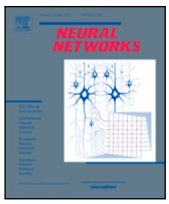
Neural Networks 173 (2024) 106180

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Full Length Article

Graph Neural Network contextual embedding for Deep Learning on tabular data

Mario Villaizán-Vallelado [a,b,*], Matteo Salvatori [a], Belén Carro [b], Antonio Javier Sanchez-Esguevillas [b]

[a] Artificial Intelligence Laboratory (AI-Lab), Telefonica I+D, Spain
[b] Universidad de Valladolid, Valladolid, 47011, Spain

ARTICLE INFO

Keywords:
Deep Learning
Graph Neural Network
Interaction Network
Contextual embedding
Tabular data
Artificial Intelligence

ABSTRACT

All industries are trying to leverage Artificial Intelligence (AI) based on their existing big data which is available in so called tabular form, where each record is composed of a number of heterogeneous continuous and categorical columns also known as features. Deep Learning (DL) has constituted a major breakthrough for AI in fields related to human skills like natural language processing, but its applicability to tabular data has been more challenging. More classical Machine Learning (ML) models like tree-based ensemble ones usually perform better. This paper presents a novel DL model using Graph Neural Network (GNN) more specifically Interaction Network (IN), for contextual embedding and modeling interactions among tabular features. Its results outperform those of a recently published survey with DL benchmark based on seven public datasets, also achieving competitive results when compared to boosted-tree solutions.

1. Introduction

Many practical real-world applications store data in tabular form, i.e. samples (rows) with the same set of attributes (columns). Medicine, finance or recommender systems are some common examples.

DL success in tasks involving texts, images or audio has sparked interest in its possible application to tabular data. Nevertheless, this success is often achieved when the input data are homogeneous and the structure used to organize the information provides insights about the data understanding. All tokens in a sentence are instances of the same categorical variable and their layout has semantic significance. Pixels in an image are continuous and usually have spatial correlation.

Tabular data have two characteristics that hinder DL performance. On one hand, tabular features are heterogeneous, having a mix of continuous and categorical distributions that may correlate or be independent. On the other hand, the meaningfulness of tabular data row is independent of the column order, i.e. position is arbitrary and does not provide information.

Tree-based ensemble models such as XGBoost (Chen & Guestrin, 2016), CatBoost (Prokhorenkova, Gusev, Vorobev, Dorogush, & Gulin, 2018), and LightGBM (Ke et al., 2017) achieve the state of the art (SOTA) performance on tabular data: they have competitive prediction accuracy and are fast to train. However, further research and development of DL models for tabular data are motivated, by the fact that standard tree-based approaches have limitations, for example, in case of continual learning, reinforcement learning or when tabular data is only part of the model input, which also includes data such as images, texts or audio.

Inspired by the success of contextual embedding in large language models (for example BERT - Bidirectional Encoder Representations from Transformers (Devlin, Chang, Lee, & Toutanova, 2019)), several recent researches (Gorishniy, Rubachev, Khrulkov, & Babenko, 2021; Huang, Khetan, Cvitkovic, & Karnin, 2020; Somepalli, Schwarzschild, Goldblum, Bruss, & Goldstein, 2022) have investigated how to enhance tabular feature representation (and hence global DL model performance) by taking into consideration their context, that is, feature interaction. The results obtained in these works, as well as the outcomes of recent comparisons on many public datasets (Borisov et al., 2022), illustrate how the contextual embedding approach tends to outperform not only standard Multi-Layer Perceptron (MLP) models, but also more complex models developed to solve complicated tasks (Cheng et al., 2016; Guo, Tang, Ye, Li, & He, 2017; He et al., 2017; Naumov et al., 2019; Wang et al., 2021) or models combining DL architectures with standard ML approaches (Arik & Pfister, 2021; Popov, Morozov, & Babenko, 2019).

Many of the most recent studies employ Transformer (Vaswani et al., 2017) as a method for contextual embedding. However, in this

* Corresponding author at: Universidad de Valladolid, Valladolid, 47011, Spain.
E-mail addresses: mario.villaizan@uva.es (M. Villaizán-Vallelado), matteo.salvatori@telefonica.com (M. Salvatori), belen.carro@uva.es (B. Carro), antoniojavier.sanchez@uva.es (A.J. Sanchez-Esguevillas).

https://doi.org/10.1016/j.neunet.2024.106180
Received 28 July 2023; Received in revised form 29 January 2024; Accepted 13 February 2024
Available online 16 February 2024

paper, we look at how to use a GNN to improve contextual embedding for tabular data. GNNs are a special subset of neural networks that are capable of managing information organized in a graph which is a structure with variable shape or size and with complex topological relations. One of the most important features of a graph is that its meaning does not depend on the order of its nodes, just as the meaning of a tabular row does not depend on the order of its columns.

Contributions. The contributions of our paper are summarized as follows:

- We introduce Interaction Network Contextual Embedding (INCE), a DL model for tabular data that employs GNNs and, more specifically, Interaction Networks (Battaglia et al., 2018; Battaglia, Pascanu, Lai, Rezende, & Kavukcuoglu, 2016; Sanchez-Gonzalez, Godwin, Pfaff, Ying, Leskovec, & Battaglia, 2020) for contextual embedding. First, all features (categorical and continuous) are individually projected in a common dense latent space. The resultant feature embedding is organized in a fully-connected graph with an extra virtual node, called <CLS> as in BERT (Devlin et al., 2019). Then, a stack of INs models the relationship among all the nodes (original features and <CLS> virtual node) and enhances their representation. The resulting <CLS> virtual node is sent into the final classifier/regressor. For sake of reproducibility, we share an implementation of INCE[1,2].
- We compare INCE against a wide range of deep tabular models and generally used tree-based approaches, using as benchmark the tabular datasets provided in Borisov et al. (2022) plus two new datasets with a large number of features. INCE outperforms all other DL methods on average, and it achieves competitive results when compared to boosted-tree solutions.
- We analyze which aspects of the GNN contextual embedder design are more significant for the model performance. More in details, we compare contextual embedding as complexity increases: MLP without graph embedding → GCN (Kipf & Welling, 2017) → GAT (Veličković, Cucurull, Casanova, Romero, Lio, & Bengio, 2018) → Transformer Encoder → IN. In each step, a new model feature is incrementally added resulting in a continuously superior performance on the benchmark of this paper. Moving from MLP to GNN implies introducing the representation of the tabular row as a fully connected graph and the use of a GNN-like contextual embedder that takes advantage of graph topology without discriminating the importance of existing edges. GAT/Transformer Encoder/IN utilize some soft-link-prune, that is they learn graph edge weights and use them to differentiate the neighborhood contribution to contextual node update. Finally moving from GAT → Transformer Encoder → IN the complexity of mechanism able to learn the edge weights is increased. The main findings can be summarized in the following: (1) Represent a tabular row as a fully-connected-graph. This enables the use of GNN-like contextual embedding which has less learning bias than a standard MLP (2) Choose GNN-like contextual embedders, that can explicitly model the edge strength (GAT/Transformer/IN better than GCN): not all features interact in the same manner, and distinguishing the strength of the different interactions is crucial for determining which neighborhoods are more relevant for the current-feature final representation (3) Increase the complexity of the mechanism modeling the edge strength (IN better than Transformer, Transformer better than GAT).
- We thoroughly investigate the differences between contextual embeddings based on Transformer Encoder and IN and analyze the influence of IN hyperparameters on model performance: quality of results, model size, computational time. Regardless of the dataset or task challenge, we gain a collection of patterns that aid in the establishment of a strong baseline.
- We investigate the interpretability of the IN ensuing contextual embeddings. On the one hand, we focus on the feature–feature relationship discovered by the IN, while on the other hand, we concentrate on how contextual embeddings improve traditional context-free embeddings.

[1] https://github.com/MatteoSalvatori/INCE
[2] https://codeocean.com/capsule/2256574

2. Related work

Standard Tabular Models. As already commented, when dealing with tabular data, tree-based ensemble models such as XGBoost, CatBoost and LightGBM are often a popular choice. They usually provide high performance regardless of the amount of data available, as they can handle many data types, are resilient in the case of null values, are fast to train and can be interpreted at least globally.

Deep Tabular Models. Due to the success of DL in task involving texts, sound or images, many efforts are being made to find the best approach to apply these models to tabular data (Arik & Pfister, 2021; Gorishniy et al., 2021; Huang et al., 2020; Joseph & Raj, 2022; Kotelnikov, Baranchuk, Rubachev, & Babenko, 2023; Somepalli et al., 2022). Most of these efforts belong to one of the 3 categories described below.

Modeling of multiplicative interactions between features Modeling explicitly the interaction between features of a tabular dataset (Cheng et al., 2016; Guo et al., 2017; He et al., 2017; Naumov et al., 2019; Wang et al., 2021) has been shown to have a significant impact on the performance of deep learning models in applications such as recommender systems and click-through-rate prediction. Nevertheless, recent comparisons (Borisov et al., 2022; Gorishniy et al., 2021) show that these approaches produce worse outcomes than the rest of categories described below.

Hybrid models. Hybrid models transform the tabular data and combine deep neural networks with classical ML approaches, frequently decision trees. Those kind of hybrid models can be designed to be optimized in a fully-differentiable end-to-end pipeline or to benefit from non-differentiable approaches combined with deep neural networks. NODE (Popov et al., 2019) is partially inspired by CatBoost (Prokhorenkova et al., 2018) and provides an example of fully differentiable model based on an ensemble of oblivious decision trees (Langley & Sage, 1994). Entmax transformation and soft splits allow to obtain a fully differentiable end-to-end optimization. Other examples of fully-differentiable hybrid architecture are (Frosst & Hinton, 2017; Katzir, Elidan, & El-Yaniv, 2021; Luo, Cheng, Yu, & Yi, 2021). On the other hand, DeepGBM model (Ke, Xu, Zhang, Bian, & Liu, 2019) is an example of how to take advantage from the combination of non-differentiable approaches with deep neural networks. It combines deep neural network flexibility with gradient boosting decision tree preprocessing capabilities. TabNN (Ke, Zhang, Xu, Bian, & Liu, 2019) first distills the knowledge from gradient boosting decision trees to retrieve feature groups and then constructs a neural network based on feature combinations produced by clusterizing the results of the previous step.

Transformer-based models. Many of DL recent successes have been driven by the use of transformer-based methods (Devlin et al., 2019; Dosovitskiy et al., 2021; Radford et al., 2018) inspiring the proposal of multiple approaches using deep attention mechanisms (Vaswani et al., 2017) for heterogeneous tabular data. The TabNet (Arik & Pfister, 2021) design is inspired by decision trees: a set of subnetworks is processed in a hierarchical order and the results of all decision steps are aggregated in order to obtain the final prediction. A feature transformer module chooses which features should be transferred to the next decision step and which should be employed to get the output at the present decision phase. TabTransformer (Huang et al., 2020) uses Transformer to improve the contextual embeddings of tabular features. First, each categorical variable goes through a specific embedding layer. A stack of Transformers is then used to enhance the categorical feature representation. The final contextual embedding is given by the concatenation of the so obtained categorical representation and the initial continuous

features. In FT-Transformer (Gorishniy et al., 2021), columnar transformations (embeddings) are applied to both categorical and continuous features. As in BERT (Devlin et al., 2019), a <CLS> token is added to the set of columnar embeddings and then, a stack of transformer layers, are applied. The final <CLS> representation is employed as final contextual embedding, i.e. for predictions. SAINT (Somepalli et al., 2022) combines the self-attention between features of the same tabular row with inter-sample attention over multiple-rows. When handling missing or noisy data, this mechanism allows the model to borrow the corresponding information from similar samples.

As in Gorishniy et al. (2021), Huang et al. (2020), Somepalli et al. (2022), we investigate how contextual embedding affects the final model performance on supervised tasks. The main difference from the existing research is that in our approach, the contextual embedding is provided via GNNs and, more specifically, by INs.

Graph Neural Network and Interaction Network. In case of neural networks such as Convolutional Neural Network or Transformer, the inputs must be structured data (grid and sequence, respectively). GNN are a special subset of neural networks that can cope with less structured data, such as a graph. This means that the input can have arbitrary shapes and sizes, and can have complex topological relations. Permutation invariance is a crucial feature distinguishing GNN from the rest of neural networks. The order of nodes in a graph has no relevance, this means, that the way in which we order the nodes in a graph does not impact the results produced by GNNs. In a tabular dataset, the order of features (columns) does not have any meaning, so GNN is a good candidate to model the interaction between them.

The flow of a GNN can be modeled using the Message-Passing scheme. (a) For each pair of nodes (u, v) in the graph, a message $M\left(u, v, e_{u,v}\right)$ from v to u is created. Here u, v are the embedding of nodes and $e_{u,v}$ is the (optional) embedding of edge. (b) Each node aggregates the messages coming from all its neighbors. The aggregation must be permutation-invariant. (c) The node is updated using its initial representation and the information obtained in point b.

It is simple to find a map between the Message-Passing scheme and the contextual embedding of tabular features. (a) Initial node representation is given by columnar feature embeddings. (b) Message-passing through edges is the pairwise interaction between features. (c) The neighbor aggregation represents the effect of the interaction of current feature with all its neighbors. (d) The update step provides the contextual representation of each feature.

In this paper, we investigate the benefits of using INs for contextual embeddings of tabular data. They are a low-biased family of GNN that have obtained enormous success when applied to simulation of complex physics or weather forecasting (Lam et al., 2023).

The potential of GNNs has attracted the community interest, and various attempts have been made to apply this type of solution to tabular data. To the best of our knowledge, past research has mostly focused on utilizing GNN to learn relationships between samples in the same table or in distinct entities of a relational database. On the contrary, *in our method we prioritize modeling feature relationships*. The approaches are complimentary, and we leave it to future research to figure out how to integrate them.

TabGNN (Guo et al., 2021) focuses on modeling the relation between samples of the same table. Using a set of heuristics, a multiplex graph (i.e. a graph modeling different types of relations between nodes) is previously built from sample features. A specific GNN obtains a customized sample representation for each edge type (i.e. for each type of node-to-node relation) and then an attention mechanism combines all contributions. This mechanism can be used in conjunction with other embedding strategies. In Du et al. (2022), to model the cross-sample and cross-column patterns a hypergraph is built from relevant data instance retrieval. Then a novel architecture of message-passing enhances the target data representation. Finally, in Bai et al. (2021), Cvitkovic (2020), GNNs are used to automatize and improve the features extraction in a relational database with a set of tables and foreign keys relationships.

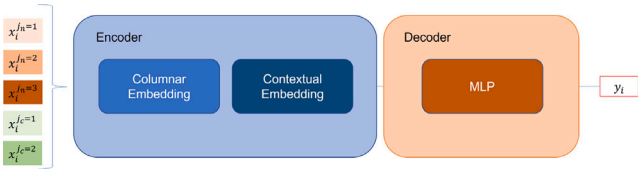


Fig. 1. The encoder–decoder perspective (Hamilton, 2020): an encoder model maps each tabular dataset feature into a latent vector, a decoder model uses the embeddings to solve the supervised learning task. In the encoding step, first a *columnar* embedding individually projects any feature in a common latent space and then a *contextual* embedding improves these representations taking into account the relationships among features. The decoder MLP transforms the *contextual* embedding output in the final model prediction.

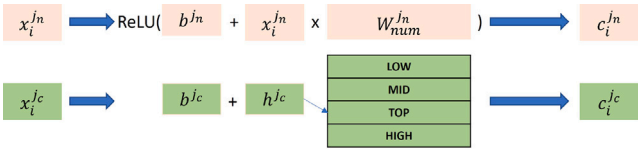


Fig. 2. The *columnar* embedding is responsible for projecting all the heterogeneous features in the tabular dataset in a common latent space. For each feature, a continuous or categorical transformation is defined. The *columnar* embedding ignores any potential relationship or similarity between the tabular dataset features.

3. Interaction Network Contextual Embedding

This section introduces the INCE model and describes its components in depth.

Problem Definition. We focus on supervised learning problems with tabular datasets $D = \left\{ x_i^{j_c}, x_i^{j_n}, y_i \right\}_{i=1}^{N}$ where $x_i^{j_n}$ with $j_n \in [1, M_{num}]$ is the set of numerical features, $x_i^{j_c}$ with $j_c \in \left[1, M_{cat}\right]$ is the set of categorical features, y_i is the label, $i \in [1, N]$ counts the dataset rows, N is the total number of rows and $M = M_{num} + M_{cat}$ is total number of features.

Encoder–Decoder Perspective.

As in Hamilton (2020), we use the encoder–decoder perspective, Fig. 1. First an encoder model maps each tabular dataset feature into a latent vector or embedding and then a decoder model takes the embeddings and uses them to solve the supervised learning task.

The encoder model is composed by two components: the *columnar* and the *contextual* embedding. The decoder model is given by a MLP tuned to the learning task to solve.

Encoder - Columnar Embedding. All of the original tabular heterogeneous features are projected in the same homogeneous and dense l-dimensional latent space by the *columnar* embedding depicted in Fig. 2. As in the (Gorishniy et al., 2021; Somepalli et al., 2022), the columnar embedding $c_i^{j_n}, c_i^{j_c} \in \mathbb{R}^l$ of continuous and categorical features $x_i^{j_n}, x_i^{j_c}$ are obtained as follows:

$$c_i^{j_n} = \text{ReLU}\left(b^{j_n} + x_i^{j_n} \cdot W_{num}^{j_n}\right) \qquad W_{num}^{j_n} \in \mathbb{R}^l \tag{1}$$

$$c_i^{j_c} = b^{j_c} + h_{j_c}^T W_{cat}^{j_c} \qquad W_{cat}^{j_c} \in \mathbb{R}^{|j_c| \times l} \tag{2}$$

where ReLU is the non-linear activation function for the continuous embedding, b^{j_n}, b^{j_c} are the feature bias, $W_{num}^{j_n} \in \mathbb{R}^l$ is a learnable vector, $W_{cat}^{j_c} \in \mathbb{R}^{|j_c| \times l}$ is a learnable lookup table and $|j_c|$ and $h_{j_c}^T$ are the size and the one-hot representation of the categorical feature $x_i^{j_c}$, respectively.

Encoder - Contextual Embedding. The *columnar* embedding works feature by feature and has trouble identifying correlation or more general relationships between features in tabular datasets. To overcome this limitation, a *contextual* embedding is introduced. In contrast to recent research (Arik & Pfister, 2021; Gorishniy et al., 2021; Huang et al., 2020; Somepalli et al., 2022) that use Transformer, we propose a *contextual* embedding based on GNN and, more specifically, IN (Battaglia et al., 2018, 2016; Sanchez-Gonzalez et al., 2020).

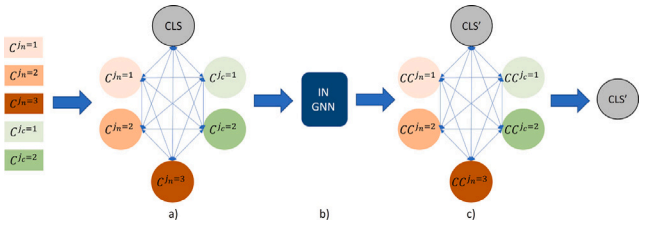


Fig. 3. *Contextual* embedding. (a) Homogeneous and fully-connected graph: it contains a node for each initial tabular feature and a bidirectional-edge for each pair of nodes. The initial node representation is obtained by the *columnar* embedding. A virtual <CLS> node is introduced to characterize the global graph state. (b) A stack of IN (Battaglia et al., 2016) models node interactions to create a more accurate representation of nodes (i.e. tabular features). (c) The final representation of the <CLS> virtual node is used as *contextual* embedding.

In this approach, the initial supervised learning task on tabular data is turned into a graph state estimation issue in which a categorical (classification task) or a continuous (regression task) graph state must be predicted. Taking into account the initial node representation (i.e. *columnar* embedding) and graph edges, a stack of GNNs has to model the interactions among nodes in the latent space and learn a richer representation of the entire graph capable of improving state estimation.

As shown in Fig. 3, the first step consists of building a fully-connected graph. For each original tabular feature, a node is created $n_j \equiv x_j$ and for each pair of nodes (n_{j_1}, n_{j_2}), two directed and independent edges are defined: $e_{j_1 j_2} : n_{j_1} \rightarrow n_{j_2}$ and $e_{j_2 j_1} : n_{j_2} \rightarrow n_{j_1}$. The dense $l-$dimensional vector $c_j \in \mathbb{R}^d$ obtained from the *columnar* embedding is used as initial node representation, giving rise to an homogeneous graph. No positional embedding is used to improve the node representation: the original tabular features are heterogeneous and each one is projected in the common latent space using a separate *columnar* embedding. This is enough to distinguish the nodes among them without explicitly modeling their position in the graph.[3] As in the BERT (Devlin et al., 2019), a virtual <CLS> node connected to each existing node is added to the graph. The $l-$dimensional initial representation of the <CLS> virtual node is a vector of learnable parameters. No features are initially considered for the edges e_{ij}.

In the following step, a stack of INs is used to improve the representation of each node and edge in the graph. The final <CLS> vector embedding produced by the stack of INs is used as global representation of the graph, i.e. as a *contextual* embedding of the tabular row.[4]

Interaction Network. The workflow of a standard IN layer (Battaglia et al., 2018, 2016) is described in Fig. 4. In the first step, the representation of each edge (i.e. interaction between each pair of tabular features) is updated using the information of the adjacent nodes (i.e. pair of tabular features):

$$e'_{j_1 \rightarrow j_2} = \text{MLP}_\text{E}\left(\text{Concat}\left(n_{j_1}, n_{j_2}, e_{j_1 \rightarrow j_2}\right)\right) \tag{3}$$

where $n_j, e_{j_1 j_2} \in \mathbb{R}^l$ are respectively node and edge representation, MLP_E is the shared neural network used to update all the graph edges. To simplify the notation we have suppressed the row index.

In the second step, all the messages coming from the incoming edges are aggregated and used to update the node representation:

$$n'_j = \text{MLP}_\text{N}\left(\text{Concat}\left(n_j, \sum_{k \in \mathcal{N}} e_{k \rightarrow j}\right)\right) \tag{4}$$

[3] We have explicitly tested this hypothesis and the experiments confirm that the use of positional embedding does not improve the model performance.

[4] We have explicitly examined several approaches of pooling the node representation learned by GNN. Our findings are consistent with the literature: the additional virtual <CLS> node method outperforms all the other proposals.

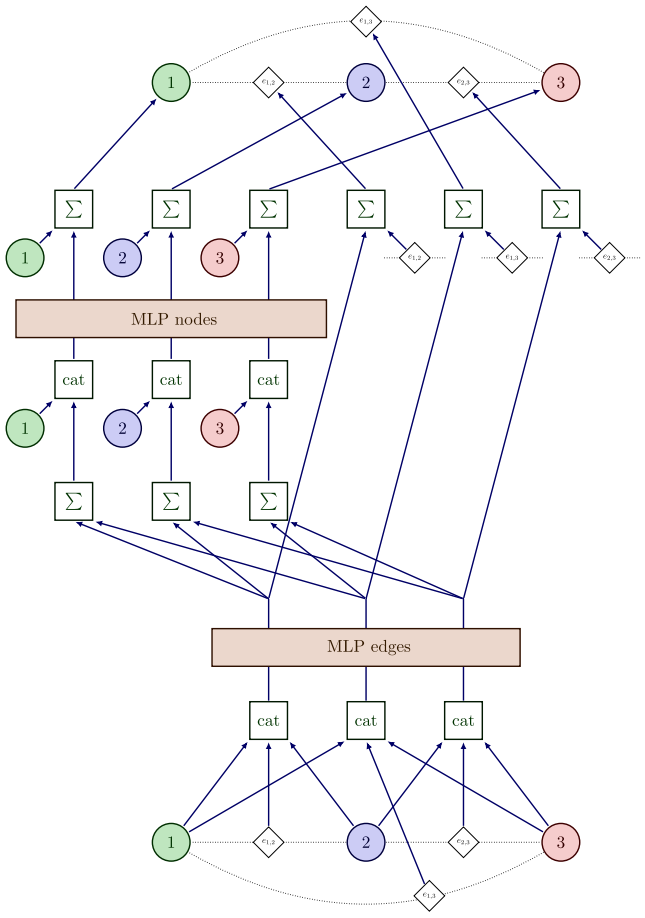


Fig. 4. Interaction Network layer. Graph nodes and edges are represented by circles and rhombuses, respectively. The operators cat and Σ denote concatenation and sum of latent space features, respectively. The workflow goes from bottom to top. In the first step, the representation of each edge is updated using the information of the adjacent nodes. In the second step, all the messages coming from the incoming edges are aggregated and used to update the node representation. The residual connection between the initial and updated representations yields the final node and edge.

where $\mathcal{N}$ is the set of n_j neighborhoods and MLP_N is the shared neural network used to update all the graph nodes.

The residual connection between the initial and updated representations yields the final node and edge representations:

$$\begin{aligned} n_j &= n'_j + n_j \\ e_{j_1 \rightarrow j_2} &= e'_{j_1 \rightarrow j_2} + e_{j_1 \rightarrow j_2} \end{aligned} \tag{5}$$

Decoder. The decoder MLP_DEC receives the contextual embedding computed by the encoder. It is a MLP where the final output layer size and activation function are adapted to the supervised learning problem to solve, classification or regression.

4. Experiments

Borisov et al. (2022) provides a detailed review on the literature of DL on tabular data together with an extensive empirical comparison of traditional ML methods and DL models on multiple real-world heterogeneous tabular datasets.

For our experiments, we reinforce the benchmark proposed therein with two new datasets (Katzir et al., 2021) with a significant number of features to investigate how the different models scale with the amount of features.

Data. The main properties of datasets are summarized in Table 1.

Table 1
Tabular benchmark properties.

Dataset	Rows	Num. Feats	Cat. Feats	Task
HELOC	9871	21	2	Binary
Gas Concentrations	13 910	129	0	Multi-Class (6)
California Housing	20 640	8	0	Regression
Adult Incoming	32 561	6	8	Binary
Otto Group	61 900	93	0	Multi-Class (9)
Forest Cover Type	581 K	10	2(4 + 40)	Multi-Class (7)
HIGGS	11 M	27	1	Binary

HELOC (FICO, 2019): Home Equity Line of Credit (HELOC) provided by FICO (a data analytics company), contains anonymized credit applications of HELOC credit lines. The dataset contains 21 numerical and two categorical features characterizing the applicant to the HELOC credit line. The task is a binary classification and the goal is to predict whether the applicant will make timely payments over a two-year period.

Gas Concentrations (Alex et al., 2012): The dataset contains measurements from 16 chemical sensors exposed to six gases at different concentration levels. It contains 13.9M of rows and 129 continuous features and the classification task is to determine which is the gas generating the data.

California Housing (Pace & Barry, 1997): The information refers to the houses located in a certain California district, as well as some basic statistics about them based on 1990 census data. This is a regression task, which requires to forecast the price of a property.

Adult Incoming (Becker & Kohavi, 1996): Personal details such as age, gender or education level, are used to predict whether an individual would earn more or less than 50k$ per year.

Otto Group (Bossan, Feigl, & Kan, 2015): The dataset provided by Otto Group (an e-commerce company) has 61.9 K of rows and 93 continuous product attributes, and the multi-class (9) classification problem consists of determining which category each product belongs to.

Forest Cover Type (Blackard, 1998): Cartographic variables are used to predict the forest cover type: it is a multi-class (seven) classification task. The first eight features are continuous whereas the last two are categorical, with four and 40 levels respectively.

HIGGS (Baldi, Sadowski, & Whiteson, 2014): The dataset contains 11M of rows and 28 features where the first 21 are kinematic properties measured by the particle detectors, and the last seven are processed features built by physicists. The data has been produced using Monte Carlo simulations and the binary classification task is to distinguish between signals with Higgs bosons and a background process.

Data Preprocessing. We reproduce the same data preprocessing described in Borisov et al. (2022). Zero-mean and unit-variance normalization is applied to the numerical features whereas an ordinal encoding is used for the categorical ones. The missing values were imputed with zeros.

Baselines. INCE is compared to the following models. *Standard methods*: XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017), CatBoost (Prokhorenkova et al., 2018). *Deep learning models*: MLP (McCulloch & Pitts, 1943), DeepFM (Guo et al., 2017), Tab-Trasformer (Huang et al., 2020), SAINT (Somepalli et al., 2022), FT-Transformer (Gorishniy et al., 2021). The baseline chosen are ones with the best performance in Borisov et al. (2022). It should be noted that we include in our study the FT-Transformer that is subsequent to Borisov et al. (2022).

Contextual-Embedding comparison. To emphasize how critical is the choice of the contextual embedding, we include in the comparison several versions of INCE where the *contextual* embedding is modified replacing the stack of INs by other architectures able to work with graph structured data. More in detail, all the INCE versions follow the workflow depicted in Fig. 1: they share *columnar* embedding and decoder and differ by the choice of the *contextual* embedding architecture. In our analysis, we explore both GNN models (GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), IN) and Transformer Encoders. Notice that the comparison includes two types of *contextual* embeddings, on the one hand, those which dynamically learn graph edge weights and use them to differentiate the neighborhood contribution to the node update (i.e. designs utilizing some soft-link-prune method) such as GAT, IN or Transformer Encoder and, on the other, those which take advantage of graph topology without discriminating the importance of existing graph edges such as GCN.

Setup. For each tabular dataset, we use the Optuna library (Akiba, Sano, Yanase, Ohta, & Koyama, 2019) with 50 iterations to fine-tune the hyperparameters. Each hyperparameter configuration is cross-validated with five folds. The search space for each baseline model as well as for each INCE version is described in Appendix B. All the DL code is implemented using PyTorch (Paszke et al., 2019) and PyTorch-Geometric (Fey & Lenssen, 2019) and parallelized with Ray (Moritz et al., 2018).

4.1. Results

In Table 2 we report the results on the tabular benchmark described above. In six of seven datasets, INCE outperforms all the DL baselines. In the seventh, HIGGS case, INCE ties with SAINT model (Somepalli et al., 2022), but largely above the rest of DL models. In two of the seven datasets, INCE outperforms tree-based models, while in the other five it achieves results that are competitive with them reducing the gap between tree-based and deep learning models as shown for example by the mean rank column of Table 2.

With respect to *contextual* embedding comparison, the main findings are the followings:

1. Regardless of the implementation, the *contextual* embedding allows to improve model performance. All the INCE versions outperforms MLP.
2. GAT, Transformer, and IN outperform GCN systemically. In our workflow the original tabular row is turned into a fully connected graph where each node is given by the *columnar* embedding of the original tabular features. The graph is fully-connected and all the edges have the same initial weight (i.e. importance). Nonetheless, GAT, Transformer Encoder and IN, all employ a mechanism to compute dynamically the weights to be assigned to each edge in the fully connected graph: attention mechanism in GAT and Transformer Encoder case and the edge update rule of Eq. (3) in the IN case. These weights allow discerning the relevance of each neighborhood contribution to the contextual embedding of each node. GCN, on the other hand, uses the graph topology to figure out which are the current node neighborhoods but without a way to differentiate between neighborhoods, all of them have the same relevance. In a fully connected graph, the situation is even worse since each node is a neighbor to every node and all have the same importance.
3. IN and Transformer Encoder get the best results being IN the best on the current benchmark and using the proposed workflow. In Section 5.1, we deeply analyze IN and Transformer Encoder, underlining similarities and differences between them.
4. The results are consistent regardless of dataset size and number of features. Regarding the latter, it is crucial to note that, while the 129 features of the Gas Concentration dataset may appear to be a large amount of features for a tabular dataset, a graph with 129 nodes remains a relatively tiny graph.

Table 2
Experiment results. Mean Squared Error (MSE) and accuracy are the metrics used in regression and classification case, respectively. An up/down arrow near the dataset name indicates whether the corresponding metric has to be minimized or maximized. For each dataset the first and second best results are highlighted using **bold** and underlined format. The last column computes the mean rank of each model across all datasets. Notice that with the resources available (Processor: i7-7700HQ (2.8 GHz), RAM: 32 GB, and GPU: GeForce GTX 1070 (8 GB)), the SAINT and DeepFM models could not be executed for datasets with a large number of features, such as Gas and Otto.

	HELOC ↑	Gas ↑	Cal. Hous. ↓	Adult Inc. ↑	Otto ↑	Forest Cov. ↑	HIGGS ↑	Mean Rank ↓
LightGBM	83.5 ± 0.2	99.2 ± 0.2	**0.195 ± 0.006**	**87.4 ± 0.3**	95.8 ± 0.1	93.4 ± 0.2	77.1 ± 0.0	4.14
XGBoost	83.7 ± 0.3	**99.3 ± 0.1**	0.198 ± 0.003	87.2 ± 0.3	**96.7 ± 0.2**	**97.2 ± 0.1**	77.6 ± 0.0	**2.57**
CatBoost	83.5 ± 0.4	99.2 ± 0.2	0.197 ± 0.005	87.1 ± 0.1	96.4 ± 0.1	96.3 ± 0.2	77.5 ± 0.0	3.43
MLP	73.3 ± 0.4	92.7 ± 0.3	0.284 ± 0.009	84.8 ± 0.1	88.5 ± 0.1	91.0 ± 0.4	76.9 ± 0.0	10.71
DeepFM	73.6 ± 0.2	–	0.260 ± 0.006	86.1 ± 0.2	–	92.1 ± 0.3	76.9 ± 0.0	6.57
TabTransformer	73.3 ± 0.1	95.1 ± 0.3	0.331 ± 0.008	85.2 ± 0.2	89.2 ± 0.2	76.5 ± 0.3	77.0 ± 0.0	10.43
FT-Transformer	83.8 ± 0.3	98.7 ± 0.1	0.228 ± 0.006	86.5 ± 0.3	94.5 ± 0.2	95.8 ± 0.1	78.5 ± 0.0	4.29
SAINT	82.1 ± 0.3	–	0.229 ± 0.005	86.1 ± 0.4	–	96.2 ± 0.2	**79.1 ± 0.0**	3.86
INCE-GCN	82.0 ± 0.2	98.5 ± 0.1	0.268 ± 0.003	85.9 ± 0.19	92.3 ± 0.1	93.3 ± 0.1	75.8 ± 0.0	9.29
INCE-GAT	82.2 ± 0.4	98.5 ± 0.1	0.234 ± 0.004	86.0 ± 0.2	93.8 ± 0.1	93.6 ± 0.1	77.7 ± 0.1	7.14
INCE-Transformer	83.8 ± 0.3	98.7 ± 0.1	0.228 ± 0.006	86.5 ± 0.3	94.5 ± 0.2	95.8 ± 0.1	78.5 ± 0.0	4.29
INCE	**84.2 ± 0.5**	99.1 ± 0.0	0.216 ± 0.007	86.8 ± 0.3	96.1 ± 0.1	97.1 ± 0.1	**79.1 ± 0.0**	2.71

5. Deep dive in Interaction Network

For each tabular dataset, we have studied how the choice of IN hyperparameters (latent space size l, $\text{MLP}_{\text{N, E}}$ depth d and number n of stacked INs) influences the model behavior: number of trainable parameters, performance and computational time. The findings from the various datasets reveal similar patterns, leading to consistent conclusions.

Trainable parameters. The number of trainable parameters $\mathcal{TP}(\text{IN})$ of a stack of n INs is given by:

$$\begin{aligned}\mathcal{TP}(\text{IN}) &= \sum_{i=1}^{n} \mathcal{TP}\left(\text{IN}^i\right) \\ &= \sum_{i=1}^{n} \left[\mathcal{TP}\left(\text{MLP}_{\text{E}}^{\text{i}}\right) + \mathcal{TP}\left(\text{MLP}_{\text{N}}^{\text{i}}\right)\right] \qquad (6)\end{aligned}$$

$$\mathcal{TP}\left(\text{MLP}_{\text{N}}^{\text{i}}\right) = \left(2 \cdot l^2 + l\right) + (d-1) \cdot \left(l^2 + l\right)$$

$$\mathcal{TP}\left(\text{MLP}_{\text{E}}^{\text{i}}\right) = \left(K_i \cdot l^2 + l\right) + (d-1) \cdot \left(l^2 + l\right)$$

where $K_i = 2$ if $i = 1$ and $K_i = 3$ otherwise. We consider all the hidden layers of $\text{MLP}_{\text{E, N}}$ of the same size. The difference in the number of parameters between $\text{MLP}_{\text{E}}^{i=1}$ and $\text{MLP}_{\text{E}}^{i>1}$ is due to the fact that all IN with $i > 1$ receive the edge features computed by preceding layers, whilst the first IN does not use any initial edge features.

The quantity of trainable parameters increases quadratically with the size of the latent space and linearly with the number of stacked INs or the $\text{MLP}_{\text{E, N}}$ depth, Fig. 5. The slope of the straight line corresponding to the number of stacked INs is steeper than the one relative to the $\text{MLP}_{\text{E, N}}$ depth.

Performance. Our experiments suggest that whereas the latent space size needs to be fine-tuned for each dataset, the impact of $\text{MLP}_{\text{E, N}}$ depth d and number n of stacked INs does not depend on the supervised learning problem to solve. The configuration with $d = 3$ and $n = 2$ is a solid baseline regardless of the underlying task.

To clarify this point, in Fig. 6 we show how the normalized metric changes as a function of the $\text{MLP}_{\text{E, N}}$ depth and the number of stacked INs. The normalized metric is a global performance measure (higher is better) generated using the findings from all of the datasets as described in Appendix A. The normalized metric is a global performance measure (higher is better) generated averaging the metrics from all of the datasets obtained using the algorithm described in Appendix A.

The left side plot in Fig. 6 depicts the normalized metric curves C_d (blue line) and C_n (orange line) obtained modifying d and n respectively while the other parameters are kept constant. The information on the right side plot is the same as on the left, but it is compared to the normalized number of trainable parameters.

The depth d of the shared neural networks $\text{MLP}_{\text{N, E}}$ has the most impact on the model performance and, at the same time, it has reduced effect on the number of learnable parameters. These results are coherent with the observed behavior of the Optuna (Akiba et al., 2019) bayesian optimizer. Regardless of the supervised learning problem, after few attempts, it quickly reduces search space for d to $[3, 4]$ and then it fine-tunes the number of stacked INs in the range $[2, 3]$. The configuration with $d = 3$ and $n = 2$ is always a solid candidate regardless of the tabular dataset.

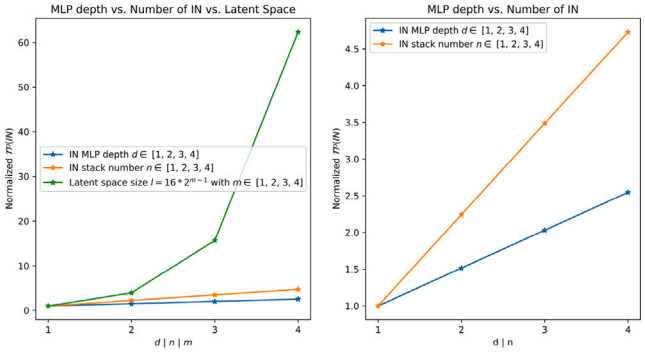


Fig. 5. Growth of the normalized $\mathcal{TP}(\text{IN})$ as a function of $\text{MLP}_{\text{E, N}}$ depth, number of stacked INs and latent space size. The plot on the left compares the evolution of $\mathcal{TP}(\text{IN})$ when two hyperparameters are fixed and the third is increased. The plot on right is a zoom on the contribution of $\text{MLP}_{\text{E, N}}$ depth and number of stacked INs. The baseline used to normalize $\mathcal{TP}(\text{IN})$ is given by the number of trainable parameters of the simplest case: $l = 16$, $d = 1$, $n = 1$. It is trivial to show using Eq. (6) that the behavior of normalized $\mathcal{TP}(\text{IN})$ curve does not depend on the particular choice of the baseline latent space size l.

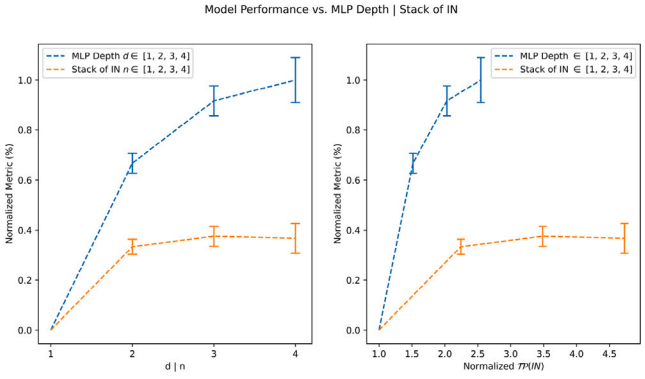


Fig. 6. Average normalized metric. Left side plot depicts how the normalized metric changes when the $\text{MLP}_{\text{E, N}}$ depth or the number of stacked INs is increased and the other is kept constant. The right side plot shows the same information but referenced to the normalized number of trainable parameters.

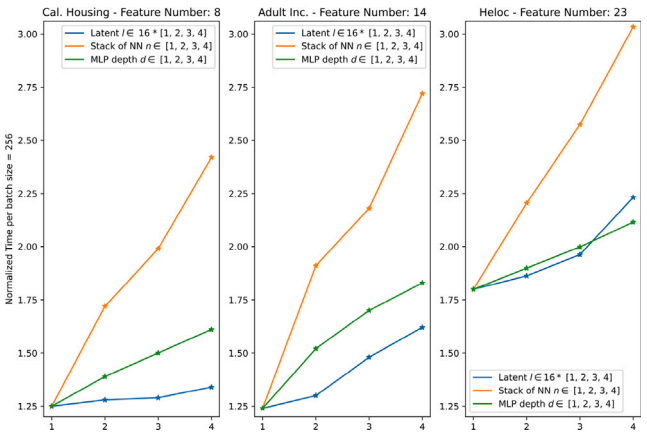


Fig. 7. Average normalized training time. For each dataset the INCE training time is normalized using the time of the corresponding MLP with the same columnar embedding and decoder but without contextual embeddings. All the results are relative to a batch size of 256. Starting from the configuration base $l = 16$, $n = 1$ and $d = 1$, the different curves are computed modifying one parameter while the others are kept constant.

Why adding more than two layers does not improve the *contextual* encoder capability? We interpret this as follows. (a) The number of nodes in the graph is small. In our formulation there is a node for each tabular feature and the number of them goes from eight (California Housing) to 129 (Gas Concentrations). After two IN layers, the information of a node has been transmitted to every other node in the graph. (b) We are working with a fully connected graph, i.e. a trivial topology. The IN has to model the strength of each edge but the initial topological information seems to be poor. (c) The size of datasets is limited (excluding HIGGS).

Computational time. Fig. 7 shows how the number of features in the tabular dataset as well as the INCE configuration (latent space size, number of stacked IN and $\text{MLP}_{\text{N, E}}$ depth) impact on the training time. In particular, Fig. 7 presents the average training time for a batch size of 256. All the INCE training times are normalized by using the corresponding train time of a MLP with the same columnar embedding and the same decoder but without contextual embeddings. For each dataset, the three curves are obtained modifying one parameter (for example $n \in \{1, 2, 3, 4\}$ for the orange line) while holding the other two constant ($l = 16$ and $d = 1$).

- As expected, the number of features in the tabular dataset has an effect on the computational time: it grows from California Housing (eight features) to Heloc (23 features) for a fixed INCE configuration. In our proposal, we are working with a fully-connected graph and the volume of operations increases quadratically in relation with the number of nodes (features).
- For a fixed dataset, the number n of stacked INs has the greatest impact on the amount of operations and, hence, on computational time.
- When the number of features is around 20, the impact of latent space size is comparable or even greater than the impact of $\text{MLP}_{\text{N, E}}$ depth.

5.1. Interaction Network vs. Transformer

Recent works Gorishniy et al. (2021), Huang et al. (2020), Somepalli et al. (2022) propose the Transformer encoder Vaswani et al. (2017) as *contextual* embedding. Here, we analyze similarities and differences between the two models.

Approach. In this work, we concentrate on the use case where either Transformer Encoders or GNNs are employed to learn the interaction between features improving the contextual embedding. For this particular use case, the following features are shared by both approaches:

- The columnar embeddings of each individual feature are organized in a fully-connected graph with an additional extra virtual node (<CLS>).
- A mechanism (the attention mechanism in the Transformer case and the IN convolution of Eqs. (3), (4) in our proposal) models the interaction between nodes/features. The strength of the interaction between nodes acts as a *soft prune mechanism*: the stronger the interaction between a neighborhood and the current node, the larger its contribution to the current node contextual embedding.

The main difference is in how the interaction is modeled. In the original Attention mechanism (Vaswani et al., 2017), the contextual node embedding (for head=1) is given by (neglecting for sake of simplicity the skip connection in both cases, Transformer Encoders and GNNs):

$$n_i'^{\alpha} = \sum_{j=1}^{M} \sum_{\beta=1}^{l} \omega_{i,j} V^{\alpha,\beta} n_j^{\beta} \tag{7}$$

where latin indexes $i, j = 1, \ldots, M$ are indexes in the topological space (that is over the graph nodes), Greek indexes $\alpha, \beta = 1, 2, \ldots, l$ are indexes in the latent space and $\omega_{i,j}$ is the attention mechanism:

$$\omega_{i,j} = \text{softmax}_j \left(\frac{n^i Q K^T n^j}{\sqrt{l}} \right) \tag{8}$$

Eq. (7) shows that the interaction between nodes n_i and n_j is written as the product of two operators $\omega_{i,j}$ and $V^{\alpha,\beta}$. The attention mechanism $\omega_{i,j}$ is an operator with no trivial structure in the topological space (it depends on the nodes indexes i and j) but diagonal in the latent space (it does not depend on the indexes in latent space). $V^{\alpha,\beta}$, on the contrary, is diagonal in the topological space (it does not depend on the node indexes) but with a non-trivial structure in the latent space (it depends on the latent space indexes α, β).

Using Eqs. (3), (4), it is possible to prove that, for the case of IN, the node contextual embedding is given by:

$$n_i'^{\alpha} = \text{MLP}_N \left(\text{Concat} \left(n_i, \sum_j \text{MLP}_E \left(n_i, n_j, e_{n_j \to n_i} \right) \right) \right) \tag{9}$$

This is a more general formulation than the case of the attention mechanism. MLP_N plays a similar role to the $V^{\alpha,\beta}$ operator of the Transformer Encoder and does not depend on the node indexes. The MLP_E plays a similar role to $\omega_{i,j}$. The main difference is that the MLP_E is a non-trivial operator in both topological and latent space and therefore, given a pair of nodes n_i and n_j, it may learn different strengths for different latent space indexes α, β.

Performance. As explained in Section 4, INCE and INCE-Transformer are two models that share the workflow of Fig. 1 and differ only by the *contextual* embedding: IN and Transformer Encoder, respectively. Table 2 shows how INCE and INCE-Transformer provide comparable results even though, at least on the selected benchmark, the IN encoder performs slightly better regardless of number of rows/features in the dataset. We interpret this result as a consequence of the previous discussion: IN employees a more general mechanism to discriminate how each neighborhood contributes to the *contextual* node update.

Trainable parameters. The size f of the latent space used by the Transformer FeedForward block has a significant impact on the number of trainable parameters in a Transformer Encoder. In the comparison[5] that follows, we take into account the setup where $f = 512$ since it achieves the best average results in the Optuna optimization.

[5] For the purpose of simplicity, we exclude the Normalization Layers parameters from our study in both cases, Transformer and IN.

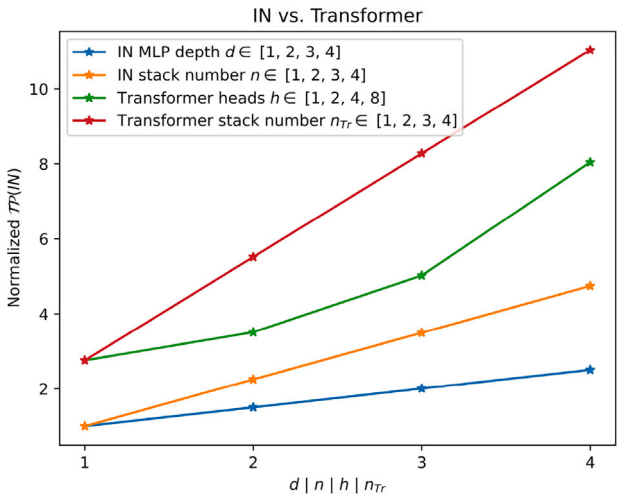


Fig. 8. Comparison of IN and Transformer trainable parameters. The normalized $\mathcal{TP}$ is obtained using Eqs. (6) and (10) and then normalizing with regarding to $\mathcal{TP}\left(IN_{l,d=1,n=1}\right)$. The plot shows the results for $l = 128$. Transformer has more trainable parameters than IN, and the relative difference grows when l decreases.

The number of trainable parameters of a Transformer Encoder is given by:

$$\begin{aligned} \mathcal{TP}(\text{Transformer}) &= n \cdot [\ \mathcal{TP}(Q,K,V) + \\ &= \quad \mathcal{TP}(\text{MultiAttention}) + \\ &= \quad \mathcal{TP}(\text{FeedForward})\ ] \\ \mathcal{TP}(Q,K,V) &= 3 \cdot h \cdot l \cdot (l+1) \\ \mathcal{TP}(\text{MultiAttention}) &= l \cdot (h \cdot l + 1) \\ \mathcal{TP}(\text{FeedForward}) &= 2 \cdot f \cdot l + f + l \end{aligned} \tag{10}$$

where l, h, f and n are respectively the latent space size, the number of attention heads, the FeedForward latent space size and the number of stacked Transformer Encoders.

Fig. 8 compares the behavior of $\mathcal{TP}$(Transformer) and $\mathcal{TP}$(IN). As in Fig. 5, the normalized number of trainable parameters $\mathcal{TP}$ is obtained dividing by $\mathcal{TP}\left(\text{IN}_{l,d=1,n=1}\right)$. Fig. 8 presents the results for $l = 128$. IN has less trainable parameters than Transformer and the relative difference is even bigger when l decreases. When the number of attention heads is $h \leq 2$, the difference is due to the FeedForward block parameters. For $h > 2$, Transformer has more parameters included, without taking into account the FeedForward block.

Limitations. When the number of tabular features increases, both IN and Transformer use greater resources. The vanilla Multi Head Self-Attention and IN on fully-connected graph share quadratic complexity regarding the number of features. This issue can be mitigated by using efficient approximations of Multi Head Self-Attention (Tay, Dehghani, Bahri, & Metzler, 2022) or a more complex graph topology with less edges in the Interaction Network case. Additionally, it is still possible to distill the final model into simpler architectures for better inference performance.

6. Interpretability of contextual embedding

6.1. Columnar vs. contextual embedding

In Section 4.1, the effect of *contextual* embedding on the model performance has been shown. INCE outperforms solutions that just use *columnar* embedding and, more generally, produces results that are on par with or even better than those of SOTA DL models when applied to tabular data.

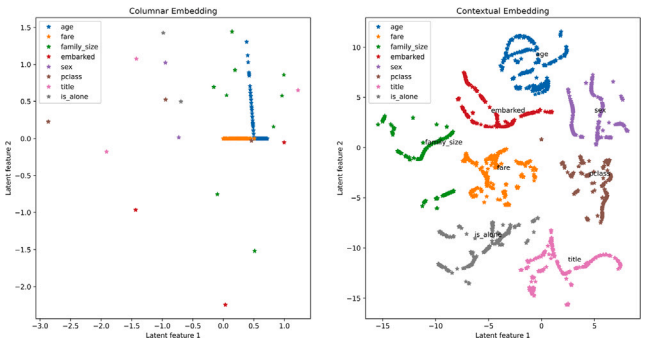


Fig. 9. Left: *Columnar* embedding before the stack of INs. **Right:** *Contextual* embedding from the last IN.

In this subsection, we visually examine how this mechanism improves the features representation, enhancing the performance of the final model. For sake of simplicity, in the following discussion, we use the Titanic (Dua & Graff, 2017) dataset. The supervised learning problem is a binary classification. The preprocessed dataset contains eight features. Age and fare are the zero-mean and one-standard-deviation continuous variables. The categorical features are sex $\in$ {female, male}, title $\in$ {Mr., Mrs., Rare}, pclass $\in$ {1, 2, 3}, family_ size $\in$ {0, 1, 2, 3, 4, 6, 7, 8}, is_alone $\in$ {0, 1}, embarked $\in$ {C = Cherbourg, Q = Queenstown, S = Southampton}. For this exercise, we consider an INCE model with latent space size $l = 128$, $\text{MLP}_{\text{N, E}}$ depth $d = 3$ and $n = 2$. Fig. 9 shows the output of *columnar* (left side plot) and *contextual* (right side plot) embedding after dimensional reduction.

The *columnar* embedding does not depend on the context: regardless of pclass, age or family_size values, title = Mrs is always projected to the same point in the latent space size.

The *contextual* embedding is given by the message sent from each node (i.e. tabular feature) to update the <CLS> representation in the last IN. It is feasible to see that the latent projections of categorical features are not yet limited to a fixed number of points when the context is taken into consideration, as shown, for example, by title embedding.

6.2. Feature importance from feature–feature interaction

The attention map for the <CLS> virtual node may be used to assess the feature relevance when the *contextual* embedding is a Transformer (Gorishniy et al., 2021; Somepalli et al., 2022). Here, we investigate if the feature–feature interaction that the IN learns can reveal details about the significance of tabular features. We first explain our methodology using the Titanic dataset for the purpose of simplicity, and then we illustrate the findings we achieved using the same technique on the other tabular datasets.

In contrast to the Transformer case, we now have two new problems to resolve: (1) The feature–feature interaction is a l-dimensional vector (that means, it is not a scalar); (2) To assess the feature global significance, we must aggregate the feature–feature importance. The description of our process is provided below.

First Step: We split data in train/test datasets. We train the model and use the trained INCE on the test dataset to produce the feature–feature interaction, i.e. $e^i_{j_1 \to j_2} \in \mathbb{R}^l$ in Eq. (5) returned by the last IN. In this notation, we have explicitly recovered the tabular row index i.

Second Step: We estimate mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{S}$ of the entire population $\left\{ e^i_{j_1 \to j_2} \right\}\ \forall\, i, j_1, j_2$. *Third Step:* For each pair (j_1, j_2) of features and for each test row i, we compute the squared Mahalanobis distance:

$$D_i^2\left(j_1 \to j_2\right) = \left(e^i_{j_1 \to j_2} - \mu\right) S^{-1} \left(e^i_{j_1 \to j_2} - \mu\right)$$

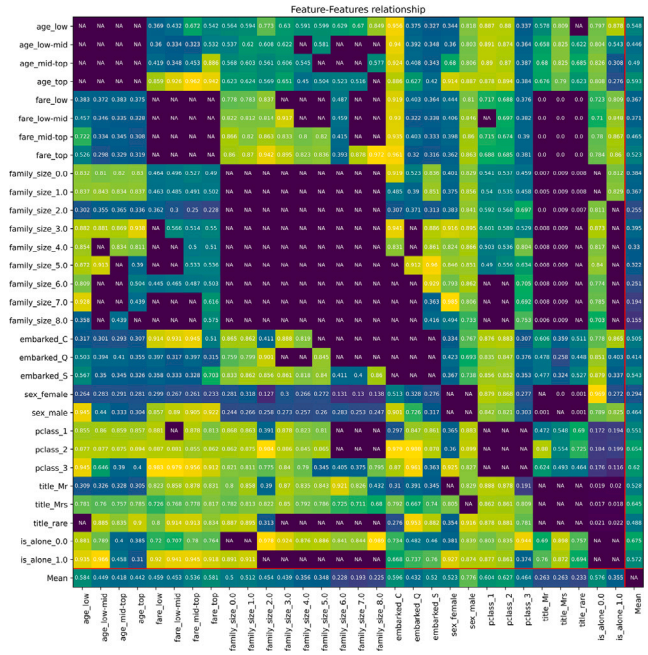


Fig. 10. Titanic feature–feature interaction at feature-value level.

Fourth Step: The squared Mahalanobis distance follows a Chi-Square distribution, so we can normalize the distance using p-value. The number of degrees of freedom of Chi-Square is given by the latent space size l:

$$p_i\left(j_1 \to j_2\right) = \Pr\left(D_i^2 \geq \chi_l^2\right)$$

Fifth Step: The global interaction p-value $p\left(j_1 \to j_2\right)$ is obtained averaging the previous results over the test dataset:

$$p\left(j_1 \to j_2\right) = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} p_i\left(j_1 \to j_2\right)$$

The findings of the proposed methodology on the Titanic dataset are displayed in the heatmap of Fig. 10. The results are broken down at the feature-value level (i.e. sex = female, sex = male, title = Mrs, title = Mr, etc.). This is how the heatmap may be understood: the relevance of the message from the row-r-feature to the column-c-feature is represented by the element (row=r, column=c) of the heatmap. A lower (blue) p-value implies more significance. The last column, "Mean", is created by averaging all of the row values and shows the average relevance of the messages sent by row-r-feature. In a similar way, the last row (also known as "Mean") is derived by averaging all the values of the columns and it represents the mean relevance of the messages received by column-c-feature.

In order to quantitatively assess the quality of the heatmap, we compute the Spearman Rank correlation ρ between

$$p(j) = \frac{1}{|\mathcal{N}|} \sum_{\hat{j} \in \mathcal{N}} p(j, \hat{j}) = \frac{1}{2}\left[p\left(j \to \hat{j}\right) + p\left(\hat{j} \to j\right)\right]$$

and the feature importance calculated by KernelShap (Lundberg & Lee, 2017). In the formula above, $\mathcal{N}$ and $|\mathcal{N}|$ are the set of neighbors of node j and its size, respectively. The outcome for the Titanic dataset is $\rho = 0.81$(p-value $= 0.05$).

The heatmap and the Spearman Rank correlation provide the following insights.

(a) The feature–feature interaction is not symmetric. In the fully connected graph we have two independent edges $j_1 \to j_2$ and $j_2 \to j_1$ and the Eq. (3) is not invariant by $j_1 \longleftrightarrow j_2$ interchange. Our experiments demonstrate that inducing $j_1 \longleftrightarrow j_2$ invariance in Eq. (3) results in a learning bias that negatively affects INCE performance.

Table 3
Spearman Rank Correlation between KernelShap and feature–feature interaction.

	HELOC	Cal. Hous.	Adult Inc.	Forest Cov.
ρ(p-value)	0.82(0.04)	0.80(0.06)	0.85(0.03)	0.81(0.04)

(b) From heatmap, it is possible to discern logical patterns. For example, is_alone = 1 does not add information (high p-value) when family_size is 0 or 1 and on the contrary, the value family_size is very relevant (low p-value) for any value of title.

(c) Considering that KernelShap evaluates global model behavior (including the decoder) and that IN models separately $\left(j_1 \to j_2\right)$ and $\left(j_2 \to j_1\right)$ and that we have to aggregate and average them to compare with KernelShap, the Spearman Rank correlation analysis result can be considered encouraging.

Finally, Table 3 summarizes the Spearman Rank correlation achieved on various datasets and demonstrates how the results are consistent regardless of the dataset under consideration.

7. Conclusions

Let us highlight the main contributions of this article:

- As far as we know, this is the first time that model architecture proposes the use of GNN for contextual embedding to solve supervised tasks involving tabular data.
- Literature discusses mainly about the usage of Transformer. This manuscript shows that GNN, particularly IN, are a valid alternative. It shows better performance with a lower number of training parameters.
- As a matter of fact, this innovative architecture outperforms the state of the art DL benchmark based on 7 different diverse datasets. Moreover, it closes the gap with classical ML models (tree-based), outperforming them in 2 of these datasets, and being very close in two more. The tradeoff versus tree-based models is additional computational load in the form of training time, and scalability issues with the number of features (nodes) of the dataset, which constitute future lines of research to keep improving its practical implementation.
- Finally, the interpretability of GNN is explored. This is a key topic for industry environments, and apparently this is the first study for GNN and tabular data.

CRediT authorship contribution statement

Mario Villaizán-Vallelado: Conceptualization, Formal analysis, Investigation, Methodology, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Matteo Salvatori:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Belén Carro:** Funding acquisition, Project administration, Supervision, Writing – review & editing. **Antonio Javier Sanchez-Esguevillas:** Funding acquisition, Project administration, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is public. The code is public on github. The code is in CodeOcean.

Acknowledgments

Grant PID2021-122210OB-I00, funded by MCIN/AEI/10.13039/501100011033 and "ERDF A way of making Europe", European Union.

Appendix A. Normalized metric

Algorithm 1 Normalized metric

Input: l : latent space, r : dataset
Output: C_d, C_n two lists of normalized metric

$base \leftarrow Metric_r(d = 1, n = 1, l, r)$
$C_d \leftarrow Metric_r(d, n = 1, l, r) \quad \forall d \in \{1, 2, 3, 4\}$
$C_n \leftarrow Metric_r(d = 1, n, l, r) \quad \forall n \in \{1, 2, 3, 4\}$
$best \leftarrow Best_r(C_d, C_n)$
$C_d \leftarrow \frac{C_d - base}{best - base} \quad \forall d \in \{1, 2, 3, 4\}$
$C_n \leftarrow \frac{C_n - base}{best - base} \quad \forall n \in \{1, 2, 3, 4\}$
return C_d, C_n

$\forall l, r/\ r \in \{$ HELOC, Gas, Cal. Hous., Adult Inc., Otto, Forest Cov., HIGGS $\}$, $l \in \{16, 32, 64, 128\}$. In Algorithm 1 $Metric_r$ and $Best_r$ are Accuracy/MSE and max/min depending on r, d is the $\text{MLP}_{\text{N, E}}$ depth and n is the number of stacked IN. Notice that computing $Metric_r$ means train–test the model 5 times with different seeds and average the results.

The curves of Fig. 6 are obtained by computing the average and the standard-deviation from results of Algorithm 1.

Appendix B. Hyperparameters search space

Baseline - Tree Based models.

- LightGBM: leaves number $\in [2, 4096]$, $\lambda_{l1} = \text{Uniform}\left(10^{-8}, 10\right)$, $\lambda_{l2} = \text{Uniform}\left(10^{-8}, 10\right)$, learning rate = $\text{Uniform}\,(0.01, 0.3)$.
- XGBoost: max depth $\in [2, 12]$, $\alpha = \text{Uniform}\left(10^{-8}, 1\right)$, $\lambda = \text{Uniform}\left(10^{-8}, 1\right)$, $\eta = \text{Uniform}\,(0.01, 0.3)$.
- CatBoost: max depth $\in [2, 12]$, learning rate = $\text{Uniform}\,(0.01, 0.3)$, l2 leaf reg = $\text{Uniform}\,(0.5, 30)$.

Baseline - Deep Learning models.

- MLP: latent space size $l \in \{32, 64, 128\}$, hidden dim $hd \in \{64, 128, 256, 512\}$, layer number $ln \in [2, 8]$.
- DeepFM: latent space size $l \in \{32, 64, 128\}$, layer number $ln \in [2, 8]$, dropout $\in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.
- TabTransformer: latent space size $l \in \{32, 64, 128\}$, number of attention heads $h \in \{1, 2, 4, 8\}$, transformer depth $d \in [1, 12]$, dropout $\in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.
- FT-Transformer: latent space size $l \in \{32, 64, 128\}$, number of attention heads $h \in \{1, 2, 4, 8\}$, transformer depth $d \in [1, 12]$, dropout $\in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.
- SAINT: latent space size $l \in \{32, 64, 128\}$, number of attention heads $h \in \{1, 2, 4, 8\}$, transformer depth $d \in [1, 12]$, dropout $\in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.

Cross-Entropy and Mean Squared Error (MSE) are the loss functions used in classification and regression tasks, respectively. We train all the models 200 epochs using Adam optimizer with a learning rate of 0.001 and with batches of size 256.

INCE. All the INCE versions used in the comparison follow the workflow depicted in Fig. 1: they share *columnar* embedding and decoder and differ by the *contextual* embedding implementation: GCN, GAT, IN or Transformer Encoder. The search space for the *contextual* embedding hyperparameters is the following:

- GCN: number of stacked GCN encoders $n \in [1, 4]$ and latent space size $l \in \{16, 32, 64, 128\}$.
- GAT: number of attention heads $h \in \{1, 2, 4, 8\}$, number of stacked GAT encoders $n \in [1, 4]$ and latent space size $l \in \{16, 32, 64, 128\}$.
- IN: latent space size $l \in \{16, 32, 64, 128\}$, number of stacked INs $n \in [1, 4]$ and depth of MLP_E, $\text{MLP}_N \in [1, 4]$.
- Transformer Encoder: number of attention heads $h \in \{1, 2, 4, 8\}$, FeedForward layer space size $f \in \{512, 1024, 2048\}$, number of stacked Transformer Encoders $n \in [1, 4]$ and latent space size $l \in \{16, 32, 64, 128\}$.

In all the experiments, we consider a decoder MLP_{DEC} with two hidden layers and ReLU is the non-linear activation function used for MLP_{E}, MLP_{N} and MLP_{DEC}. Cross-Entropy and Mean Squared Error (MSE) are the loss functions used in classification and regression tasks, respectively. We train all the models 200 epochs using Adam optimizer with a learning rate of 0.001 and with batches of size 256.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2623–2631). New York, NY, USA: Association for Computing Machinery.

Arik, S. Ö., & Pfister, T. (2021). TabNet: Attentive Interpretable Tabular Learning. In *Proceedings of the AAAI conference on artificial intelligence, vol. 35, no. 8* (pp. 6679–6687).

Bai, J., Wang, J., Li, Z., Ding, D., Zhang, J., & Gao, J. (2021). ATJ-Net: Auto-table-join network for automatic learning on relational databases. In *Proceedings of the web conference 2021* (pp. 1540–1551). New York, NY, USA: Association for Computing Machinery, ISBN: 9781450383127.

Baldi, P., Sadowski, P., & Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, *5*(1), 1–9.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., et al. (2018). Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261.

Battaglia, P., Pascanu, R., Lai, M., Rezende, D., & Kavukcuoglu, K. (2016). Interaction networks for learning about objects, relations and physics. *Advances in Neural Information Processing Systems*, 4509–4517.

Becker, B., & Kohavi, R. (1996). Adult. http://dx.doi.org/10.24432/C5XW20, UCI Machine Learning Repository.

Blackard, J. (1998). Covertype. http://dx.doi.org/10.24432/C50K5N, UCI Machine Learning Repository.

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., & Kasneci, G. (2022). Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21.

Bossan, B., Feigl, J., & Kan, W. (2015). *Otto group product classification challenge*. Kaggle, URL: https://kaggle.com/competitions/otto-group-product-classification-challenge.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785–794). New York, NY, USA: Association for Computing Machinery.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., et al. (2016). Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems* (pp. 7–10). New York, NY, USA: Association for Computing Machinery.

Cvitkovic, M. (2020). Supervised learning on relational databases with graph neural networks. arXiv preprint arXiv:2002.02046.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International conference on learning representations*.

Du, K., Zhang, W., Zhou, R., Wang, Y., Zhao, X., Jin, J., et al. (2022). Learning enhanced representation for tabular data via neighborhood propagation. In *Advances in neural information processing systems: vol. 35*, (pp. 16373–16384). Curran Associates, Inc.

Dua, D., & Graff, C. (2017). *UCI machine learning repository*. University of California, Irvine, School of Information and Computer Sciences, URL: http://archive.ics.uci.edu/ml.

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR workshop on representation learning on graphs and manifolds*.

FICO (2019). Home equity line of credit (HELOC) dataset. URL: https://community.fico.com/s/explainable-machine-learning-challenge.

Frosst, N., & Hinton, G. (2017). Distilling a neural network into a soft decision tree. arXiv preprint arXiv:1711.09784.

Gorishniy, Y., Rubachev, I., Khrulkov, V., & Babenko, A. (2021). Revisiting deep learning models for tabular data. *34*, In *Advances in Neural Information Processing Systems* (pp. 18932–18943). Curran Associates, Inc.

Guo, X., Quan, Y., Zhao, H., Yao, Q., Li, Y., & Tu, W. (2021). TabGNN: Multiplex graph neural network for tabular data prediction. In *3rd workshop on deep learning practice for high-dimensional sparse data with KDD*.

Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). Deepfm: A factorization-machine based neural network for ctr prediction. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence, IJCAI-17* (pp. 1725–1731).

Hamilton, W. L. (2020). Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *14*(3), 1–159.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173–182). Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee.

Huang, X., Khetan, A., Cvitkovic, M., & Karnin, Z. (2020). TabTransformer: Tabular data modeling using contextual embeddings. arXiv preprint arXiv:2012.06678.

Joseph, M., & Raj, H. (2022). GATE: Gated additive tree ensemble for tabular classification and regression. arXiv preprint arXiv:2207.08548.

Katzir, L., Elidan, G., & El-Yaniv, R. (2021). Net-DNF: Effective deep modeling of tabular data. In *International conference on learning representations*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., et al. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, *30*, 3146–3154.

Ke, G., Xu, Z., Zhang, J., Bian, J., & Liu, T.-Y. (2019). DeepGBM: A deep learning framework distilled by GBDT for online prediction tasks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 384–394). New York, NY, USA: Association for Computing Machinery.

Ke, G., Zhang, J., Xu, Z., Bian, J., & Liu, T.-Y. (2019). TabNN: A universal neural network solution for tabular data. URL: https://openreview.net/forum?id=r1eJssCqY7.

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International conference on learning representations*.

Kotelnikov, A., Baranchuk, D., Rubachev, I., & Babenko, A. (2023). TabDDPM: Modelling tabular data with diffusion models. In *International conference on machine learning* (pp. 17564–17579). PMLR.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., et al. (2023). Learning skillful medium-range global weather forecasting. *Science*.

Langley, P., & Sage, S. (1994). Oblivious decision trees and abstract cases. In *Working notes of the AAAI-94 workshop on case-based reasoning* (pp. 113–117).

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 4768–4777). Red Hook, NY, USA: Curran Associates Inc.

Luo, H., Cheng, F., Yu, H., & Yi, Y. (2021). SDTR: Soft decision tree regressor for tabular data. *IEEE Access*, *9*, 55999–56011.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., et al. (2018). Ray: A distributed framework for emerging AI applications. In *Proceedings of the 13th USENIX conference on operating systems design and implementation* (pp. 561–577). USA: USENIX Association.

Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., et al. (2019). Deep learning recommendation model for personalization and recommendation systems. arXiv preprint arXiv:1906.00091.

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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. *vol. 32*, In *Advances in neural information processing systems*. Curran Associates, Inc.

Popov, S., Morozov, S., & Babenko, A. (2019). Neural oblivious decision ensembles for deep learning on tabular data. arXiv preprint arXiv:1909.06312.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, *31*.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). *Improving language understanding by generative pre-training*. OpenAI.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. W. (2020). Learning to simulate complex physics with graph networks. In *37th International conference on machine learning, vol. PartF168147-11* (pp. 8428–8437).

Somepalli, G., Schwarzschild, A., Goldblum, M., Bruss, C. B., & Goldstein, T. (2022). SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. In *NeurIPS 2022 first table representation workshop*.

Tay, Y., Dehghani, M., Bahri, D., & Metzler, D. (2022). Efficient transformers: A survey. *ACM Computing Surveys*, *55*(6).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. In *Advances in neural information processing systems*: *vol. 30*, Curran Associates, Inc.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. In *International conference on learning representations*.

Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., & Huerta, R. (2012). Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B (Chemical)*, [ISSN: 0925-4005] *166–167*, 320–329.

Wang, R., Shivanna, R., Cheng, D., Jain, S., Lin, D., Hong, L., et al. (2021). DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021* (pp. 1785–1797). New York, NY, USA: Association for Computing Machinery.