sum_{j=1}^{m}\beta_j\mathbf{q}_j\right)\right)$$
$$= \operatorname{sign}\left(\sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_i\beta_j\left(\mathbf{p}_i \cdot \mathbf{q}_j\right)\right) > 0,\ \forall\alpha_i, \beta_j > 0.$$

Proposition 2: Given two vectors $\mathbf{q}$ and $\mathbf{p}$ where $\cos(\mathbf{q}, \mathbf{p}) > 0$, the L_2 norm of their sum exceeds that of either individual vector. Specifically, $\mathbf{q} + \mathbf{p} = \sqrt{\mathbf{q}^2 + \mathbf{p}^2 + 2\mathbf{q}\mathbf{p}\cos(\mathbf{q}, \mathbf{p})} > \mathbf{q}$ Next, we denote $\Delta\mathbf{e}_{12}$ as the embedding margin between x_1 and x_2. Formally, if $t_1 \neq t_2$, $\Delta\mathbf{e}_{12} = \left(1 - v_{t_1}\right)\mathbf{e}_{t_1} + \sum_{t=t_1+1}^{t_2-1}\mathbf{e}_t + v_{t_2}\mathbf{e}_{t_2}$, otherwise $\Delta\mathbf{e}_{12} = \left(v_{t_2} - v_{t_1}\right)\mathbf{e}_{t_1}$. A similar formulation applies to $\Delta\mathbf{e}_{13}$ and $\Delta\mathbf{e}_{23}$. Furthermore, we can decompose $\Delta\mathbf{e}$ as $\Delta\mathbf{e}_{13} = \Delta\mathbf{e}_{12} + \Delta\mathbf{e}_{23}$. According to proposition 1, given that any pair of bin embeddings $(\mathbf{e}_i, \mathbf{e}_j)$ satisfies $\cos(\mathbf{e}_i, \mathbf{e}_j) > 0$, we have $\cos(\Delta\mathbf{e}_{12}, \Delta\mathbf{e}_{23}) > 0$. Then, by proposition 2, we obtain $\Delta\rho = \Delta\mathbf{e}_{13} - \Delta\mathbf{e}_{12} > 0$. Based on this proof, we propose a pairwise regularization loss, as shown in Equation (11), to encourage the cosine values between different pairs of bin embeddings to remain positive. Note that in practice we introduce the hyperparameter $\tau_{\text{reg}} > 0$ to penalize embedding pairs with low-magnitude cosine values, preventing degradation into vanilla embeddings.

Overall, vanilla embeddings inherently possess an ordering property, while two regularizations can be applied to token embeddings and PLE respectively to achieve ordering. Both regularization techniques consist of supervised loss and regularization loss as expressed in Equation (12), where $\lambda \in \mathrm{R}$ acts as a weighting term.

$$\mathcal{L}_{\text{reg}}^{\text{RMT}} = \max(\rho\left(\mathbf{e}_{t_1}, \mathbf{e}_{t_2}\right) - \rho\left(\mathbf{e}_{t_1}, \mathbf{e}_{t_3}\right) + \frac{|t_1 - t_3| - |t_1 - t_2|}{T}, 0) \quad (10)$$

$$\mathcal{L}_{\text{reg}}^{\text{PCR}} = \max(\tau_{\text{reg}} - \cos\left(\mathbf{e}_i, \mathbf{e}_j\right), 0), \forall 1 \le i, j \le T \quad (11)$$

$$\mathcal{L} = \mathcal{L}_{\text{sup}} + \lambda \mathcal{L}_{\text{reg}}. \quad (12)$$

3.2.3. Embedding-Sharing Strategies

In many cases, it is necessary for different numerical features to share their embeddings, such as when a lightweight embedding module is required [9] or when a model is trained across datasets [43]. Formally, $\mathbf{e}_j = \mathbf{e}, \forall j \in J_{\text{num}}$. Furthermore, we can consider different embedding-sharing strategies from the perspective of feature grouping: the two paradigms can be described uniformly as features within the same group sharing embeddings, where the private strategy considers each feature as an individual group and the fully shared one treats the entire feature set J_{num} as a single group. Naturally, between the two extremes of sharing degree, there exists a third paradigm, which groups a subset of features. Embedding each feature separately offers greater flexibility, while shared embeddings typically require fewer embedding parameters, a trend that becomes more pronounced as the number of numerical features increases. However, this sharing introduces inductive biases, causing the model to treat features within the same group more similarly, as reflected in representations determined primarily by value magnitudes and overlooking semantic differences. Consider an extreme case where the target variable is a linear combination of all features with identical weights, indicating that each feature has exactly the same influence on predicting the target value. At this stage, there is no need to distinguish between the semantic meanings of features, as only their values matter, and sharing embeddings will not confuse models. Intuitively, it becomes more reasonable to share features that exhibit a more homogeneous role in interactions with other features. Motivated by this insight, we propose an unsupervised method for feature grouping, which can automatically obtain feature groups without relying on prior knowledge about the semantic meaning of features. Specifically, we first compute the Spearman correlation matrix of all numerical features, and then apply the KMeans algorithm [25] to this matrix to obtain a specified number of clusters, which correspond to groups of feature indices. In a nutshell, this approach delineates the roles of different features within the dataset through the computation of correlation coefficients, subsequently employing clustering techniques to categorize more homogeneous features into distinct groups, thereby facilitating application to datasets with an arbitrary number of features. The choice of using Spearman over other correlation coefficients, such as Pearson or Kendall, is attributed to its relatively simple computation and its ability to handle non-linear relationships and outliers, thus effectively meeting the requirements of most tabular tasks. However, this method also has a notable limitation: it is contingent upon the reliability of the correlation coefficients, which may be affected by data quality and the presence of confounding variables.

3.3. On Embeddings for Meta-Features

Apart from feature values, tabular data contains numerous auxiliary pieces of information that are essential for understanding the nature and semantics of the features. For example, a feature name like “income” may imply a relationship with “spending habits”. Therefore, many embedding methods for meta-features have been proposed, and here we focus on the designs of data type embeddings and positional embeddings. Note that we exclude the embeddings for feature descriptions for two reasons: first, textualizing the embeddings for feature descriptions tends to generate a long sequence of tokens, which generally requires processing by large models, a resource-expensive approach that is not feasible in many cases; second, the method is also unsuitable for datasets with anonymized features or non-descriptive feature codes, such as Arabic symbols.

3.3.1. Data Type Embeddings

In general, features in tabular datasets can be categorized into three types: numerical, categorical and binary (a special case of categorical features containing only two values). Most embedding paradigms differentiate feature types solely by the approaches used for embedding feature values. However, some specialized designs introduce additional

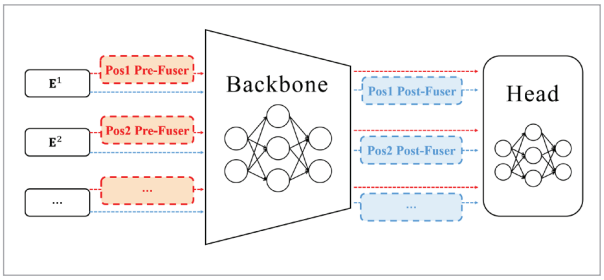
embedding modules to explicitly model data type information, integrating them with feature value embeddings through additive or gating mechanisms. To elaborate, given the data type embedding $\mathbf{E}^{\mathrm{dt}_j}$ for the j-th feature drawn from a lookup table, the addition mechanism can be expressed as $\mathbf{E}^j+\mathbf{E}^{\mathrm{dt}_j}$. With respect to the gate mechanism, we introduce a lightweight version, formulated as $\mathbf{E}^j+\mathrm{Sigmoid}\left(\mathbf{E}^{\mathrm{dt}_j}\cdot\mathbf{W}\right)$, where $\mathrm{W}\in\mathrm{R}^{\mathrm{d}\times\mathrm{d}}$ is a trainable parameter matrix, and the dimensionality of $\mathbf{E}^j$ and $\mathbf{E}^{\mathrm{dt}_j}$ is equal.

3.3.2. Positional Embeddings

The traditional positional embeddings in the NLP domain, which are designed to introduce sequential information into models, do not align with the permutation-invariant nature of tabular data. Here, the positional embeddings refer to feature-specific embeddings uniquely identified by their indices. These embeddings are expected to capture feature-dependent information, helping models distinguish between different features and understand their relationships. Following the description, the individual bias embeddings of each feature can be regarded as a form of additive positional embedding. Moreover, we propose a framework that incorporates various positional embedding designs, as illustrated in Figure 3. In this framework, there are two choices to fuse the positional embeddings with the main representations, including addition and concatenation. For the Transformer-based backbone, the positional embeddings are additionally allowed to operate in the latent space of the backbone outputs. Assume that the main representations are d-dimensional and the positional embeddings are d_{pos}-dimensional, d equals d_{pos} if fused before passing into the backbone; otherwise, the representations first need to be projected via a linear layer $\mathrm{R}^d\rightarrow\mathrm{R}^{d_{\mathrm{pos}}}$. Therefore, the common bias terms of embeddings for each feature can be viewed as the pre-add variant.

Figure 3

A framework for diverse positional embeddings designs. Only when the length of the output representations from the backbone equals the number of features, such as in a Transformer-based backbone, can the fusion modules operate in the latent space. $\mathbf{E}^j$ denotes the embeddings of the j-th feature.



4. Experiment and Analysis

In this section, we commence by evaluating token embeddings equipped with different simple differentiable modules to determine a recommended configuration for practical applications. Subsequently, we compare diverse embedding schemes across various datasets, aiming to answer four key questions within the standard learning setting: (1) How do different designs for preserving ordering in embeddings for numerical features affect model performance? (2) Can partial or full sharing of embeddings for numerical features lead to further improvements in performance and cost efficiency? (3) How do existing embeddings for numerical features perform under varying noise conditions? (4) Can data type embeddings and positional embeddings each consistently provide significant improvements? Finally, we compare the best scores of different embedding methods under various maximum rounds of hyperparameter search, highlighting the differences in training costs required to fully unleash their potential.

4.1. Experiment Setup

4.1.1. Datasets

We utilize a selection of popular benchmarks from prior literature on tabular deep learning [9, 27]. To be specific, these eight datasets are medium- and large-scale, each containing at least 10,000 objects, representing diverse tabular problems with regression and classification targets. Importantly, we leverage four of them that consist solely of numerical features for experiments on the evaluation criteria, ensuring that all performance impacts are derived from embeddings for numerical features. When demonstrating the generality of effectiveness, we use the full datasets. The main statistics of the full datasets are listed in Table 1 and more details are provided in Appendix A.2.

Table 1
Dataset statistics. We use AUC (Area Under the ROC Curve) as the metric for binary classification tasks and use RMSE (Root Mean Square Error) for regression tasks.

Dataset	#Object	(#Num, #Bin, #Cat)	Metric
EL	16,599	(18, 0, 0)	AUC.
CA	20,640	(8, 0, 1)	RMSE
HO	22,784	(16, 0, 0)	RMSE
AD	48,842	(6, 1, 7)	AUC.
DI	53,940	(6, 0, 3)	RMSE
HI	98,050	(28, 0, 0)	AUC.
MI	130,064	(50, 0, 0)	AUC.
BL	166,821	(4, 2, 3)	RMSE

4.1.2. Data Preprocessing

We establish a common preprocessing protocol for a fair comparison between models. For each dataset, we treat nominal features with a cardinality of two as binary features and the rest as categorical features. We randomly sample 20% of the entire dataset as D_{test}, then randomly select 20% of the remaining data as D_{val}. To handle missing values, we impute each numerical feature with the mean and treat missing values in categorical features as a new category. Additionally, we follow the normalization strategies in the work of Gorishniy et al. [10], which apply quantile-based transformation to numerical features and z-score normalization to labels in regression tasks.

4.1.3. Tuning and Evaluation

To fully explore the potential of each model, we conduct thorough hyperparameter searches for individual tasks. Specifically, we use the TPEsampler in the Optuna library [1] to run 320 hyperparameter optimization (HPO) trials for each model. In each trial, we run up to 1000 epochs, applying early stopping with a patience of 16, and use the AdamW optimizer [26], applying weight decay to all layers except bias and normalization layers. We minimize mean squared error for regression tasks and cross-entropy for classification tasks, and adopt the default hyperparameter space from the corresponding works. The details of tuning configurations are outlined in Appendix A.3. For each tuned hyperparameter configuration, we retrain 10 models with different random seeds and report their scores on the test set. Additionally, to assess the statistical significance of performance differences, we apply the one-sided Wilcoxon test [5] with $p < 0.05$ for pairwise evaluations.

4.1.4. Models

The models follow the "embedding-backbone" paradigm, with three types of backbones—MLP, ResNet, and Transformer—which have been shown to be competitive [10]. For all models, an embedding layer with a bias term is used to process categorical features. We distinguish the embedding layers of different models by their embeddings for numerical features. For simplicity, we follow the abbreviations introduced by Gorishniy et al. [8], with some redefinitions and additional symbols, as shown in Table 2. Unless otherwise specified, different embeddings for numerical features use the recommended configuration , while the configurations of token embeddings will be studied in Section 4.2.

Table 2
Embedding notation and definitions.

Notation	Definition
L^0	A linear layer without a bias term
L	A linear layer with a bias term
R	ReLU activation function
V	No encoding (corresponding to vanilla embeddings)
P	Periodic encoding
QP	Quantile-based piecewise linear encoding
TP	Target-aware piecewise linear encoding
QT	Quantile-based token encoding
TT	Target-aware token encoding

4.2. Simple Differentiable Models Stop Token Embeddings

Aside from the final multiplication operation, the original token embeddings first discretize each numerical value into its bin index and then directly use their corresponding embeddings, which is equivalent to applying a linear transformation without a bias term on the one-hot-like encoding (see Section 3.2.1). In line with Gorishniy et al.

[8], we evaluate different configurations of token embeddings, where the encodings are followed by various differentiable modules consisting of linear layers and ReLU activation functions. Some of the results are presented in Table 3, while the full dataset is provided in Appendix 2.

We analyze the experimental data by isolating the effects of different choices of embedding modules, binning algorithms, and backbone architectures, while keeping other components constant. We summarize the main conclusions: (1) In terms of performance, the LRLR configuration demonstrates a significant advantage, while the others remain relatively comparable. However, despite its performance superiority, the heavier LRLR setup incurs high overhead, making it less suitable for scenarios requiring lightweight embedding modules. For the other configurations, their rankings are similar in most cases, except under the ResNet-TT conditions, where LR and L achieve noticeably second-best performances, respectively (see Appendix B). Hence, the L^0 configuration is recommended as the default choice for tasks requiring a lighter embedding module. (2) There is no clear preference between QT and TT. They perform similarly on most tasks, they perform on par in most cases, with few exceptions: QT performs better under the Transformer backbone on the AD dataset, while TT performs better under the ResNet backbone on the HO dataset (see Appendix B). (3) There is no clear preference between different backbones. They perform similarly on most tasks, with the exception of the MLP consistently underperforming on the CA dataset.

4.3. Ordering Designs of Embeddings for Numerical Features

To investigate the ordering property of embeddings for numerical features, we begin by comparing the performance of models with and without ordering-based regularization methods. Specifically, we evaluate token embeddings with and without RMT regularization, as well as the lightweight version of PLE with and without PCR. As shown in Table 4, the main takeaways include that (1) RMT regularization proves ineffective in standard supervised learning tasks. In most cases, it even undermines the performance of the original token embeddings, with particularly adverse effects when applied to the Transformer backbone. These performance issues are not due to insufficient hyperparameter tuning, as we extend the optimization rounds to 640 on the EI and HO dataset with the Transformer backbone, without observing any notable improvement. (2) PCR can occasionally provide improvements; however, these gains are not consistently maintained, as seen on the HI dataset, where it can even lead to performance degradation.

We scrutinize the cases where PCR has a positive impact, and the loss curves in Figure 4 suggest that PCR has the potential to accelerate convergence and prevent overfitting. Furthermore, we apply PCR to PLE with the recommended configuration. As presented in Table 4, the models with and without PCR perform nearly equally. This raises a key question: do we really need explicit constraints to achieve the ordering of embeddings? To address this, we con-

Figure 4

The loss curves in PCR-effective cases. The subplots (a) and (b) show the results of MLP-QP-L^0 on the HO dataset with/without PCR, respectively. Similarly, the subplots (c) and (d) show the results of Transformer-QP-L^0 on the EL dataset with/without PCR. The red curves represent validation losses, and the blue curves represent training losses.

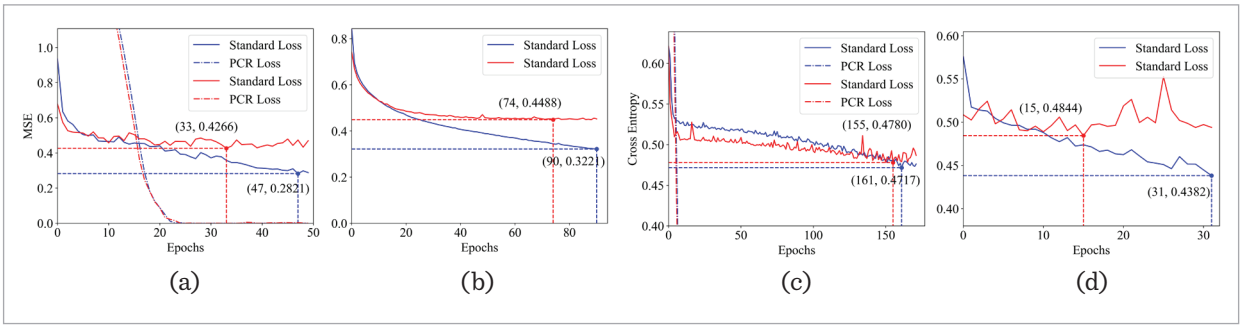


Figure 5

The t-SNE [38] visualization of segment embeddings for the first feature from models trained on the HO dataset with seed=0. The subplots (a) and (b) correspond to MLP-QP-L^0 without/with PCR, respectively. The subplots (c) and (d) correspond to MLP-QP-LR without/with PCR, respectively. The color bar shows the segment IDs aligned with their relative magnitudes.

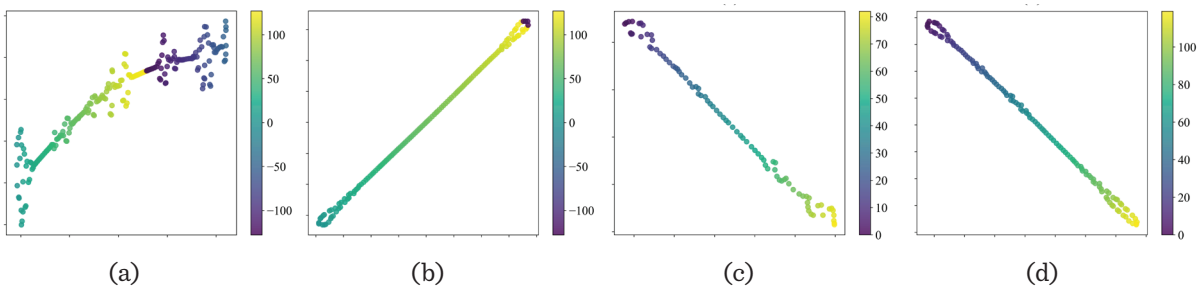


Table 3

Results of models using quantile-based token embeddings with different differentiable modules. The averaged metric values are reported, with standard deviations provided in Appendix B. Comparisons are made within each group. As a result, the top results are in bold, and the averaged ranks are reported. Notations: ↑indicates AUC, ↓indicates RMSE.

Model	EL↑	CA↓	HO↓	AD↑	DI↓	HI↑	MI↑	BL↓	Avg. Rank
MLP-QT-L^0	**0.6987**	0.5024	0.3431	**0.7788**	0.5426	0.7203	0.9379	0.3542	2.0 ± 0.7
MLP-QT-L	0.6929	0.5111	**0.3346**	0.7759	0.5436	0.7187	**0.9388**	0.3540	2.1 ± 0.9
MLP-QT-L^{0}R	0.6769	0.5197	**0.3374**	**0.7775**	0.5411	0.7178	**0.9381**	0.3543	2.2 ± 1.2
MLP-QT-LR	0.6765	0.5094	0.3370	**0.7816**	0.5446	0.7216	0.9372	**0.3535**	2.1 ± 0.8
MLP-QT-LRLR	**0.7000**	**0.4873**	0.3372	**0.7807**	**0.5367**	**0.7233**	0.9370	**0.3534**	1.2 ± 0.4
ResNet-QT-L^0	0.6961	0.4858	**0.3359**	**0.7809**	0.5413	**0.7264**	**0.9392**	0.3541	1.5 ± 0.5
ResNet-QT-L	**0.7049**	0.4876	**0.3356**	**0.7747**	0.5392	0.7267	**0.9401**	0.3542	1.5 ± 0.5
ResNet-QT-L^{0}R	0.7013	0.4885	**0.3354**	**0.7769**	**0.5326**	0.7265	**0.9387**	0.3542	1.6 ± 0.7
ResNet-QT-LR	0.6971	0.4865	0.3400	**0.7779**	**0.5360**	0.7265	**0.9401**	**0.3534**	1.5 ± 0.5
ResNet-QT-LRLR	**0.7056**	**0.4710**	0.3385	**0.7751**	0.5405	**0.7285**	**0.9391**	**0.3534**	1.2 ± 0.4
Transformer-QT-L^0	0.6904	0.4903	0.3473	**0.7820**	**0.5368**	0.7206	**0.9401**	0.3540	1.6 ± 0.7
Transformer-QT-L	0.6811	**0.4877**	0.3436	**0.7744**	0.5489	**0.7233**	**0.9404**	**0.3553**	1.6 ± 0.7
Transformer-QT-L^{0}R	**0.6905**	0.4910	**0.3339**	0.7710	**0.5392**	**0.7239**	0.9377	0.3600	1.6 ± 0.7
Transformer-QT-LR	**0.7001**	0.4891	0.3380	0.7708	0.5470	**0.7257**	0.9377	**0.3550**	1.6 ± 0.5
Transformer-QT-LRLR	**0.6953**	**0.4844**	**0.3333**	**0.7799**	0.5504	**0.7241**	**0.9397**	**0.3621**	1.1 ± 0.3

duct an in-depth analysis from two perspectives: (1) we visualize some embedding results for each segment across different models for the first numerical feature. As shown in Figure 5, even without bias terms, PLE still learn a degree of ordering. However, this ordering is limited, as the arrangement of points does not strictly follow the ideal color transition, and a few points deviate from the principal pattern. Adding differentiable layers enables the embeddings to learn a more desirable ordering, with adjacent points following a more orderly progression along the color spectrum. At the same time, the regularization enforces stricter alignment, as reflected in the more concentrated point distributions. (2) We

Table 4

Results of models applying and not applying ordering-based regularizations to corresponding embeddings. Pairwise comparisons are made between models with and without the application of regularizations, and the better results are in bold. Here, we only present the results for embeddings using quantile-based binning, while the results with target-aware binning are provided in Appendix B. Specifically, the last two groups of data illustrate the effect of applying PCR to well-configured PLE. Notations follow Table 3.

Model	EL↑	HO↓	HI↑	MI↑
MLP-QT-L^0	**0.6987**	0.3431	**0.7203**	**0.9379**
MLP-QT-L^0 (w/ RMT reg.)	0.6908	**0.3376**	**0.7204**	**0.9386**
Transformer-QT-L^0	**0.6904**	**0.3473**	**0.7206**	**0.9401**
Transformer-QT-L^0(w/RMT reg.)	0.5000	0.5141	0.5430	0.6368
MLP-QP-L^0	**0.6869**	0.3383	**0.7230**	0.9365
MLP-QP-L^0 (w/ PCR)	**0.6891**	**0.3329**	0.7162	**0.9389**
Transformer-QP-L^0	0.6845	**0.3494**	**0.7275**	0.9368
Transformer-QP-L^0 (w/ PCR)	**0.6922**	**0.3467**	0.7221	**0.9397**
MLP-QP-LR	**0.6863**	**0.3411**	**0.7201**	0.9349
MLP-QP-LR (w/ PCR)	**0.6890**	**0.3900**	**0.7221**	**0.9384**
MLP-TP-LR	**0.6980**	**0.3339**	**0.7260**	0.9355
MLP-TP-LR (w/PCR)	**0.6993**	0.3356	**0.7246**	**0.9384**

Table 5

Results of models applying different sharing strategies to various embeddings for numerical features. "Corr-grouped" refers to correlation-based feature grouping, "order-grouped" denotes order-based feature grouping, and "bin-grouped" denotes bin-count-based feature grouping. Comparisons are made within each group, with better results shown in bold. Notations follow Table 3.

Model	EL↑	HO↓	HI↑	MI↑
MLP-V-LR	**0.7107±9.347e-03**	**0.3278±2.966e-03**	**0.7240±1.309e-03**	**0.9393±2.390e-03**
MLP-V-LR (corr-grouped)	**0.7051±4.289e-03**	**0.3268±3.800e-03**	**0.7235±1.689e-03**	**0.9393±2.304e-03**
MLP-V-LR (shared)	**0.7053±6.294e-03**	**0.3285±2.318e-03**	0.7208±9.911e-04	**0.9398±1.801e-03**
Transformer-V-LR	**0.6970±1.402e-02**	**0.3371±2.542e-0**3	**0.7272±2.056e-03**	**0.9380±1.425e-03**
Transformer-V-LR (corr-grouped)	0.6732±8.170e-03	0.3716±1.483e-03	0.6579±2.495e-03	0.9303±1.712e-03
Transformer-V-LR (shared)	0.5000±0.000e+00	0.5131±8.941e-04	0.5443±2.823e-03	0.6403±4.262e-03
MLP-QT-L^0	0.6987±4.655e-03	0.3431±6.141e-03	**0.7203±8.848e-04**	**0.9379±4.293e-04**
MLP-QT-L^0 (corr-grouped)	0.6881±8.355e-03	**0.3390±2.383e-03**	**0.7197±1.780e-03**	**0.9377±6.130e-04**
MLP-QT-L^0 (shared)	**0.7069±8.160e-03**	0.3482±3.444e-03	0.7136±3.006e-03	**0.9376±5.734e-04**
MLP-QP-LR	0.6863±5.662e-03	0.3411±8.960e-04	0.7201±2.805e-03	**0.9349±1.983e-03**
MLP-QP-LR (corr-grouped)	0.6839±5.431e-03	**0.3393±2.374e-03**	**0.7230±2.732e-03**	0.9373±1.822e-03
MLP-QP-LR (shared)	**0.6973±5.351e-03**	**0.3399±2.834e-03**	0.7209±5.658e-04	**0.8512±1.756e-01**
MLP-P-LR	0.7001±7.951e-03	0.3311±3.313e-03	**0.7285±1.452e-03**	**0.9397±6.867e-04**
MLP-P-LR (corr-grouped)	**0.7110±4.181e-03**	**0.3273±4.416e-03**	**0.7283±2.803e-03**	0.9411±2.544e-03
MLP-P-LR (shared)	**0.7097±5.287e-03**	**0.3272±2.714e-03**	0.7254±1.614e-03	**0.8965±1.322e-01**
MLP-TT-L^0 (corr-grouped)	**0.6901±6.110e-03**	**0.3387±2.126e-03**	**0.7227±5.539e-04**	**0.9377±6.369e-04**
MLP-TT-L^0 (order-grouped)	0.6780±5.609e-03	0.3418±3.132e-03	0.7207±1.120e-03	**0.9373±6.867e-04**
MLP-TT-L^0 (bin-grouped)	0.6835±4.822e-03	0.3440±2.313e-03	0.7216±1.815e-03	0.9356±7.872e-04

further conduct feature reconstruction experiments on the trained embeddings. Specifically, we froze the embeddings and added a shallow decoder consisting of two linear layers, then trained the models to reconstruct the original feature values. The results show that the performance of embeddings trained with PCR was significantly worse than those trained without it (see Appendix B). This indicates that the regularization narrows the ability of the embeddings to capture the characteristics of the original data, due to their insufficient expressiveness caused by constraining the angles between embedding vectors. Overall, PCR helps PLE embeddings maintain stricter ordering, providing the gains observed in Figure 4; however, this comes at the cost of reduced expressiveness. For well-configured PLE, which can already achieve good ordering, PCR is often unnecessary. On the other hand, for more lightweight PLE, which are more prone to failure in maintaining ordering, it is crucial to weigh the tradeoff between the benefits and the loss that PCR introduces.

4.4. Sharing Strategies of Embeddings for Numerical Features

We compare the performance of models employing different sharing strategies across various embedding techniques. The grouping method based on feature correlations (described in Section 3.2.3) is used, with the number of groups set to half of the total number of features. The experimental results are presented in Table 5, and the key conclusions are summarized as follows: (1) For the MLP backbone, there are numerous opportunities to achieve more lightweight embedding modules by partially or fully sharing feature embeddings, without sacrificing performance—and in many cases, even improving it—when using different embedding techniques across various datasets. (2) For the Transformer backbone, shared feature embeddings can severely degrade model performance, or even lead to complete failure. Since the self-attention mechanism relies on differences between input vectors to compute attention weights, overly similar inputs can result in uniform attention distributions, making it difficult to differentiate between features. Moreover, we conduct an ablation study by designing two simple alternative grouping strategies based on the original feature order and the number of bins. Concretely, the order-based grouping divides features evenly according to their indices. The bin-count-based grouping applies only to embeddings using target-aware binning, as these techniques sometimes produce different number of bins for different features. This method groups features with similar bin counts together, effectively reducing the total number of embeddings. We evaluate the effects of applying the different grouping algorithms. As shown in Table 5, the correlation-based feature grouping method demonstrates clear superiority. Furthermore, we can apply different grouping ratios to this method to fine-tune the sharing level, spanning from full to no sharing. In practical applications, the grouping ratio can be treated as a hyperparameter, and hyperparameter optimization can be used to identify the optimal sharing strategy configuration.

4.5. Robustness of Embeddings for Numerical Features

We introduce various types of noise into the test sets of the datasets, at different proportions, to assess the performance of embedding methods for numerical features under high-interference conditions. More concretely, we experiment with three types of noise—Gaussian noise, swap noise, and zero-out noise [41]—with each type applied in proportions ranging from 0 to 0.5 at intervals of 0.1. Given the noise ratio p, we generate a binomial mask $\mathrm{m}\sim\mathrm{Bernoulli}(p)$ with the same shape as sample $\mathbf{x}$, indicating the positions of the noisy features. Then the corrupted version is computed as $\tilde{\mathrm{x}}=(1-\mathrm{m})\odot\mathrm{x}+\mathrm{m}\odot\mathrm{noise}$. With respect to noise types, swap noise shuffles the original data along the columns, while zero-out noise substitutes the original values with zeros. Gaussian noise is implemented as $\mathrm{noise}\sim\mathrm{N}\left(0,\mathrm{diag}\left(\sigma_1^2,\sigma_2^2,\cdots,\sigma_\ell^2\right)\right)$, where σ_j represents the standard deviation of the j-th numerical feature $(1\le\mathrm{j}\le\ell)$. Overall, all types of noise cause significant distortion to the affected data, requiring the embedding techniques to be as robust as possible to such substantial perturbations. Due to space limitations, a subset of the data supporting our conclusions is visualized in Figure 6. The full experimental data is available in Appendix B for readers seeking more comprehensive evidence.

We focus on the performance degradation of different embeddings, with the key findings as follows: Among all embedding methods, (1) periodic embed-

dings exhibit the weakest robustness against noise. While the periodic activation functions generally map inputs at lower frequencies, with c_i initialized from a Gaussian distribution with a mean of 0 (see Section 3.2.1), they tend to favor higher dimensions (the hyperparameter T), which increases their sensitivity to variations in feature values; (2) vanilla embeddings display better robustness than periodic embeddings alone, which is expected since they directly multiply feature values by the embedding vectors; (3) PLE, benefiting from the weighted summation of multiple bin embeddings, show the strongest resistance to noise; (4) token embeddings exhibit performance variability across datasets, showing the strongest resistance to noise on the EI dataset while ranking last on the remaining datasets. Additionally, we note that (1) the application of the Transformer backbone consistently leads to improvements over the MLP backbone when using vanilla embeddings; (2) swap noise is a relatively safe form of noise, which can introduce a certain level of challenge without causing catastrophic interference, distinguishing it from the Gaussian noise observed on the MI dataset and zero-out noise on the HO dataset (see Appendix B). This also provides empirical evidence supporting the superiority of column-shuffle-based data augmentation [48].

4.6. Evaluating Data Type and Positional Embeddings

We evaluate the effectiveness of several designs for meta feature embeddings. To be more precise, we use the additive and gating variant of the data type embeddings, and the various designs for positional embeddings (see Section 3.3.2). The main takeaways include that (1) for existing designs, it is generally not necessary to use extra embeddings for data types (see Table 6). Regarding the fusion mechanism of embeddings for data types, the gating mechanism outperforms simple addition, and in few cases, it can even result in significant performance improvements. (2) In addition to the commonly used bias terms, other positional embedding designs also demonstrate strong competitiveness. Among them, the bias terms ("pre-add" variant) take the lead, which is in line with Gorishniy et al. [10], while the "post-concat" variant lags behind (see Table 7). Overall, this indicates that adding individual positional embeddings for each feature is often an effective practice.

Figure 6

Performance of different models under various noise conditions across multiple datasets, with ribbons representing the standard deviation using different random seeds.

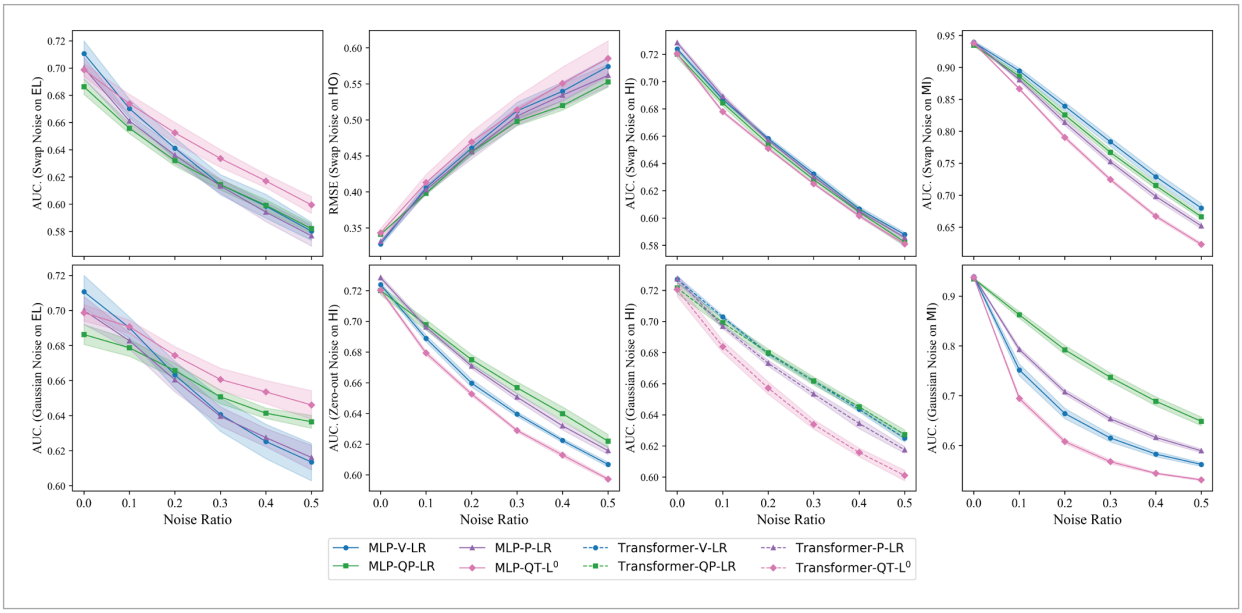


Table 6
Results of models applying different strategies for data type embeddings. $Embet_{dt}$ denotes data type embeddings. Comparisons are made within each group, with better results shown in bold. The standard deviations are provided in Appendix B. Notations follow Table 3.

Model	EL↑	CA↓	HO↓	AD↑	DI↓	HI↑	MI↑	BL↓
MLP-V-LR	**0.7107**	**0.4893**	**0.3278**	**0.7655**	**0.5567**	**0.7240**	**0.9393**	**0.3527**
MLP-V-LR (w/ additive $Embed_{dt}$)	0.6950	0.5124	0.3335	**0.7628**	**0.5533**	0.7202	**0.9395**	0.3566
MLP-V-LR (w/ gating $Embed_{dt}$)	**0.7098**	**0.4898**	**0.3267**	**0.7649**	**0.5560**	**0.7229**	**0.9395**	0.3540
Transformer-V-LR	**0.6970**	**0.4745**	**0.3371**	**0.7687**	0.5434	0.7272	**0.9380**	**0.3511**
Transformer-V-LR (w/ additive $Embed_{dt}$)	**0.6833**	0.5105	0.3512	**0.7647**	0.5517	0.7211	0.9341	0.3554
Transformer-V-LR (w/ gating $Embed_{dt}$)	**0.6974**	0.4806	0.3428	**0.7656**	**0.5410**	**0.7287**	**0.9383**	0.3563

Table 7
Results of models applying different strategies for positional embeddings. Top results are in bold. The standard deviations are provided in Appendix B. Notations follow Table 3.

Model	EL↑	CA↓	HO↓	AD↑	DI↓	HI↑	MI↑	BL↓	Avg. Rank
Transformer-V-L^0R	**0.6962**	0.4859	0.3364	0.7657	0.5477	**0.7284**	**0.9389**	0.3575	2.0±1.0
Transformer-V-L^0R (w/ pre-concat)	**0.7016**	0.4923	**0.3307**	0.7698	0.5461	**0.7285**	**0.9376**	0.3525	1.8±0.8
Transformer-V-L^0R (w/ pre-add)	**0.7034**	0.4790	0.3345	**0.7702**	0.5481	**0.7302**	**0.9399**	0.3510	1.5±0.5
Transformer-V-L^0R (w/ post-concat)	0.6965	0.4892	0.3531	**0.7721**	**0.5385**	0.7235	0.9373	**0.3496**	2.0±1.0
Transformer-V-L^0R (w/ post-add)	**0.7041**	**0.4727**	0.3469	**0.7732**	**0.5392**	0.7239	0.9367	**0.3494**	1.5±0.7

Table 8
Minimum Validation Loss with Different Maximum HPO Rounds on the HI dataset.

Models	80	160	240	320
MLP-V-LR	0.5292	0.5287	0.5283	0.5282
MLP-QT-L^0	0.5448	0.5406	0.5373	0.5365
MLP-TT-L^0	0.5410	0.5402	0.5366	0.5353
MLP-QP-LR	0.5324	0.5323	0.5323	0.5323
MLP-TP-LR	0.5288	0.5266	0.5263	0.5263
MLP-P-LR	0.5252	0.5230	0.5224	0.5223

4.7. HPO Rounds of Embeddings for Numerical Features

In our experiments, we conduct 320 rounds of HPO, since we found that the commonly used 100-round search in previous works [8-10] often fails to identify hyperparameter configurations that fully unlock the potential of models. We report the minimum loss values observed within the first 80, 160, 240 and 320 rounds of HPO for models employing different embedding methods for numerical features. Table 8 shows the results of the MLP backbones on the HI dataset, with complete data provided in Appendix B for reference. Our observations are as follows: (1) Even after 160 rounds, additional HPO rounds can still significantly improve the optimal scores for many algorithms. (2) Compared to other methods, the token embedding method is more heavily dependent on HPO. Since token embeddings often yield higher

loss values during training, failure to sufficiently reduce the loss through HPO can result in a model with catastrophic predictive performance. This makes token embeddings particularly costly in terms of training, which poses a significant limitation for practical use. In contrast, vanilla embeddings generally require fewer HPO rounds, and combined with their simpler embedding computation, they offer significant advantages in terms of training overhead.

5. Conclusion and Future Work

In this paper, we explored tabular embeddings for numerical features and meta-features. We identified two underexplored design directions of embeddings for numerical features: ordering-oriented regularization techniques and embedding sharing strategies. Based on these directions, we proposed two specific designs and demonstrated that appropriate embedding designs, guided by ordering and sharing principles, can indeed enhance both the effectiveness and efficiency of models. Additionally, we addressed several evaluation gaps in existing research. Specifically, we assessed the robustness of existing embeddings for numerical features, as well as the effectiveness of data type embeddings and positional embeddings in general tabular tasks.

This work points to several promising directions for future research. For instance, the proposed PCR is merely one sufficient condition for achieving piecewise linear embeddings; however, there are numerous feasible regularization constraints that can be explored to enhance ordering while providing embeddings with a higher degree of freedom. Moreover, we have only introduced a straightforward feature grouping strategy, which has notable limitations due to its heavy reliance on the reliability of feature correlation coefficients. Therefore, developing more stable and effective feature grouping methods, as well as metrics to indicate when embeddings sharing should occur, is worthwhile. Lastly, embedding techniques have limited capacity for handling high-dimensional features, especially in datasets with a large number of numerical features, where excessive embedding parameters can significantly burden the model. Thus, developing a well-designed embedding sharing strategy can be the key to alleviating this issue in the future.

Appendix A

Implementation Details

A.1. Datasets

We use eight datasets available at the OpenML repository [39], along with their default tasks. Specifically, the datasets with their OpenML dataset IDs are: EL (Elavators, ID: 845), CA (California Housing, ID: 43939), HO (House 16H, ID: 574), AD (Adult, ID: 45068), DI (Diamonds, ID: 42225), HI (Higgs, ID: 4532), MI (MiniBooNE, ID: 41150), BL (Black Friday, ID: 41540).

A.2. Computing Environment

All experiments were conducted using the PyTorch framework, version 2.1.2 [30]. We ran the hyperparameter optimization on any available hardware, and evaluated the tuned models on NVIDIA RTX 4090 GPU (24GB).

A.3. HPO Space

We mostly follow the hyperparameter tuning configurations recommended by the original papers. Subsequently, we detail the hyperparameter space for each component individually.

Optimizer. The space depends on the backbone.

1. For MLP or ResNet, the weight decay is either set to 0 or sampled from a log-uniform distribution over [1e-6, 1e-3]. The learning rate is sampled from a log-uniform distribution over [5e-5, 0.005].
2. For Transformer, the weight decay is sampled from a log-uniform distribution over [1e-6, 1e-4]. The learning rate is sampled from a log-uniform distribution over [1e-5, 1e-3].

Backbone.

1. For MLP models, the layer number was sampled uniformly from integer values between 1 and 16, while the layer size was selected from integers ranging from 1 to 1024. The dropout rate was either disabled (set to 0) or sampled uniformly from the interval [0, 0.5], and the embedding dimensionality was chosen from integers between 1 and 128.
2. For ResNet, the layer number varied between 1 and 8, with layer sizes sampled from integers in [32, 512]. Similar to MLP, the dropout rate could be disabled or sampled from [0, 0.5]. Additional

architectural parameters included a hidden factor (sampled from [1, 4]), hidden dimensionality ([0, 3]), hidden dropout rate ([0, 0.5]), and embedding dimensionality (integers in [1, 128]).

3 Transformer models employed between 1 and 4 layers, with distinct dropout rates: residual dropout was either disabled or sampled from [0, 0.2], attention dropout from [0, 0.5], and FFN dropout from [0, 0.5]. The FFN factor varied in [2/3, 8/3], while embedding dimensionality was selected from integer value between 96 and 512 in increments of 8 randomly.

Embedding. Embedding configurations included binning algorithms with the number of bins varying between 2 and 256. Target-aware binning incorporated two parameters: minimum samples per leaf node (integers in [1, 128]) and periodic encoding scale (log-uniform in [0.01, 100]).

Regularization. For RMT regularization, λ_{reg}. is sampled fron an uniform distribution over [0.01, 1]. For PCR, λ_{reg} is sampled fron an uniform distribution over [0.01, 100].

Appendix B

Extended Experiment Results

The full results are available at

https://github.com/OrangeMooon/TabEmbedExp/blob/main/full_results.xlsx.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M. Optuna: A Next-Generation Hyperparameter Optimization Framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019, 2623-2631. https://doi.org/10.1145/3292500.3330701
2. Borisov, V., Broelemann, K., Kasneci, E., Kasneci, G. DeepTLF: Robust Deep Neural Networks for Heterogeneous Tabular Data. International Journal of Data Science and Analytics, 2023, 16(1), 85-100. https://doi.org/10.1007/s41060-022-00350-z
3. Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., Kasneci, G. Deep Neural Networks and Tabular Data: A Survey. IEEE Transactions on Neural Networks and Learning Systems, 2022, 35(6), 7499-7519. https://doi.org/10.1109/TNNLS.2022.3229161
4. Chen, T., Guestrin, C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, 785-794. https://doi.org/10.1145/2939672.2939785
5. Conover, W. J. Practical Nonparametric Statistics. John Wiley & Sons, 1999.
6. Dastile, X., Celik, T., Potsane, M. Statistical and Machine Learning Models in Credit Scoring: A Systematic Literature Survey. Applied Soft Computing, 2020, 91, 106263. https://doi.org/10.1016/j.asoc.2020.106263
7. Dinh, T., Zeng, Y., Zhang, R., Lin, Z., Gira, M., Rajput, S., Sohn, J., Papailiopoulos, D., Lee, K. LIFT: Language-Interfaced Fine-Tuning for Non-Language Machine Learning Tasks. Advances in Neural Information Processing Systems, 2022, 35, 11763-11784.
8. Gorishniy, Y., Rubachev, I., Babenko, A. On Embeddings for Numerical Features in Tabular Deep Learning. Advances in Neural Information Processing Systems, 2022, 35, 24991-25004.
9. Gorishniy, Y., Rubachev, I., Kartashev, N., Shlenskii, D., Kotelnikov, A., Babenko, A. TabR: Tabular Deep Learning Meets Nearest Neighbors. International Conference on Learning Representations, 2024.
10. Gorishniy, Y., Rubachev, I., Khrulkov, V., Babenko, A. Revisiting Deep Learning Models for Tabular Data. Advances in Neural Information Processing Systems, 2021, 34, 18932-18943.
11. Grinsztajn, L., Oyallon, E., Varoquaux, G. Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data? Advances in Neural Information Processing Systems, 2022, 35, 507-520.
12. Guo, H., Tang, R., Ye, Y., Li, Z., He, X. DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction. Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017, 1725-1731. https://doi.org/10.24963/ijcai.2017/239
13. Hegselmann, S., Buendia, A., Lang, H., Agrawal, M., Jiang, X., Sontag, D. TabLLM: Few-Shot Classification

of Tabular Data with Large Language Models. International Conference on Artificial Intelligence and Statistics, 2023, 5549-5581.

14. Hollmann, N., Müller, S., Eggensperger, K., Hutter, F. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. International Conference on Learning Representations, 2023.
15. Huang, X., Khetan, A., Cvitkovic, M., Karnin, Z. TabTransformer: Tabular Data Modeling Using Contextual Embeddings. arXiv preprint, arXiv:2012.06678, 2020.
16. Kadra, A., Lindauer, M., Hutter, F., Grabocka, J. Well-Tuned Simple Nets Excel on Tabular Datasets. Advances in Neural Information Processing Systems, 2021, 34, 23928-23941.
17. Katzir, L., Elidan, G., El-Yaniv, R. Net-DNF: Effective Deep Modeling of Tabular Data. International Conference on Learning Representations, 2020.
18. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T. Y. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems, 2017, 30, 3146-3154.
19. Ke, G., Xu, Z., Zhang, J., Bian, J., Liu, T. Y. DeepGBM: A Deep Learning Framework Distilled by GBDT for Online Prediction Tasks. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019, 384-394. https://doi.org/10.1145/3292500.3330858
20. Kohavi, R., Sahami, M. Error-Based and Entropy-Based Discretization of Continuous Features. KDD, 1996, 114-119.
21. Kong, S., Cheng, W., Shen, Y., Huang, L. AutoSRH: An Embedding Dimensionality Search Framework for Tabular Data Prediction. IEEE Transactions on Knowledge and Data Engineering, 2022, 35(7), 6673-6686. https://doi.org/10.1109/TKDE.2022.3186387
22. Kossen, J., Band, N., Lyle, C., Gomez, A. N., Rainforth, T., Gal, Y. Self-Attention Between Datapoints: Going Beyond Individual Input-Output Pairs in Deep Learning. Advances in Neural Information Processing Systems, 2021, 34, 28742-28756.
23. Levin, R., Cherepanova, V., Schwarzschild, A., Bansal, A., Bruss, C. B., Goldstein, T., Wilson, A. G., Goldblum, M. Transfer Learning with Deep Tabular Models. The Eleventh International Conference on Learning Representations, 2023.
24. Liu, Q., Hu, J., Xiao, Y., Zhao, X., Gao, J., Wang, W., Li, Q., Tang, J. Multimodal Recommender Systems: A Survey. ACM Computing Surveys, 2024, 57(2), 1-17. https://doi.org/10.1145/3695461
25. Lloyd, S. Least Squares Quantization in PCM. IEEE Transactions on Information Theory, 1982, 28(2), 129-137. https://doi.org/10.1109/TIT.1982.1056489
26. Loshchilov, I., Hutter, F. Decoupled Weight Decay Regularization. International Conference on Learning Representations, 2019.
27. McElfresh, D., Khandagale, S., Valverde, J., Prasad C, V., Ramakrishnan, G., Goldblum, M., White, C. When Do Neural Nets Outperform Boosted Trees on Tabular Data? Advances in Neural Information Processing Systems, 2024, 36, 76336-76369.
28. Nam, J., Tack, J., Lee, K., Lee, H., Shin, J. STUNT: Few-Shot Tabular Learning with Self-Generated Tasks from Unlabeled Tables. International Conference on Learning Representations, 2023.
29. Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., Wermter, S. Continual Lifelong Learning with Neural Networks: A Review. Neural Networks, 2019, 113, 54-71. https://doi.org/10.1016/j.neunet.2019.01.012
30. Paszke, A., Gross, S., Massa, F., et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems, 2019, 32.
31. Popov, S., Morozov, S., Babenko, A. Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data. International Conference on Learning Representations, 2019.
32. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., Gulin, A. CatBoost: unbiased boosting with categorical features. Advances in Neural Information Processing Systems, 2018, 31.
33. Qian, J., Wang, H., Li, Z., Li, S., Yan, X. Limitations of Language Models in Arithmetic and Symbolic Induction. The 61st Annual Meeting of the Association for Computational Linguistics, 2023. https://doi.org/10.18653/v1/2023.acl-long.516
34. Qu, Y., Chen, T., Nguyen, Q. V. H., Yin, H. Budgeted Embedding Table for Recommender Systems. Proceedings of the 17th ACM International Conference on Web Search and Data Mining, 2024, 557-566. https://doi.org/10.1145/3616855.3635778
35. Salvi, M., Loh, H. W., Seoni, S., Barua, P. D., García, S., Molinari, F., Acharya, U. R. Multi-Modality Approaches for Medical Support Systems: A Systematic Review of the Last Decade. Information Fusion, 2024, 103, 102134. https://doi.org/10.1016/j.inffus.2023.102134

36. Shwartz-Ziv, R., Armon, A. Tabular Data: Deep Learning Is Not All You Need. Information Fusion, 2022, 81, 84-90. https://doi.org/10.1016/j.inffus.2021.11.011

37. Twala, B. Impact of Noise on Credit Risk Prediction: Does Data Quality Really Matter? Intelligent Data Analysis, 2013, 17(6), 1115-1134. https://doi.org/10.3233/IDA-130623

38. Van der Maaten, L., Hinton, G. Visualizing Data Using t-SNE. Journal of Machine Learning Research, 2008, 9(11).

39. Vanschoren, J., Van Rijn, J. N., Bischl, B., Torgo, L. OpenML: Networked Science in Machine Learning. ACM SIGKDD Explorations Newsletter, 2014, 15(2), 49-60. https://doi.org/10.1145/2641190.2641198

40. Wang, L., Zhang, X., Su, H., Zhu, J. A Comprehensive Survey of Continual Learning: Theory, Method and Application. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2024, 46(8), 5362-5383. https://doi.org/10.1109/TPAMI.2024.3367329

41. Wang, Z., Sun, J. TransTab: Learning Transferable Tabular Transformers Across Tables. Advances in Neural Information Processing Systems, 2022, 35, 2902-2915.

42. Wu, J., Chen, S., Zhao, Q., Sergazinov, R., Li, C., Liu, S., Zhao, C., Xie, T., Guo, H., Ji, C., Cociorva, D., Brunzell, H. SwitchTab: Switched Autoencoders Are Effective Tabular Learners. Proceedings of the AAAI Conference on Artificial Intelligence, 2024, 38(14), 15924-15933. https://doi.org/10.1609/aaai.v38i14.29523

43. Yan, J., Zheng, B., Xu, H., Zhu, Y., Chen, D., Sun, J., Wu, J., Chen, J. Making Pre-Trained Language Models Great on Tabular Prediction. International Conference on Learning Representations, 2024.

44. Yang, J., Triendl, H., Soltan, A. A., Prakash, M., Clifton, D. A. Addressing Label Noise for Electronic Health Records: Insights from Computer Vision for Tabular Data. BMC Medical Informatics and Decision Making, 2024, 24(1), 183. https://doi.org/10.1186/s12911-024-02581-5

45. Yang, Y., Wang, Y., Liu, G., Wu, L., Liu, Q. UniTabE: A Universal Pretraining Protocol for Tabular Foundation Model in Data Science. International Conference on Learning Representations, 2024.

46. Ye, C., Lu, G., Wang, H., Li, L., Wu, S., Chen, G., Zhao, J. Towards Cross-Table Masked Pretraining for Web Data Mining. Proceedings of the ACM on Web Conference 2024, 2024, 4449-4459. https://doi.org/10.1145/3589334.3645707

47. Yèche, H., Dresdner, G., Locatello, F., Hüser, M., Rätsch, G. Neighborhood Contrastive Learning Applied to Online Patient Monitoring. International Conference on Machine Learning, 2021, 11964-11974.

48. Yoon, J., Zhang, Y., Jordon, J., Van der Schaar, M. VIME: Extending the Success of Self- and Semi-Supervised Learning to Tabular Domain. Advances in Neural Information Processing Systems, 2020, 33, 11033-11043.

49. Zhu, B., Shi, X., Erickson, N., Li, M., Karypis, G., Shoaran, M. XTab: Cross-Table Pretraining for Tabular Transformers. Proceedings of Machine Learning Research, 2023, 202, 43181-43204.

50. Zhu, X., Wu, X. Class Noise vs. Attribute Noise: A Quantitative Study. Artificial Intelligence Review, 2004, 22, 177-210. https://doi.org/10.1007/s10462-004-0751-8