On Embeddings for Numerical Features in Tabular Deep Learning

Yury Gorishniy Ivan Rubachev Artem Babenko

Abstract

Recently, Transformer-like deep architectures have shown strong performance on tabular data problems. Unlike traditional models, e.g., MLP, these architectures map scalar values of numerical features to high-dimensional embeddings before mixing them in the main backbone. In this work, we argue that embeddings for numerical features are an underexplored degree of freedom in tabular DL, which allows constructing more powerful DL models and competing with GBDT on some traditionally GBDT-friendly benchmarks. We start by describing two conceptually different approaches to building embedding modules: the first one is based on a piecewise linear encoding of scalar values, and the second one utilizes periodic activations. Then, we empirically demonstrate that these two approaches can lead to significant performance boosts compared to the embeddings based on conventional blocks such as linear layers and ReLU activations. Importantly, we also show that embedding numerical features is beneficial for many backbones, not only for Transformers. Specifically, after proper embeddings, simple MLP-like models can perform on par with the attention-based architectures. Overall, we highlight embeddings for numerical features as an important design aspect with good potential for further improvements in tabular DL.

1. Introduction

Tabular data problems are currently a final frontier for deep learning research. While the most recent breakthroughs in NLP, vision, and speech are achieved by deep models (Goodfellow et al., 2016), their success in the tabular domain is not convincing yet. Despite a large number of proposed architectures for tabular DL (Klambauer et al., 2017; Popov et al., 2020; Arik & Pfister, 2020; Song et al., 2019; Badirli et al., 2020; Huang et al., 2020; Somepalli et al., 2021; Gorishniy et al., 2021; Kossen et al., 2021), the performance gap between them and the "shallow" ensembles of decision trees, like GBDT, often remains significant (Gorishniy et al., 2021; Kadra et al., 2021).

The recent line of works (Gorishniy et al., 2021; Somepalli et al., 2021; Kossen et al., 2021) reduce this performance gap by successfully adapting the Transformer architecture (Vaswani et al., 2017) for the tabular domain. Compared to traditional models, like MLP or ResNet, the proposed Transformer-like architectures have a specific way to handle numerical features of the data. Namely, they map scalar values of numerical features to high-dimensional embedding vectors, which are then mixed by the self-attention modules. Beyond transformers, mapping numerical features to vectors was also employed in different forms in the CTR prediction problems (Covington et al., 2016; Song et al., 2019; Guo et al., 2021). Nevertheless, the literature is mostly focused on developing more powerful backbones while keeping the design of embedding modules relatively simple. In particular, the existing architectures (Gorishniy et al., 2021; Somepalli et al., 2021; Kossen et al., 2021; Song et al., 2019; Guo et al., 2021) construct embeddings for numerical features using quite restrictive parametric mappings, e.g., linear functions, which can lead to suboptimal performance. In this work, we demonstrate that the embedding step has a substantial impact on the model effectiveness, and its proper design can be a game-changer in tabular DL.

Specifically, we describe two different building blocks suitable for constructing embeddings for numerical features. The first one is a piecewise linear encoding that produces alternative initial representations for the original scalar values and is based on feature binning, a long-existing preprocessing technique (Dougherty et al., 1995). The second one relies on periodic activation functions, which is inspired by their usage in implicit neural representations (Mildenhall et al., 2020; Tancik et al., 2020; Sitzmann et al., 2020) and positional encodings in NLP and CV tasks (Vaswani et al., 2017; Li et al., 2021). We observe that DL models equipped with our embedding schemes successfully compete with GBDT on the established GBTD-friendly benchmarks and achieve the new state-of-the-art of tabular DL.

As another important finding, we demonstrate that the step of embedding the numerical features is universally beneficial for different deep architectures, not only for

Correspondence to: yuragorishniy@icloud.com
Code: https://github.com/Yura52/tabular-dl-num-embeddings

Transformer-like ones. In particular, we show, that after proper embeddings, simple MLP-like architectures often provide the performance comparable to the state-of-the-art attention-based models. Overall, our work demonstrates the large impact of the embeddings of numerical features on the tabular DL performance and shows the potential of investigating more advanced embedding schemes in future research.

To sum up, our contributions are as follows:

1. We show that embedding schemes for numerical features are an underexplored research question in tabular DL. Namely, we show that more expressive embedding schemes can provide substantial performance improvements over prior models.
2. We demonstrate that the profit from embedding numerical features is not specific for Transformer-like architectures, and proper embedding schemes benefit traditional models as well.
3. On a number of public benchmarks, we achieve the new state-of-the-art of tabular DL.

2. Related Work

Tabular deep learning. During several recent years, the community has proposed a large number of deep models for tabular data (Klambauer et al., 2017; Popov et al., 2020; Arik & Pfister, 2020; Song et al., 2019; Wang et al., 2017; Badirli et al., 2020; Hazimeh et al., 2020; Huang et al., 2020; Somepalli et al., 2021; Gorishniy et al., 2021; Kossen et al., 2021). However, when systematically evaluated, these models do not consistently outperform the ensembles of decision trees, such as GBDT (Gradient Boosting Decision Tree) (Chen & Guestrin, 2016; Prokhorenkova et al., 2018; Ke et al., 2017), which are typically the top-choice in various ML competitions (Gorishniy et al., 2021; Shwartz-Ziv & Armon, 2021). Moreover, several recent works have shown that the proposed sophisticated architectures are not superior to properly tuned simple models, like MLP and ResNet (Gorishniy et al., 2021; Kadra et al., 2021). In this work, unlike the prior literature, we do not aim to propose a new backbone architecture. Instead, we focus on more accurate ways to handle numerical features, and our developments can be potentially combined with any model, including traditional MLPs and more recent Transformer-like ones.

Transformers in tabular DL. Due to the tremendous success of Transformers for different domains (Vaswani et al., 2017; Dosovitskiy et al., 2021), several recent works adapt their self-attention design for tabular DL as well (Huang et al., 2020; Gorishniy et al., 2021; Somepalli et al., 2021; Kossen et al., 2021). Compared to existing alternatives, applying self-attention modules to the numerical features of tabular data requires mapping the scalar values of these features to high-dimensional embedding vectors. So far, the existing architectures perform this "scalar" $\rightarrow$ "vector" mapping by relatively simple computational blocks, which, in practice, can limit the model expressiveness. For instance, the recent FT-Transformer architecture (Gorishniy et al., 2021) employs only a single linear layer. In our experiments, we demonstrate that such embedding schemes can provide suboptimal performance, and more advanced schemes often lead to substantial profit.

CTR Prediction. In CTR prediction problems, objects are represented by numerical and categorical features, which makes this field highly relevant to tabular data problems. In several works, numerical features are handled in some non-trivial way while not being the central part of the research (Covington et al., 2016; Song et al., 2019). Recently, however, a more advanced scheme has been proposed in Guo et al. (2021). Nevertheless, it is still based on linear layers and conventional activation functions, which we found to be suboptimal in our evaluation.

Feature binning. Binning is a discretization technique that converts numerical features to categorical features. Namely, for a given feature, its value range is split into bins (intervals), after which the original feature values are replaced with discrete descriptors (e.g. bin indices or one-hot vectors) of the bins corresponding to the initial scalar values. We point to the work by Dougherty et al. (1995), which performs an overview of some classic approaches to binning and can serve as an entry point to the relevant literature on the topic. In our work, however, we utilize bins in a different way. Specifically, we use their edges to construct lossless piecewise linear representations of the original scalar values. It turns out that such representations can provide substantial benefit to deep models on several tabular problems.

Periodic activations. Recently, periodic activation functions have become a key component in processing coordinates-like inputs, which is required in many applications. Examples include NLP (Vaswani et al., 2017), vision (Li et al., 2021), implicit neural representations (Mildenhall et al., 2020; Tancik et al., 2020; Sitzmann et al., 2020). In our work, we show that periodic activations can be used to construct powerful embedding modules for numerical features in tabular data problems. Importantly, contrary to some of the aforementioned papers, where components of the multidimensional coordinates are mixed (e.g. with linear layers) before passing them to periodic functions (Sitzmann et al., 2020; Tancik et al., 2020), we find it crucial to embed each feature separately before mixing them in the main backbone.

3. Embeddings for Numerical Features

In this section, we describe the general framework for what we call "embeddings for numerical features", show their usage for MLP-like architectures, and describe the main building blocks used in the experimental comparison in section 4.

Notation. For a given supervised learning problem on tabular data, we denote the dataset as $\left\{\left(x^j,\, y^j\right)\right\}_{j=1}^{n}$ where $y^j \in \mathbb{Y}$ represents the object's label and $x^j = \left(x^{j(num)},\, x^{j(cat)}\right) \in \mathbb{X}$ represents the object's features (numerical and categorical). $x_i^{j(num)}$, in turn, denotes the i-th numerical feature of the j-th object. Depending on the context, the j index can be omitted. The dataset is split into three disjoint parts $\overline{1,n} = J_{train} \cup J_{val} \cup J_{test}$, where the "train" part is used for training, the "validation" part is used for early stopping and hyperparameter tuning, and the "test" part is used for the final evaluation.

3.1. General Framework

We formalize the notion of "embeddings for numerical features" in Equation 1:

$$z_i = f_i\left(x_i^{(num)}\right) \in \mathbb{R}^{d_i} \tag{1}$$

where $f_i(x)$ is the embedding function for the i-th numerical feature, z_i is the embedding of the i-th numerical feature and d_i is the dimensionality of the embedding. Importantly, Equation 1 implies that embeddings for all features are computed *independently* of each other. Note that the function f_i can depend on parameters that are trained as a part of the whole model or in some other fashion (e.g. before the main optimization). In this work, we consider only embedding schemes where the embedding functions for all features are of the same functional form. We never share parameters of embedding functions of different features.

The subsequent use of the embeddings depends on the model backbone. For MLP-like architectures, they are concatenated into one flat vector and passed to the backbone as shown in Equation 2 (categorical features are omitted for simplicity):

$$\begin{aligned} &\texttt{MLP}(z_1,\, \ldots,\, z_k) = \texttt{MLP}\left(\texttt{concat}[z_1,\, \ldots,\, z_k]\right) \\ &\texttt{concat}[z_1,\, \ldots,\, z_k] \in \mathbb{R}^{d_1 + \ldots + d_k} \end{aligned} \tag{2}$$

From a practical standpoint, this approach can result in a noticeable increase in parameters count. We give more details on that in subsection 5.1.

For Transformer-based architectures, no extra step is performed and the embeddings are passed as is, so the usage is defined by the original architectures.

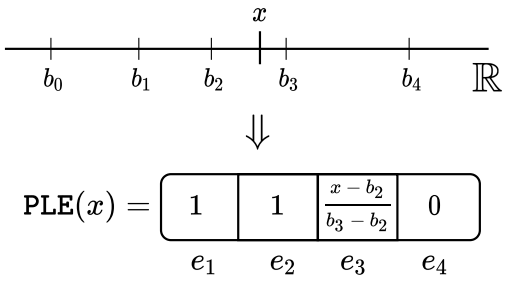

Figure 1. The piecewise linear encoding (PLE) in action, as defined in Equation 4. In the example, $T = 4$.

3.2. Piecewise Linear Encoding

While vanilla MLP is known to be a universal approximator (Cybenko, 1989; Hornik, 1991), in practice, due to optimization peculiarities, it has limitations in its learning capabilities (Rahaman et al., 2019). However, the recent work by Tancik et al. (2020) uncovers the case where changing the input space alleviates the above issue. This observation motivates us to check if changing the representations of the original scalar values of numerical features can improve the learning capabilities of tabular DL models.

At this point, we take inspiration from the one-hot encoding algorithm that is widely and successfully used for representing discrete entities such as categorical features in tabular data problems or tokens in NLP. We note that the one-hot representation can be seen as an opposite solution to the scalar representation in terms of the trade-off between parameter efficiency and expressivity. In order to check whether the one-hot-like approach can be beneficial for tabular DL models, we design a continuous alternative to the one-hot encoding (since the vanilla one-hot encoding is barely applicable to numerical features).

Formally, for the i-th numerical feature, we split its value range into the disjoint set of T^i intervals $B_1^i,\, \ldots,\, B_T^i$, which we call *bins*:

$$B_t^i = [b_{t-1}^i, b_t^i) \tag{3}$$

The splitting algorithm is an important implementation detail that we discuss later. From now on, we omit the feature index i for simplicity. Once the bins are determined, we define the encoding scheme as in Equation 4:

$$\begin{aligned} &\texttt{PLE}(x) = [e_1,\, \ldots,\, e_T] \in \mathbb{R}^T \\ &e_t = \begin{cases} 0, & x < b_{t-1} \texttt{ AND } t > 1 \\ 1, & x \geq b_t \texttt{ AND } t < T \\ \frac{x - b_{t-1}}{b_t - b_{t-1}}, & \text{otherwise} \end{cases} \end{aligned} \tag{4}$$

where PLE stands for "**p**eicewise **l**inear **e**ncoding".

We provide the visualization in Figure 1.
Note that:

- PLE produces alternative initial representations for the numerical features and can be viewed as a preprocessing strategy. These representations are computed once and then used instead of the original scalar values during the main optimization.
- For $T = 1$, the PLE-representation is effectively equivalent to the scalar representation.
- Contrary to categorical features, numerical features are ordered; we express that by setting to 1 the components corresponding to bins with the right boundaries lower than the given feature value.
- The cases $(x < b_0)$ and $(x \geq b_T)$ are also covered by Equation 4 (which leads to $(e_0 \leq 0)$ and $(e_T \geq 1)$ respectively).
- The choice to make the representation piecewise linear is itself a subject for discussion. We analyze some alternatives in subsection 5.3.

A note on attention-based models. While the described PLE-representations can be passed to MLP-like models as-is as described in Equation 2, attention-based models are inherently invariant to the order of input embeddings, so one additional step is required to add the information about feature indices to the obtained encodings. Technically, we observe that it is enough to place one linear layer after PLE(without sharing weights between features). Conceptually, however, this solution has a clear semantic interpretation. Namely, it is equivalent to allocating one trainable embedding $v_t \in \mathbb{R}^d$ for each bin B_t and obtaining the final feature embedding by aggregating the embeddings of its bins with e_t as weights, plus bias v_0, as expressed in Equation 5.

$$f_i(x) = v_0 + \sum_{t=1}^{T} e_t \cdot v_t = \texttt{Linear}(\texttt{PLE}(x)) \qquad (5)$$

In the following two sections, we describe two simple algorithms for building bins suitable for PLE. Namely, we rely on the classic binning algorithms (Dougherty et al., 1995) and one of the two algorithms is unsupervised, while another one utilizes labels for constructing bins.

3.2.1. Obtaining Bins from Quantiles

A natural baseline way to construct the bins for PLE is by splitting value ranges according to the uniformly chosen empirical quantiles of the corresponding individual feature distributions. Formally, for the i-th feature:

$$b_t = \texttt{Q}_{\frac{t}{T}}\left(\{x_i^{j(num)}\}_{j \in J_{train}}\right) \qquad (6)$$

where Q is the empirical quantile function. Trivial bins of zero size are removed.

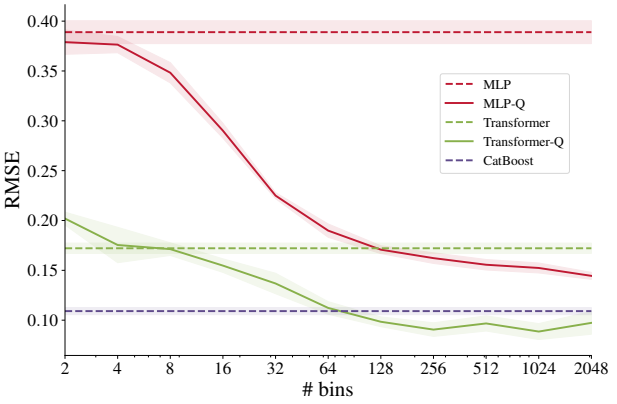

Figure 2. RMSE (averaged over five random seeds) of different approaches on the same synthetic GBDT-friendly task. Using PLE-representations ("-Q") instead of scalar values improves the performance of MLP and Transformer. Note that in practice, increasing the number of bins does not always lead to better results.

For a demonstration, we applied the proposed scheme to MLP and Transformer on the synthetic GBDT-friendly dataset described in section 5.1 in Gorishniy et al. (2021). In a nutshell, features of this dataset are sampled randomly from $\mathcal{N}(0, 1)$, and the target is produced by an ensemble of randomly constructed decision trees applied to the sampled features. This task turns out to be easy for GBDT, but hard for traditional DL models (Gorishniy et al., 2021). The technical details are provided in Appendix D. The results are visualized in Figure 2. As the plot shows, PLE-representations can be helpful for both MLP and Transformer backbones. In the considered synthetic setup, increasing the number of bins leads to better results, however, in practice, using too many bins can lead to overfitting; therefore, we recommend tuning the number of bins based on a validation set.

3.2.2. Building Target-aware Bins

In fact, there are also supervised approaches that employ training labels for constructing bins (Dougherty et al., 1995). Intuitively, such target-aware algorithms aim to produce bins that correspond to relatively narrow ranges of possible target values. The supervised approach used in our work is identical in its spirit to the "C4.5 Discretization" algorithm from Kohavi & Sahami (1996). In a nutshell, for each feature, we recursively split its value range in a greedy manner using target as guidance, which is equivalent to building a decision tree (which uses for growing only this one feature and the target) and treating the regions corresponding to its leaves as the bins for PLE. Additionally, we define the leftmost and rightmost bounds as follows:

$$b_0^i = \min_{j \in J_{train}} x_i^j \qquad b_T^i = \max_{j \in J_{train}} x_i^j \qquad (7)$$

Table 1. Dataset properties. Notation: "RMSE" denotes root-mean-square error, "Acc." denotes accuracy.

	GE	CH	EY	CA	HO	AD	OT	HI	FB	SA	CO	MI
#objects	9873	10000	10936	20640	22784	48842	61878	98049	197080	200000	581012	1200192
#num. features	32	10	26	8	16	6	93	28	50	200	54	136
#cat. features	0	1	0	0	0	8	0	0	1	0	0	0
metric	Acc.	Acc.	Acc.	RMSE	RMSE	Acc.	Acc.	Acc.	RMSE	Acc.	Acc.	RMSE
#classes	5	2	3	–	–	2	9	2	–	2	7	–

3.3. Periodic Activation Functions

Recall that in subsection 3.2 the work by Tancik et al. (2020) was used as a starting point of our motivation for developing `PLE`. Thus, we also try to adapt the original work itself for tabular data problems. Our variation differs in two aspects. First, we take into account the fact that Equation 1 forbids mixing features during the embedding process (see Appendix B for additional discussion). Second, we train the pre-activation coefficients instead of keeping them fixed. As a result, our approach is rather close to Li et al. (2021) with the number of "groups" equal to the number of numerical features. We formalize the described scheme in Equation 8,

$$f_i(x) = \texttt{Periodic}(x) = \texttt{concat}[\sin(v),\ \cos(v)],$$
$$v = [2\pi c_1 x,\ \ldots,\ 2\pi c_k x] \tag{8}$$

where c_i are trainable parameters initialized from $\mathcal{N}(0, \sigma)$. σ is an important hyperparameter that is tuned using validation sets.

3.4. Simple Differentiable Layers

In the context of Deep Learning, embedding numerical features with conventional differentiable layers (e.g. linear layers, ReLU activation, etc.) is a natural approach. In fact, this technique is already used on its own in the recently proposed attention-based architectures (Gorishniy et al., 2021; Kossen et al., 2021; Somepalli et al., 2021) and in some models for CTR prediction problems (Song et al., 2019; Guo et al., 2021). However, we also note that such conventional modules can be used on top of the components described in subsection 3.2 and subsection 3.3. In section 4, we find that such combinations often lead to better results.

4. Experiments

In this section, we empirically evaluate the techniques discussed in section 3 and compare them with Gradient Boosted Decision Trees to check the status quo of the "DL vs GBDT" competition.

4.1. Datasets

We use twelve public datasets mostly from the previous works on tabular DL and Kaggle competitions. Importantly, we focus on the middle and large scale tasks, and our benchmark is biased towards GBDT-friendly problems, since, as of now, closing the gap with GBDT models on such tasks is one of the main challenges for tabular DL.

The datasets include Gesture Phase Prediction (GE), Churn Modeling (CH), Eye Movements (EY), California Housing (CA), House 16H (HO), Adult (AD), Otto Group Product Classification (OT), Higgs (HI, we use the version with 98K samples available at the OpenML repository (Vanschoren et al., 2014)), Facebook Comments (FA), Santander Customer Transaction Prediction (SA), Covertype (CO), Microsoft (MI). The main dataset properties are summarized in Table 1 and the used sources and additional details are provided in Appendix A.

4.2. Implementation Details

We mostly follow Gorishniy et al. (2021) in terms of the tuning, training and evaluation protocols. Nevertheless, for completeness, we list all the details in this section.

Data preprocessing. Preliminary data preprocessing is known to be crucial for the optimization of tabular DL models. For each dataset, the same preprocessing was used for all deep models for a fair comparison. By default, we use the quantile transformation from the Scikit-learn library (Pedregosa et al., 2011). We apply standardization (mean subtraction and scaling) to Eye Movements and no preprocessing to Otto Group Product Classification since these preprocessing strategies lead to much better results for most DL models compared to the quantile-based one. We also apply standardization to regression targets for all algorithms.

Tuning. For every dataset, we carefully tune each model's hyperparameters. The best hyperparameters are the ones that perform best on the validation set, so the test set is never used for tuning. For most algorithms, we use the Optuna library (Akiba et al., 2019) to run Bayesian optimization (the Tree-Structured Parzen Estimator algorithm), which is reported to be superior to random search (Turner et al., 2021). The search spaces for all hyperparameters are reported in the appendix.

Evaluation. For each tuned configuration, we run 15 experiments with different random seeds and report the average performance on the test set.

Table 2. Results for MLP equipped with simple embedding modules (see subsection 4.3). The metric values averaged over 15 random seeds are reported. The standard deviations are provided in Appendix F. We consider one result to be better than another if its mean score is better and its standard deviation is less than the difference. For each dataset, top results are in **bold**. Notation: ↓ corresponds to RMSE, ↑ corresponds to accuracy

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
MLP	0.632	0.856	0.615	0.495	3.204	0.854	0.818	0.720	5.686	0.912	**0.964**	0.747
MLP-L	**0.639**	**0.861**	0.635	0.475	3.123	0.856	**0.820**	0.723	5.684	0.916	0.963	0.748
MLP-LR	**0.642**	**0.860**	**0.660**	**0.471**	**3.084**	0.857	**0.819**	**0.726**	**5.625**	0.923	0.963	**0.746**
MLP-AutoDis	**0.649**	0.857	0.634	0.474	3.165	**0.859**	0.807	**0.725**	**5.670**	**0.924**	**0.963**	–

Ensembles. For each model, on each dataset, we obtain three ensembles by splitting the 15 single models into three disjoint groups of equal size and averaging predictions of single models within each group.

Neural networks. The implementations of the MLP, ResNet, and Transformer backbones are taken from Gorishniy et al. (2021). We minimize cross-entropy for classification problems and mean squared error for regression problems. We use the AdamW optimizer (Loshchilov & Hutter, 2019). We do not apply learning rate schedules. For each dataset, we use a predefined batch size (see Appendix A for the specific values). We continue training until there are $\texttt{patience} + 1$ consecutive epochs without improvements on the validation set; we set $\texttt{patience} = 16$ for all models.

Categorical features. For CatBoost, we employ the built-in support for categorical features. For all other algorithms, we use the one-hot encoding.

Embeddings for numerical features. When differentiable components are presented, we tune the output dimensions of the corresponding linear layers. Hyperparameters that define the `PLE`-representations are tuned and they are the same for all features. For quantile-based bins, we tune the number of quantiles. For target-aware bins, we tune the following parameters for decision trees: the maximum number of leaves, the minimum number of items per leaf, and the minimum information gain required for making a split when growing the tree. For embeddings based on the `Periodic` module (subsection 3.3), we tune σ (it is the same for all features).

4.3. Model Names

In the experiments, we consider different combinations of backbones and embeddings. For convenience, we use the "Backbone-Embedding" pattern to name the models, where "Backbone" denotes the backbone (e.g. MLP, ResNet, Transformer) and "Embedding" denotes the embedding type. See Table 3 for all considered embedding modules. Note that "Transformer-L" is equivalent to FT-Transformer (Gorishniy et al., 2021).

Name	Embedding function (f_i)
`L`	`Linear`
`LR`	`ReLU` ∘ `Linear`
`LRLR`	`ReLU` ∘ `Linear` ∘ `ReLU` ∘ `Linear`
`Q`	$\texttt{PLE}_\texttt{q}$
`Q-L`	`Linear` ∘ $\texttt{PLE}_\texttt{q}$
`Q-LR`	`ReLU` ∘ `Linear` ∘ $\texttt{PLE}_\texttt{q}$
`Q-LRLR`	`ReLU` ∘ `Linear` ∘ `ReLU` ∘ `Linear` ∘ $\texttt{PLE}_\texttt{q}$
`T`	$\texttt{PLE}_\texttt{t}$
`T-L`	`Linear` ∘ $\texttt{PLE}_\texttt{t}$
`T-LR`	`ReLU` ∘ `Linear` ∘ $\texttt{PLE}_\texttt{t}$
`T-LRLR`	`ReLU` ∘ `Linear` ∘ `ReLU` ∘ `Linear` ∘ $\texttt{PLE}_\texttt{t}$
`P`	`Periodic`
`PL`	`Linear` ∘ `Periodic`
`PLR`	`ReLU` ∘ `Linear` ∘ `Periodic`
`PLRLR`	`ReLU` ∘ `Linear` ∘ `ReLU` ∘ `Linear` ∘ `Periodic`
`AutoDis`	`Linear` ∘ `SoftMax` ∘ $\texttt{Linear}_{-}$ ∘ `LReLU` ∘ $\texttt{Linear}_{-}$

Table 3. Embedding names. `Periodic` is defined in Equation 8. $\texttt{PLE}_\texttt{q}$ denotes `PLE` powered by quantiles. $\texttt{PLE}_\texttt{t}$ denotes the target-aware `PLE`. $\texttt{Linear}_{-}$ denotes linear layer without bias. `LReLU` denotes leaky ReLU. `AutoDis` was proposed in Guo et al. (2021)

4.4. Simple Differentiable Embedding Modules

We start by evaluating embedding modules consisting of "conventional" differentiable layers (linear layers, ReLU activations, etc.). The results are summarized in Table 2.
The main takeaways:

- first and foremost, the results indicate that MLP can benefit from embedding modules. Thus, we conclude that this backbone is worth attention when it comes to evaluating embedding modules.
- the simple `LR` module leads to modest, but consistent improvements when applied to MLP.

Interestingly, the "redundant" MLP-L configuration also tends to outperform the vanilla MLP. Although the improvements are not dramatic, the special property of this architecture is that the linear embedding module can be fused together with the first linear layer of MLP after training, which completely removes the overhead. As for `LRLR` (see the results in Appendix F) and AutoDis, we observe that these heavy modules do not justify the extra costs.

Table 4. Results for MLP and Transformer with embedding modules based on the piecewise linear encoding (subsection 3.2). Notation follows Table 2 and Table 3. The best results are defined separately for the MLP and Transformer backbones.

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
MLP	0.632	0.856	0.615	0.495	3.204	0.854	0.818	0.720	5.686	0.912	0.964	**0.747**
MLP-Q	**0.653**	0.854	0.604	0.464	**3.163**	0.859	0.816	0.721	5.766	0.922	0.968	0.750
MLP-T	**0.647**	**0.861**	0.682	0.447	**3.149**	0.864	**0.821**	0.720	5.577	0.923	0.967	0.749
MLP-Q-LR	**0.646**	0.857	**0.693**	0.455	**3.184**	0.863	0.811	0.720	**5.394**	0.923	**0.969**	**0.747**
MLP-T-LR	0.640	**0.861**	**0.685**	**0.439**	3.207	**0.868**	0.818	**0.724**	**5.508**	**0.924**	**0.968**	0.747
Transformer-L	0.632	0.860	0.731	0.465	3.239	0.858	**0.817**	0.725	**5.602**	0.924	0.971	**0.746**
Transformer-Q-L	**0.659**	0.856	0.753	0.451	3.319	0.867	0.812	**0.729**	5.741	**0.924**	**0.973**	0.747
Transformer-T-L	**0.663**	**0.861**	**0.775**	0.454	**3.197**	**0.871**	**0.817**	0.726	5.803	**0.924**	**0.974**	0.747
Transformer-Q-LR	**0.659**	0.857	**0.796**	0.448	3.270	0.867	0.812	0.723	5.683	0.923	0.972	0.748
Transformer-T-LR	**0.665**	0.860	**0.789**	**0.442**	**3.219**	0.870	**0.818**	**0.729**	5.699	**0.924**	0.973	0.747

Table 5. Results for MLP and Transformer with embedding modules based on periodic activations (subsection 3.3). Notation follows Table 2 and Table 3. The best results are defined separately for the MLP and Transformer backbones.

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
MLP	0.632	0.856	0.615	0.495	3.204	0.854	**0.818**	0.720	5.686	0.912	0.964	0.747
MLP-P	0.631	**0.860**	0.701	0.489	3.129	0.869	0.807	0.723	5.845	0.923	0.968	0.747
MLP-PL	0.641	**0.859**	0.866	**0.467**	3.113	0.868	**0.819**	0.727	**5.530**	0.924	**0.969**	0.746
MLP-PLR	**0.674**	**0.857**	**0.920**	**0.467**	**3.050**	**0.870**	**0.819**	**0.728**	**5.525**	**0.924**	**0.970**	**0.746**
Transformer-L	**0.632**	0.860	0.731	**0.465**	3.239	0.858	**0.817**	0.725	**5.602**	**0.924**	**0.971**	**0.746**
Transformer-PLR	**0.646**	**0.863**	**0.940**	**0.464**	**3.162**	**0.870**	0.814	**0.730**	5.760	**0.924**	**0.972**	**0.746**

4.5. Piecewise Linear Encoding

In this section, we evaluate the encoding scheme described in subsection 3.2. The results are summarized in Table 4.
The main takeaways:

- The piecewise linear encoding is often beneficial for both types of architectures (MLP and Transformer) and the profit can be significant (for example, see the California Housing and Adult datasets).
- Adding differentiable components on top of the `PLE` can improve the performance. Though, the most expensive modifications such as `Q-LRLR` and `T-LRLR` are not worth it (see Appendix F).

Note that the benchmark is biased towards GBDT-friendly problems, so the typical superiority of tree-based bins over quantile-based bins, which can be observed in Table 4, may not generalize to more DL-friendly datasets. Thus, we do not make any general claims about the relative advantages of the two schemes here. In practice, we recommend considering the type of embedding scheme as an additional hyperparameter that can be tuned on the validation set.

4.6. Periodic Activation Functions

In this section, we evaluate embedding modules based on periodic activation functions as described in subsection 3.3. The results are reported in Table 5.

The main takeaway: on average, MLP-P is superior to the vanilla MLP. However, adding a differentiable component on top of the `Periodic` module should be the default strategy (which is in line with Li et al. (2021)). Indeed, MLP-PLR and MLP-PL provide meaningful improvements over MLP-P on many datasets (e.g. GE, EY, CA, HO) and even "fix" MLP-P where it is inferior to MLP (OT, FB).

Although MLP-PLR is usually superior to MLP-PL, we note that in the latter case the last linear layer of the embedding module is "redundant" in terms of expressivity and can be fused with the first linear layer of the backbone after training, which, in theory, can lead to a more lightweight model. Finally, we observe that MLP-PLRLR and MLP-PLR do not differ significantly enough to justify the extra cost of the `PLRLR` module (see Appendix F).

4.7. Comparing DL Models and GBDT

In this section, we perform a big comparison of different approaches in order to identify the best embedding modules and backbones, as well as to check if embeddings for numerical features allow DL models to compete with GBDT on more tasks than before. Importantly, we compare *ensembles* of DL models against *ensembles* of GBDT, since Gradient Boosting is essentially an ensembling technique, so such comparison will be fairer. Note that we focus only on the best metric values without taking efficiency into account,

Table 6. Results for ensembles of GBDT, the baseline DL models and their modifications using different types of embeddings for numerical features. Notation follows Table 2 and Table 3. Due to the limited precision, some *different* values are represented with the same figures.

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓	Avg. Rank
CatBoost	0.692	0.861	0.757	0.430	3.093	0.873	0.825	0.727	5.226	0.924	0.967	**0.741**	6.8 ± 4.9
XGBoost	0.683	0.859	0.738	0.434	3.152	**0.875**	0.827	0.726	5.338	0.919	0.969	0.742	9.0 ± 5.7
MLP	0.665	0.856	0.637	0.486	3.109	0.856	0.822	0.727	5.616	0.913	0.968	0.746	15.6 ± 2.4
MLP-LR	0.679	0.861	0.694	0.463	3.012	0.859	0.826	0.731	5.477	0.924	0.972	0.744	10.2 ± 4.4
MLP-Q-LR	0.682	0.859	0.732	0.433	3.080	0.867	0.818	0.724	**5.144**	0.924	0.974	0.745	10.7 ± 4.6
MLP-T-LR	0.673	0.861	0.729	0.435	3.099	0.870	0.821	0.727	5.409	0.924	0.973	0.746	10.3 ± 3.8
MLP-PLR	**0.700**	0.858	0.968	0.453	**2.975**	0.874	**0.830**	**0.734**	5.388	**0.924**	0.975	0.743	4.9 ± 4.8
ResNet	0.690	0.861	0.667	0.483	3.081	0.856	0.821	0.734	5.482	0.918	0.968	0.745	12.1 ± 4.7
ResNet-LR	0.672	0.862	0.735	0.450	2.992	0.859	0.822	0.733	5.415	0.923	0.971	0.743	9.8 ± 4.3
ResNet-Q-LR	0.674	0.859	0.794	0.427	3.066	0.868	0.815	0.729	5.309	0.923	0.976	0.746	9.2 ± 4.8
ResNet-T-LR	0.683	0.862	0.817	**0.425**	3.030	0.872	0.822	0.731	5.471	0.923	0.975	0.744	7.8 ± 3.6
ResNet-PLR	0.691	0.861	0.925	0.443	3.040	**0.874**	0.825	0.734	5.400	0.924	0.975	0.743	5.2 ± 2.3
Transformer-L	0.668	0.861	0.769	0.455	3.188	0.860	0.824	0.727	5.434	0.924	0.973	0.743	10.6 ± 3.3
Transformer-LR	0.666	0.861	0.776	0.446	3.193	0.861	0.824	0.733	5.430	0.924	0.973	0.743	9.4 ± 4.1
Transformer-Q-LR	0.690	0.857	0.842	**0.425**	3.143	0.868	0.818	0.726	5.471	**0.924**	0.975	0.744	8.5 ± 5.5
Transformer-T-LR	0.686	0.862	0.833	**0.423**	3.149	0.871	0.823	0.733	5.515	0.924	**0.976**	0.744	7.2 ± 4.6
Transformer-PLR	0.686	**0.864**	**0.977**	0.449	3.091	0.873	0.823	0.734	5.581	**0.924**	0.975	0.743	6.0 ± 4.5

so we only check if DL models are conceptually ready to compete with GBDT. We consider three backbones: MLP, ResNet, and Transformer, since they are reported to be representative of what baseline DL backbones are currently capable of (Gorishniy et al., 2021). Note that we do not include the attention-based models that also apply attention on the level of *objects* (Somepalli et al., 2021; Kossen et al., 2021; Ramsauer et al., 2021), since this non-parametric component is orthogonal to the central topic of our work. The results are summarized in Table 6.

The main takeaways for DL models:

- For most datasets, embeddings for numerical features can provide noticeable improvements for three different backbones. Although the average rank is not a good metric for making subtle conclusions, we can't help noting the epic difference in average ranks between the MLP and MLP-PLR models.
- On the considered datasets, the `P-LR` module provides the best average performance. However, given the limited size of the benchmark, we cannot conclude that `P-LR` is the only solution that is worth attention. Thus, we also highlight the `PLE`-based embeddings, which outperform embeddings-free models in many cases and additionally are more hardware-friendly, since they do not rely on computationally expensive periodic functions.
- We highlight `LR` as a simple and stable embedding module for the MLP and ResNet backbones: although the performance gains are not dramatic, its main advantage is consistency, which makes MLP-LR and ResNet-LR attractive as simple DL solutions.
- Importantly, after the MLP-like architectures are coupled with embeddings for numerical features, they perform on par with the Transformer-based models.

The main takeaway for the "DL vs GBDT" competition: embeddings for numerical features is a significant design aspect that has a great potential for improving DL models and closing the gap with GBDT on GBDT-friendly tasks. Let us illustrate this claim with several observations:

- The benchmark is initially biased to GBDT-friendly problems, which can be observed by comparing GBDT solutions with the vanilla DL models (MLP, ResNet, Transformer-L).
- However, after coupling deep architectures with embeddings for numerical features, the situation changes and the MI dataset remains the only one where all DL models lag behind GBDT.
- Additionally, to the best of our knowledge, it is the first time when DL models perform on par with GBDT on the well-known California Housing and Adult datasets, the long-established GBDT-friendly benchmarks.

That said, compared to GBDT models, efficiency can still be an issue for the considered DL architectures. In any case, the trade-off completely depends on the specific use case and requirements.

5. Analysis

5.1. Comparing Model Sizes

While embeddings for numerical features are useful in terms of task performance, they can also lead to heavier models, especially for MLP-like models (see Equation 2). In order

Table 7. Parameter counts for MLP with different embedding modules. All the models are tuned and the corresponding backbones are not identical in their sizes, so we take into account the fact that different approaches require a different number of parameters to realize their full potential.

	GE	CH	EY	CA	HO	AD	OT	HI	FB	SA	CO	MI
MLP	2.0M	1.5K	1.7M	43.5K	3.6M	5.3M	479.9K	25.8K	937.3K	5.8M	3.2M	276.5K
MLP-LR	×2.52	×1931.03	×1.96	×25.05	×1.28	×0.35	×12.53	×68.16	×4.76	×1.58	×0.72	×15.79
MLP-T	×1.58	×14.13	×2.92	×7.97	×0.43	×0.04	×2.27	×5.85	×0.47	×0.59	×0.74	×3.85
MLP-T-LR	×1.61	×463.55	×0.85	×6.80	×0.23	×0.16	×2.52	×113.22	×3.43	×0.41	×0.35	×8.47
MLP-PLR	×1.73	×250.24	×0.57	×12.94	×1.07	×0.66	×8.05	×110.57	×4.93	×0.64	×0.44	×9.57

Table 8. Results for MLP and MLP with PLE for different types of data preprocessing. Solutions using PLE are significantly less sensitive to data preprocessing. Notation follows Table 2 and Table 3.

	GE ↑	CH ↑	CA ↓	HO ↓	AD ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
MLP (none)	0.565	0.796	1.118	5.328	0.808	0.707	13.125	0.911	0.948	0.844
MLP (standard)	0.629	0.855	0.509	3.303	0.855	0.721	5.919	0.912	0.963	0.754
MLP (quantile)	0.632	0.856	0.495	3.204	0.854	0.720	5.686	0.912	0.964	0.747
MLP-Q (none)	0.654	0.851	0.463	3.162	0.860	0.721	5.889	0.922	0.968	0.754
MLP-Q (quantile)	0.653	0.854	0.464	3.163	0.859	0.721	5.766	0.922	0.968	0.750
MLP-T (none)	0.644	0.860	0.447	3.175	0.865	0.721	5.598	0.923	0.968	0.749
MLP-T (quantile)	0.647	0.861	0.447	3.149	0.864	0.720	5.577	0.923	0.967	0.749

Table 9. Comparing piecewise linear encoding (PLE) with the two variations described in subsection 5.3. Notation follows Table 2 and Table 3.

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓
MLP-Q (piecewise linear)	**0.653**	**0.854**	0.604	0.464	**3.163**	**0.859**	**0.816**	**0.721**	5.766
MLP-Q (binary)	**0.652**	0.815	0.597	0.462	3.200	**0.860**	0.810	**0.720**	5.748
MLP-Q (one-blob)	0.613	0.851	**0.630**	**0.461**	**3.187**	0.857	0.808	0.719	**5.645**
MLP-T (piecewise linear)	**0.647**	**0.861**	**0.682**	**0.447**	**3.149**	0.864	**0.821**	0.720	5.577
MLP-T (binary)	0.639	0.855	0.633	0.464	**3.163**	**0.869**	0.813	0.718	5.572
MLP-T (one-blob)	0.622	0.858	0.660	0.464	**3.158**	**0.870**	0.809	**0.724**	**5.475**

to quantify this effect, we report the model sizes in Table 7. Overall, introducing embeddings for numerical features can cause non-negligible overhead in terms of parameter count. Importantly, the overhead in terms of size does not translate to the same overhead in terms of training times and throughput. For example, the almost 2000-fold increase in model size for MLP-LR on the CH dataset results in only 1.5-fold increase in training times. Finally, in practice, we observe that coupling MLP and ResNet with embedding modules lead to architectures that are still faster than Transformer-based models.

5.2. Is Piecewise Linear Encoding Sensitive to Preprocessing?

It is known that data preprocessing, such as standardization or quantile transformation, is often crucial for DL models for achieving competitive performance. Moreover, the performance can significantly vary between different types of preprocessing. At the same time, PLE-representations contain only values from $[0, 1]$ and they are invariant to shifting and scaling. These observations motivate us to check if models using PLE are sensitive to preprocessing to the same extent as vanilla models. For datasets where the quantile transformation was used in section 4, we reevaluate the tuned configurations of MLP, MLP-Q, and MLP-T with different preprocessing policies and report the results in Table 8 (note that standardization is equivalent to no preprocessing for models with PLE). First, the vanilla MLP often becomes unusable without preprocessing. Second, for the vanilla MLP, it can be important to choose one specific type of preprocessing (CA, HO, FB, MI), which is less pronounced for MLP-Q and not the case for MLP-T (though, this specific observation can be the property of the benchmarks, not of MLP-T). Overall, the results indicate that models using PLE are less sensitive to data preprocessing compared to the vanilla MLP. This is an additional benefit of PLE-representations for practitioners since the aspect of preprocessing becomes less critical with PLE.

5.3. Ablation Study

In this section, we compare two alternative binning-based encoding schemes with `PLE` (see subsection 3.2). The first considered option is to make the encoding "binary", that is, to assign value 1 instead of the ratio in the representation component corresponding to the bin to which the feature value belongs. The second alternative is a generalized version of the one-blob encoding (Müller et al., 2019) (see Appendix C for details). We follow the tuning and evaluation protocols described in subsection 4.2 and report results in Table 9. The table indicates that making the binning-based encoding piecewise linear is a good default strategy.

6. Conclusion & Future Work

In this work, we have demonstrated that embeddings for numerical features are an important design aspect of tabular DL architectures. Namely, it allows existing DL backbones to achieve noticeably better results and significantly reduce the gap with Gradient Boosted Decision Trees. We have described two approaches illustrating this phenomenon, one of which is based on the piecewise linear encoding of original scalar values, and another using periodic functions. We have also shown that traditional MLP-like models coupled with embeddings for numerical features can perform on par with attention-based models.

Nevertheless, we have only scratched the surface of the new direction. For example, it is still to be explained how exactly the discussed embedding modules help optimization on the fundamental level. Additionally, we have considered only schemes where the same functional transformation was applied to all features, which may be a suboptimal choice.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *KDD*, 2019.

Arik, S. O. and Pfister, T. Tabnet: Attentive interpretable tabular learning. *arXiv*, 1908.07442v5, 2020.

Badirli, S., Liu, X., Xing, Z., Bhowmik, A., Doan, K., and Keerthi, S. S. Gradient boosting neural networks: Grownet. *arXiv*, 2002.07971v2, 2020.

Baldi, P., Sadowski, P., and Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5, 2014.

Blackard, J. A. and Dean., D. J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, 2000.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *SIGKDD*, 2016.

Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. In *RecSys*, 2016.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.*, 2(4), 1989.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

Dougherty, J., Kohavi, R., and Sahami, M. Supervised and unsupervised discretization of continuous features. In *ICML*, 1995.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting deep learning models for tabular data. In *NeurIPS*, 2021.

Guo, H., Chen, B., Tang, R., Zhang, W., Li, Z., and He, X. An embedding learning framework for numerical features in CTR prediction. In *KDD*, 2021.

Hazimeh, H., Ponomareva, N., Mol, P., Tan, Z., and Mazumder, R. The tree ensemble layer: Differentiability meets conditional computation. In *ICML*, 2020.

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2), 1991.

Huang, X., Khetan, A., Cvitkovic, M., and Karnin, Z. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv*, 2012.06678v1, 2020.

Kadra, A., Lindauer, M., Hutter, F., and Grabocka, J. Well-tuned simple nets excel on tabular datasets. In *NeurIPS*, 2021.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

Kelley Pace, R. and Barry, R. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *NIPS*, 2017.

Kohavi, R. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *KDD*, 1996.

Kohavi, R. and Sahami, M. Error-based and entropy-based discretization of continuous features. In *KDD*, pp. 114–119. AAAI Press, 1996.

Kossen, J., Band, N., Lyle, C., Gomez, A. N., Rainforth, T., and Gal, Y. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. In *NeurIPS*, 2021.

Li, Y., Si, S., Li, G., Hsieh, C., and Bengio, S. Learnable fourier features for multi-dimensional spatial positional encoding. In *NeurIPS*, 2021.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *ICLR*, 2019.

Madeo, R. C. B., Lima, C. A. M., and Peres, S. M. Gesture unit segmentation using support vector machines: segmenting gestures from rest positions. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC*, 2013.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

Müller, T., McWilliams, B., Rousselle, F., Gross, M., and Novák, J. Neural importance sampling. *ACM Trans. Graph.*, 38(5), 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Popov, S., Morozov, S., and Babenko, A. Neural oblivious decision ensembles for deep learning on tabular data. In *ICLR*, 2020.

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

Qin, T. and Liu, T. Introducing LETOR 4.0 datasets. *arXiv*, 1306.2597v1, 2013.

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F. A., Bengio, Y., and Courville, A. C. On the spectral bias of neural networks. In *ICML*, 2019.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Adler, T., Kreil, D. P., Kopp, M. K., Klambauer, G., Brandstetter, J., and Hochreiter, S. Hopfield networks is all you need. In *ICLR*, 2021.

Salojarvi, J., Puolamaki, K., Simola, J., Kovanen, L., Kojo, I., and Kaski, S. Inferring relevance from eye movements: Feature extraction. *Helsinki University of Technology, Publications in Computer and Information Science*, 2005.

Shwartz-Ziv, R. and Armon, A. Tabular data: Deep learning is not all you need. *arXiv*, 2106.03253v1, 2021.

Singh, K., Sandhu, R. K., and Kumar, D. Comment volume prediction using neural networks and decision trees. In *IEEE UKSim-AMSS 17th International Conference on Computer Modelling and Simulation, UKSim*, 2015.

Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., and Wetzstein, G. Implicit neural representations with periodic activation functions. In *NeurIPS*, 2020.

Somepalli, G., Goldblum, M., Schwarzschild, A., Bruss, C. B., and Goldstein, T. SAINT: improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv*, 2106.01342v1, 2021.

Song, W., Shi, C., Xiao, Z., Duan, Z., Xu, Y., Zhang, M., and Tang, J. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *CIKM*, 2019.

Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., and Ng, R. Fourier features let networks learn high frequency functions in low dimensional domains. In *NeurIPS*, 2020.

Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., and Guyon, I. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. *arXiv*, https://arxiv.org/abs/2104.10201v1, 2021.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. Openml: networked science in machine learning. *arXiv*, 1407.7722v1, 2014.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, 2017.

Wang, R., Fu, B., Fu, G., and Wang, M. Deep & cross network for ad click predictions. In *ADKDD*, 2017.

A. Additional Details on Datasets

We used the following datasets:

- Gesture Phase Prediction (Madeo et al. (2013))
- Churn Modeling[1]
- Eye Movements (Salojarvi et al. (2005))
- California Housing (real estate data, Kelley Pace & Barry (1997))
- House 16H[2]
- Adult (income estimation, Kohavi (1996))
- Otto Group Product Classification[3]
- Higgs (simulated physical particles, Baldi et al. (2014); we use the version with 98K samples available at the
- OpenML repository (Vanschoren et al., 2014))
- Facebook Comments (Singh et al. (2015))
- Santander Customer Transaction Prediction[4]
- Covertype (forest characteristics, Blackard & Dean. (2000))
- Microsoft (search queries, Qin & Liu (2013)). We follow the pointwise approach to learning-to-rank and treat this ranking problem as a regression problem.

Table 10. Deteails on datasets, used for experiments

Abbr	Name	# Train	# Validation	# Test	# Num	# Cat	Task type	Batch size
GE	Gesture Phase	6318	1580	1975	32	0	Multiclass	128
CH	Churn Modelling	6400	1600	2000	10	1	Binclass	128
EY	Eye Movements	6998	1750	2188	26	0	Multiclass	128
CA	California Housing	13209	3303	4128	8	0	Regression	256
HO	House 16H	14581	3646	4557	16	0	Regression	256
AD	Adult	26048	6513	16281	6	8	Binclass	256
OT	Otto Group Products	39601	9901	12376	93	0	Multiclass	512
HI	Higgs Small	62751	15688	19610	28	0	Binclass	512
FB	Facebook Comments Volume	157638	19722	19720	50	1	Regression	512
SA	Santander Customer Transactions	128000	32000	40000	200	0	Binclass	1024
CO	Covertype	371847	92962	116203	54	0	Multiclass	1024
MI	MSLR-WEB10K (Fold 1)	723412	235259	241521	136	0	Regression	1024

B. Fourier Features

In this section, we test Fourier features implemented exactly as in Tancik et al. (2020), i.e. pre-activation coefficients are not trained and features are mixed right from the start. Importantly, the latter means that this approach is not covered by Equation 1. As reported in Table 11, MLP equipped with the original Fourier features does not perform well even compared to the vanilla MLP. So, it seems to be important to embed each feature separately as expressed in Equation 1.

Table 11. Results for the vanilla MLP and MLP equipped with Fourier features (Tancik et al., 2020). Notation follows Table 3 and Table 2.

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
MLP	**0.632**	**0.856**	**0.615**	**0.495**	**3.204**	0.854	**0.818**	**0.720**	**5.686**	0.912	**0.964**	**0.747**
MLP (Fourier features)	0.612	0.845	0.568	**0.495**	3.267	**0.858**	0.810	0.711	5.767	**0.915**	0.961	0.749

[1]https://www.kaggle.com/shrutimechlearn/churn-modelling
[2]https://www.openml.org/d/574
[3]https://www.kaggle.com/c/otto-group-product-classification-challenge/data
[4]https://www.kaggle.com/c/santander-customer-transaction-prediction

C. One-blob Encoding

In subsection 5.3, we used a slightly generalized version of the original one-blob encoding (Müller et al., 2019). Namely, while the original sets the width of the kernel to T^{-1} (T is the number of bins), we set it to $T^{-\gamma}$ and tune γ.

D. Experiments on the Synthetic Dataset

In this section, we provide technical details on the experiment that resulted in Figure 2.

Our dataset has $10,000$ objects, 8 features and the target was produced by 16 decision trees of depth 6. CatBoost is trained with the default hyperparameters. Transformer is trained with the default hyperparameters of FT-Transformer. The MLP backbone has four layers of size 256 each. Importantly, the task GBDT-friendly, which can be illustrated by the performance of the *tuned* MLP: 0.2229 ± 0.0055 (it is still worse than the performance of CatBoost). The remaining details can be found in the source code.

E. Hyperparameter Tuning Configurations

E.1. CatBoost

We fix and do not tune the following hyperparameters:

- `early-stopping-rounds` $= 50$
- `od-pval` $= 0.001$
- `iterations` $= 2000$

For tuning on the MI and CO datasets, we set the `task_type` parameter to "GPU". In all other cases (including the evaluation on these two datasets), we set this parameter to "CPU".

Table 12. CatBoost hyperparameter space

Parameter	Distribution
Max depth	$\mathrm{UniformInt}[1, 10]$
Learning rate	$\mathrm{LogUniform}[0.001, 1]$
Bagging temperature	$\mathrm{Uniform}[0, 1]$
L2 leaf reg	$\mathrm{LogUniform}[1, 10]$
Leaf estimation iterations	$\mathrm{UniformInt}[1, 10]$
# Iterations	100

E.2. XGBoost

We fix and do not tune the following hyperparameters:

- `booster` $=$ "gbtree"
- `early-stopping-rounds` $= 50$
- `n-estimators` $= 2000$

Table 13. XGBoost hyperparameter space.

Parameter	Distribution
Max depth	$\mathrm{UniformInt}[3, 10]$
Min child weight	$\mathrm{LogUniform}[0.0001, 100]$
Subsample	$\mathrm{Uniform}[0.5, 1]$
Learning rate	$\mathrm{LogUniform}[0.001, 1]$
Col sample by tree	$\mathrm{Uniform}[0.5, 1]$
Gamma	$\{0, \mathrm{LogUniform}[0.001, 100]\}$
Lambda	$\{0, \mathrm{LogUniform}[0.1, 10]\}$
# Iterations	100

E.3. MLP

Table 14. MLP hyperparameter space.

Parameter	Distribution
# Layers	$\mathrm{UniformInt}[1, 16]$
Layer size	$\mathrm{UniformInt}[1, 1024]$
Dropout	$\{0, \mathrm{Uniform}[0, 0.5]\}$
Learning rate	$\mathrm{LogUniform}[5e\text{-}5, 0.005]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}6, 1e\text{-}3]\}$
# Iterations	100

E.4. ResNet

Table 15. ResNet hyperparameter space.

Parameter	Distribution
# Layers	$\mathrm{UniformInt}[1, 8]$
Layer size	$\mathrm{UniformInt}[32, 512]$
Hidden factor	$\mathrm{Uniform}[1, 4]$
Hidden dropout	$\mathrm{Uniform}[0, 0.5]$
Residual dropout	$\{0, \mathrm{Uniform}[0, 0.5]\}$
Learning rate	$\mathrm{LogUniform}[5e\text{-}5, 0.005]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}6, 1e\text{-}3]\}$
# Iterations	100

E.5. Transformer

Table 16. Transformer hyperparameter space. Here (A) = {SA, CO, MI} and (B) = the rest

Parameter	(Datasets) Distribution
# Layers	(A) $\mathrm{UniformInt}[2, 4]$, (B) $\mathrm{UniformInt}[1, 4]$
Embedding size	(A) $\mathrm{UniformInt}[192, 512]$, (B) $\mathrm{UniformInt}[96, 512]$
Residual dropout	(A) $\mathrm{Const}(0.0)$, (B) $\{0, \mathrm{Uniform}[0, 0.2]\}$
Attention dropout	(A,B) $\mathrm{Uniform}[0, 0.5]$
FFN dropout	(A,B) $\mathrm{Uniform}[0, 0.5]$
FFN factor	(A,B) $\mathrm{Uniform}[2/3, 8/3]$
Learning rate	(A) $\mathrm{LogUniform}[1e\text{-}5, 3e\text{-}4]$, (B) $\mathrm{LogUniform}[1e\text{-}5, 1e\text{-}3]$
Weight decay	(A) $\mathrm{Const}(1e\text{-}5)$, (B) $\mathrm{LogUniform}[1e\text{-}6, 1e\text{-}4]$
# Iterations	(A) 50, (B) 100

E.6. Embedding Hyperparameters

The distribution for the output dimensions of linear layers is $\mathrm{UniformInt}[1, 128]$.

`PLE`. We share the same hyperparameter space for `PLE` across all datasets and models. For the quantile-based `PLE`, the distribution for the number of quantiles is $\mathrm{UniformInt}[2, 256]$. For the target-aware (tree-based) `PLE`, the distribution for the number of leaves is $\mathrm{UniformInt}[2, 256]$, the distribution for the minimum number of items per leaf is $\mathrm{UniformInt}[1, 128]$ and the distribution for the minimum information gain required for making a split is $\mathrm{LogUniform}[1e\text{-}9, 0.01]$.

`Periodic`. The distribution for k (see Equation 8) is $\mathrm{UniformInt}[1, 128]$.

F. Extended Tables With Experimental Results

The scores with standard deviations for single models and ensembles are provided in Table 17 and Table 18 respectively.

Table 17. Extended results for single models

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
CatBoost	0.683±4.7e-3	0.861±3.5e-3	0.748±7.2e-3	0.433±1.8e-3	3.115±1.9e-2	0.872±9.0e-4	0.824±1.1e-3	0.726±1.0e-3	5.324±4.1e-2	0.923±3.6e-4	0.966±3.3e-4	0.743±3.1e-4
XGBoost	0.678±4.9e-3	0.858±2.2e-3	0.730±8.5e-3	0.436±2.5e-3	3.160±6.9e-3	0.874±8.2e-4	0.825±2.3e-3	0.724±1.0e-3	5.383±2.9e-2	0.918±5.0e-3	0.969±6.1e-4	0.742±1.6e-4
MLP	0.632±1.4e-2	0.856±2.8e-3	0.615±6.9e-3	0.495±4.3e-3	3.204±4.0e-2	0.854±1.6e-3	0.818±3.1e-3	0.720±2.3e-3	5.686±4.7e-2	0.912±4.3e-4	0.964±8.6e-4	0.747±2.5e-4
MLP-L	0.639±1.3e-2	0.861±2.1e-3	0.635±1.1e-2	0.475±5.4e-3	3.123±4.5e-2	0.856±1.6e-3	0.820±1.5e-3	0.723±1.6e-3	5.684±4.5e-2	0.916±3.5e-4	0.963±9.3e-4	0.748±4.1e-4
MLP-LR	0.642±1.5e-2	0.860±3.0e-3	0.660±1.0e-2	0.471±2.6e-3	3.084±2.7e-2	0.857±1.9e-3	0.819±1.6e-3	0.726±1.9e-3	5.625±5.6e-2	0.923±3.1e-4	0.963±1.4e-3	0.746±3.9e-4
MLP-LRLR	0.654±1.7e-2	0.861±2.6e-3	0.697±6.5e-3	0.460±3.8e-3	3.070±3.6e-2	0.857±1.4e-3	0.819±2.2e-3	0.725±1.2e-3	5.551±4.6e-2	0.923±3.0e-4	0.963±1.4e-3	0.746±3.0e-4
MLP-Q	0.653±8.9e-3	0.854±3.0e-3	0.604±8.8e-3	0.464±3.1e-3	3.163±3.1e-2	0.859±1.6e-3	0.816±2.6e-3	0.721±1.0e-3	5.766±5.3e-2	0.922±6.3e-4	0.968±6.9e-4	0.750±3.7e-4
MLP-Q-LR	0.646±6.3e-3	0.857±2.6e-3	0.693±1.0e-2	0.455±3.4e-3	3.184±3.1e-2	0.863±1.7e-3	0.811±1.8e-3	0.720±1.5e-3	5.394±1.5e-1	0.923±6.1e-4	0.969±4.8e-4	0.747±3.9e-4
MLP-Q-LRLR	0.644±6.2e-3	0.859±2.2e-3	0.692±1.4e-2	0.452±4.3e-3	3.118±4.6e-2	0.869±1.5e-3	0.812±2.5e-3	0.724±1.3e-3	5.618±2.0e-1	0.924±4.5e-4	0.969±1.2e-3	0.748±4.6e-4
MLP-T	0.647±5.7e-3	0.861±1.1e-3	0.682±6.8e-3	0.447±2.0e-3	3.149±5.2e-2	0.864±6.3e-4	0.821±1.8e-3	0.720±1.9e-3	5.577±3.7e-2	0.923±3.0e-4	0.967±1.1e-3	0.749±4.4e-4
MLP-T-LR	0.640±6.9e-3	0.861±2.0e-3	0.685±8.4e-3	0.439±3.7e-3	3.207±5.2e-2	0.868±1.1e-3	0.818±1.3e-3	0.724±1.7e-3	5.508±3.0e-2	0.924±2.4e-4	0.968±7.2e-4	0.747±5.7e-4
MLP-T-LRLR	0.629±1.0e-2	0.857±2.4e-3	0.729±7.6e-3	0.446±3.6e-3	3.153±4.0e-2	0.870±9.9e-4	0.818±2.1e-3	0.725±1.3e-3	5.553±2.4e-2	0.924±3.6e-4	0.967±8.5e-4	0.748±5.8e-4
MLP-P	0.631±1.7e-2	0.860±3.1e-3	0.701±1.1e-2	0.489±2.4e-3	3.129±4.3e-2	0.869±1.5e-3	0.807±4.3e-3	0.723±1.5e-3	5.845±6.4e-2	0.923±4.3e-4	0.968±9.0e-4	0.747±3.1e-4
MLP-PL	0.641±1.0e-2	0.859±2.4e-3	0.866±3.1e-2	0.467±2.9e-3	3.113±3.1e-2	0.868±1.1e-3	0.819±1.7e-3	0.727±1.7e-3	5.530±9.5e-2	0.924±4.0e-4	0.969±5.0e-4	0.746±2.6e-4
MLP-PLR	0.674±1.0e-2	0.857±2.4e-3	0.920±3.1e-2	0.467±5.8e-3	3.050±3.4e-2	0.870±1.0e-3	0.819±2.0e-3	0.728±1.6e-3	5.525±3.5e-2	0.924±4.0e-4	0.970±9.5e-4	0.746±3.0e-4
MLP-PLRLR	0.676±1.6e-2	0.863±3.1e-3	0.990±3.9e-3	0.456±3.7e-3	3.038±2.3e-2	0.871±1.4e-3	0.818±1.7e-3	0.725±1.6e-3	5.606±8.9e-2	0.924±2.8e-4	0.968±2.0e-3	0.744±2.8e-4
MLP-AutoDis	0.649±1.2e-2	0.857±3.2e-3	0.634±1.1e-2	0.474±5.1e-3	3.165±1.8e-2	0.859±1.3e-3	0.807±2.4e-3	0.725±1.9e-3	5.670±6.1e-2	0.924±3.0e-4	0.963±8.7e-4	–
ResNet	0.655±2.0e-2	0.858±3.1e-3	0.639±1.1e-2	0.490±5.0e-3	3.153±3.6e-2	0.855±8.9e-4	0.817±3.4e-3	0.729±2.1e-3	5.681±5.3e-2	0.916±5.0e-4	0.965±8.3e-4	0.747±4.1e-4
ResNet-L	0.644±1.9e-2	0.859±1.8e-3	0.662±9.5e-3	0.490±6.6e-3	3.126±5.6e-2	0.855±1.4e-3	0.813±2.2e-3	0.730±9.7e-4	5.758±8.0e-2	0.915±4.3e-4	0.964±1.8e-3	0.747±4.7e-4
ResNet-LR	0.635±2.3e-2	0.861±2.2e-3	0.707±1.1e-2	0.465±3.5e-3	3.096±5.8e-2	0.856±1.6e-3	0.815±3.3e-3	0.729±1.3e-3	5.574±7.4e-2	0.922±4.4e-4	0.967±8.8e-4	0.746±4.4e-4
ResNet-Q	0.658±8.0e-3	0.858±2.4e-3	0.669±7.9e-3	0.454±3.6e-3	3.251±3.8e-2	0.860±1.3e-3	0.811±1.6e-3	0.718±1.0e-3	5.828±9.3e-2	0.921±9.1e-4	0.970±5.7e-4	0.749±2.9e-4
ResNet-Q-LR	0.650±9.2e-3	0.854±4.2e-3	0.733±4.7e-2	0.446±5.1e-3	3.217±5.2e-2	0.865±2.2e-3	0.808±2.6e-3	0.722±1.9e-3	5.514±6.0e-2	0.922±5.9e-4	0.972±3.7e-4	0.748±5.0e-4
ResNet-T	0.657±9.0e-3	0.859±2.9e-3	0.724±9.4e-3	0.441±3.2e-3	3.151±5.9e-2	0.866±1.8e-3	0.817±1.7e-3	0.724±2.0e-3	5.781±4.1e-2	0.923±6.0e-4	0.970±1.1e-3	0.749±7.8e-4
ResNet-T-LR	0.650±1.2e-2	0.861±2.0e-3	0.771±1.2e-2	0.438±2.9e-3	3.163±6.1e-2	0.870±1.5e-3	0.813±2.5e-3	0.725±1.6e-3	5.687±5.9e-2	0.922±8.1e-4	0.972±3.7e-4	0.748±6.1e-4
ResNet-P	0.630±1.8e-2	0.858±3.1e-3	0.725±1.7e-2	0.471±6.5e-3	3.147±2.9e-2	0.866±1.7e-3	0.812±1.6e-3	0.729±7.0e-4	5.566±7.5e-2	0.922±6.7e-4	0.968±7.7e-4	0.747±6.3e-4
ResNet-PLR	0.651±1.3e-2	0.859±3.7e-3	0.889±3.4e-2	0.461±4.2e-3	3.188±7.3e-2	0.869±1.7e-3	0.816±2.5e-3	0.728±1.8e-3	5.582±4.9e-2	0.923±5.9e-4	0.972±5.1e-4	0.747±6.4e-4
Transformer-L	0.632±2.0e-2	0.860±3.0e-3	0.731±8.3e-3	0.465±4.8e-3	3.239±3.2e-2	0.858±1.3e-3	0.817±2.3e-3	0.725±3.2e-3	5.602±4.8e-2	0.924±4.4e-4	0.971±6.8e-4	0.746±5.7e-4
Transformer-LR	0.614±4.5e-2	0.860±2.2e-3	0.739±1.0e-2	0.456±3.7e-3	3.261±5.6e-2	0.858±1.6e-3	0.817±2.2e-3	0.729±1.5e-3	5.644±5.5e-2	0.924±3.9e-4	0.971±7.6e-4	0.746±5.8e-4
Transformer-Q-L	0.659±8.7e-3	0.856±5.9e-3	0.753±1.7e-2	0.451±5.4e-3	3.319±4.2e-2	0.867±1.6e-3	0.812±2.6e-3	0.729±2.9e-3	5.741±4.5e-2	0.924±3.8e-4	0.973±6.1e-4	0.747±7.9e-4
Transformer-Q-LR	0.659±1.2e-2	0.857±2.0e-3	0.796±2.5e-2	0.448±6.1e-3	3.270±4.6e-2	0.867±1.1e-3	0.812±2.5e-3	0.723±3.3e-3	5.683±4.8e-2	0.923±5.8e-4	0.972±4.2e-4	0.748±7.7e-4
Transformer-T-L	0.663±7.4e-3	0.861±1.4e-3	0.775±1.2e-2	0.454±4.7e-3	3.197±2.9e-2	0.871±1.4e-3	0.817±2.6e-3	0.726±1.7e-3	5.803±6.5e-2	0.924±3.3e-4	0.974±4.5e-4	0.747±6.5e-4
Transformer-T-LR	0.665±6.6e-3	0.860±3.4e-3	0.789±1.5e-2	0.442±5.3e-3	3.219±3.2e-2	0.870±1.5e-3	0.818±2.6e-3	0.729±1.4e-3	5.699±6.7e-2	0.924±4.4e-4	0.973±5.6e-4	0.747±8.4e-4
Transformer-PLR	0.646±2.0e-2	0.863±2.7e-3	0.940±3.8e-2	0.464±2.8e-3	3.162±4.2e-2	0.870±1.5e-3	0.814±2.1e-3	0.730±1.9e-3	5.760±1.1e-1	0.924±5.2e-4	0.972±1.1e-3	0.746±5.9e-4

Table 18. Extended results for ensembles

	GE ↑	CH ↑	EY ↑	CA ↓	HO ↓	AD ↑	OT ↑	HI ↑	FB ↓	SA ↑	CO ↑	MI ↓
CatBoost	0.692±1.9e-3	0.861±2.4e-4	0.757±2.3e-3	0.430±1.1e-3	3.093±5.1e-3	0.873±5.1e-4	0.825±4.7e-4	0.727±3.6e-4	5.226±1.3e-2	0.924±1.0e-4	0.967±1.4e-4	0.741±1.4e-4
XGBoost	0.683±1.3e-3	0.859±2.4e-4	0.738±5.6e-3	0.434±7.1e-4	3.152±1.2e-3	0.875±5.5e-4	0.827±8.4e-4	0.726±8.1e-4	5.338±1.9e-2	0.919±4.8e-4	0.969±8.8e-5	0.742±5.3e-5
MLP	0.665±2.7e-3	0.856±1.2e-3	0.637±7.5e-3	0.486±7.8e-4	3.109±1.0e-2	0.856±4.6e-4	0.822±8.0e-4	0.727±1.7e-3	5.616±7.6e-3	0.913±8.2e-5	0.968±4.8e-4	0.746±1.1e-4
MLP-L	0.670±3.2e-3	0.862±1.5e-3	0.661±3.5e-3	0.471±4.7e-4	3.021±1.1e-2	0.857±5.9e-4	0.824±1.1e-3	0.728±2.7e-4	5.508±2.1e-2	0.916±1.5e-4	0.971±6.8e-5	0.746±2.3e-4
MLP-LR	0.679±4.9e-3	0.861±9.4e-4	0.694±6.5e-3	0.463±1.9e-3	3.012±1.8e-3	0.859±8.0e-4	0.826±1.6e-3	0.731±1.1e-3	5.477±3.6e-2	0.924±7.1e-5	0.972±7.6e-5	0.744±1.6e-4
MLP-LRLR	0.676±4.8e-3	0.863±1.4e-3	0.728±4.5e-3	0.453±1.1e-3	3.017±1.1e-2	0.858±1.6e-4	0.828±1.3e-3	0.725±5.9e-4	5.427±2.1e-2	0.924±1.2e-4	0.973±1.8e-4	0.744±1.8e-4
MLP-Q	0.677±4.8e-3	0.856±1.4e-3	0.640±1.2e-3	0.458±1.7e-4	3.080±1.5e-2	0.862±4.0e-4	0.822±1.7e-3	0.723±5.6e-4	5.706±1.9e-2	0.922±1.7e-4	0.973±2.1e-4	0.748±2.2e-4
MLP-Q-LR	0.682±3.9e-3	0.859±4.7e-4	0.732±3.9e-3	0.433±1.9e-3	3.080±9.7e-3	0.867±4.2e-4	0.818±1.4e-3	0.724±3.2e-4	5.144±1.4e-2	0.924±3.7e-4	0.974±1.5e-4	0.745±2.8e-4
MLP-Q-LRLR	0.674±2.9e-3	0.862±1.5e-3	0.739±3.0e-3	0.438±2.1e-3	3.066±9.9e-3	0.870±4.1e-4	0.817±2.4e-3	0.727±2.1e-4	5.268±7.5e-2	0.924±3.1e-5	0.973±2.8e-4	0.745±1.7e-4
MLP-T	0.669±4.3e-3	0.861±1.0e-3	0.703±4.9e-3	0.439±2.1e-4	3.058±1.4e-2	0.865±5.3e-4	0.822±6.3e-4	0.724±7.2e-4	5.507±2.0e-2	0.923±8.5e-5	0.972±2.7e-4	0.747±4.1e-5
MLP-T-LR	0.673±8.3e-4	0.861±8.5e-4	0.729±4.9e-3	0.435±1.1e-3	3.099±2.4e-2	0.870±6.6e-4	0.821±2.6e-4	0.727±7.2e-4	5.409±6.2e-3	0.924±1.3e-4	0.973±1.3e-4	0.746±1.6e-4
MLP-T-LRLR	0.670±4.1e-4	0.860±2.5e-3	0.752±3.5e-3	0.431±6.0e-4	3.056±2.2e-2	0.870±2.6e-4	0.826±5.0e-4	0.725±7.4e-4	5.440±1.8e-3	0.925±6.1e-5	0.973±2.2e-4	0.745±4.7e-4
MLP-P	0.661±6.0e-3	0.861±6.2e-4	0.758±3.5e-3	0.473±1.1e-3	3.042±1.0e-2	0.871±1.1e-3	0.812±1.7e-3	0.725±6.2e-4	5.508±3.1e-2	0.924±5.4e-5	0.973±3.0e-4	0.745±2.1e-4
MLP-PL	0.671±6.2e-3	0.860±1.2e-3	0.932±4.4e-3	0.456±1.3e-3	3.065±8.1e-3	0.872±6.3e-4	0.825±4.1e-4	0.730±3.5e-4	5.216±2.0e-2	0.924±1.2e-4	0.974±1.9e-4	0.744±2.1e-4
MLP-PLR	0.700±2.1e-3	0.858±1.6e-3	0.968±5.9e-3	0.453±5.8e-4	2.975±6.6e-3	0.874±9.0e-4	0.830±2.4e-3	0.734±3.5e-4	5.388±1.6e-2	0.924±5.4e-5	0.975±4.8e-4	0.743±1.0e-4
MLP-PLRLR	0.699±9.3e-3	0.867±1.8e-3	0.997±1.5e-3	0.448±8.3e-4	2.993±6.5e-3	0.873±4.1e-4	0.823±8.3e-4	0.729±9.1e-4	5.346±4.8e-2	0.924±2.6e-4	0.972±8.2e-4	0.743±9.9e-5
MLP-AutoDis	0.676±7.6e-3	0.860±1.7e-3	0.660±7.3e-3	0.464±1.6e-3	3.132±5.7e-3	0.860±2.8e-4	0.817±2.1e-3	0.730±2.5e-4	5.580±2.2e-2	0.924±1.1e-4	0.970±3.2e-4	–
ResNet	0.690±5.9e-3	0.861±1.6e-3	0.667±5.2e-3	0.483±1.8e-3	3.081±7.8e-3	0.856±3.4e-4	0.821±1.8e-3	0.734±1.1e-3	5.482±1.1e-2	0.918±5.3e-4	0.968±4.4e-4	0.745±6.5e-5
ResNet-L	0.674±5.2e-3	0.859±6.2e-4	0.689±5.8e-3	0.481±2.5e-3	3.025±1.8e-2	0.857±2.9e-4	0.819±1.3e-3	0.735±5.2e-4	5.522±2.4e-2	0.917±2.2e-4	0.966±5.1e-4	0.744±3.0e-4
ResNet-LR	0.672±6.0e-3	0.862±1.7e-3	0.735±3.0e-3	0.450±2.2e-3	2.992±2.4e-2	0.859±4.7e-4	0.822±9.2e-4	0.733±4.2e-5	5.415±9.5e-5	0.923±7.7e-5	0.971±1.5e-4	0.743±2.1e-4
ResNet-Q	0.671±1.7e-3	0.862±8.2e-4	0.697±9.5e-3	0.442±8.0e-4	3.128±9.0e-3	0.862±5.8e-4	0.816±9.4e-4	0.722±7.1e-4	5.402±3.3e-2	0.923±4.6e-4	0.974±6.3e-5	0.746±2.4e-4
ResNet-Q-LR	0.674±2.5e-3	0.859±1.8e-3	0.794±1.2e-2	0.427±2.3e-3	3.066±2.2e-2	0.868±1.1e-3	0.815±7.1e-4	0.729±1.6e-3	5.309±4.9e-2	0.923±3.9e-4	0.976±1.2e-4	0.746±1.8e-4
ResNet-T	0.681±1.3e-3	0.861±2.1e-3	0.742±4.8e-3	0.428±8.0e-4	3.064±3.6e-2	0.868±8.3e-4	0.823±4.0e-4	0.725±9.5e-4	5.657±1.5e-2	0.923±1.0e-4	0.973±6.0e-4	0.746±6.0e-4
ResNet-T-LR	0.683±6.1e-3	0.862±0.0e+00	0.817±2.1e-3	0.425±7.4e-4	3.030±3.4e-2	0.872±7.3e-4	0.822±5.5e-4	0.731±1.1e-3	5.471±9.2e-3	0.923±5.8e-4	0.975±1.0e-4	0.744±3.3e-4
ResNet-P	0.675±4.2e-3	0.860±6.2e-4	0.755±6.8e-3	0.453±3.1e-3	3.041±1.7e-2	0.872±1.4e-3	0.820±2.0e-4	0.733±5.0e-4	5.305±2.3e-2	0.923±3.6e-4	0.972±2.1e-4	0.744±1.7e-4
ResNet-PLR	0.691±6.3e-3	0.861±4.1e-4	0.925±7.7e-3	0.443±1.4e-3	3.040±2.1e-2	0.874±5.0e-4	0.825±1.1e-3	0.734±6.3e-4	5.400±2.6e-2	0.924±2.9e-4	0.975±9.1e-5	0.743±4.0e-4
Transformer-L	0.668±1.3e-2	0.861±6.2e-4	0.769±6.3e-3	0.455±1.4e-3	3.188±8.8e-3	0.860±6.5e-4	0.824±4.6e-4	0.727±1.1e-3	5.434±2.3e-2	0.924±1.1e-4	0.973±2.0e-4	0.743±2.7e-4
Transformer-LR	0.666±1.0e-3	0.861±4.1e-4	0.776±4.5e-3	0.446±1.1e-3	3.193±1.6e-2	0.861±2.0e-4	0.824±1.6e-3	0.733±7.8e-4	5.430±3.0e-2	0.924±1.8e-4	0.973±1.0e-4	0.743±1.8e-4
Transformer-Q-L	0.704±1.5e-3	0.861±1.1e-3	0.796±4.1e-3	0.426±1.6e-3	3.183±2.5e-2	0.869±2.7e-4	0.820±3.1e-3	0.735±1.5e-3	5.553±1.5e-2	0.925±2.8e-4	0.976±5.9e-5	0.744±2.0e-4
Transformer-Q-LR	0.690±1.9e-3	0.857±2.4e-4	0.842±6.1e-3	0.425±1.2e-3	3.143±1.6e-2	0.868±4.9e-4	0.818±2.3e-3	0.726±1.2e-3	5.471±1.5e-2	0.924±2.0e-4	0.975±1.9e-4	0.744±3.5e-4
Transformer-T-L	0.693±6.8e-3	0.862±2.4e-4	0.805±5.3e-3	0.439±1.0e-3	3.136±3.5e-3	0.872±1.3e-4	0.826±2.3e-3	0.731±1.6e-3	5.579±5.2e-2	0.924±4.0e-4	0.977±2.1e-4	0.743±2.3e-4
Transformer-T-LR	0.686±4.1e-3	0.862±1.1e-3	0.833±6.4e-3	0.423±3.4e-3	3.149±1.4e-2	0.871±8.0e-4	0.823±2.4e-3	0.733±9.4e-4	5.515±2.0e-2	0.924±6.1e-5	0.976±9.2e-5	0.744±2.9e-4
Transformer-PLR	0.686±6.2e-3	0.864±9.4e-4	0.977±3.9e-3	0.449±1.2e-3	3.091±1.3e-2	0.873±1.5e-3	0.823±1.7e-3	0.734±2.1e-4	5.581±6.4e-2	0.924±1.8e-4	0.975±2.2e-4	0.743±2.4e-4