A Deep Learning Blueprint for Relational Databases

Lukáš Zahradník
Czech Technical University
LukasZahradnik96@gmail.com

Jan Neumann
Czech Technical University
janneumannprg@gmail.com

Gustav Šír
Czech Technical University
gustav.sir@cvut.cz

Abstract

We introduce a modular neural message-passing scheme that closely follows the formal model of relational databases, effectively enabling end-to-end deep learning directly from database storages. We experiment with several instantiations of the scheme, including notably the use of cross-attention modules to capture the referential constraints of the relational model. We address the issues of efficient learning data representation and loading, salient to the database setting, and compare against representative models from a number of related fields, demonstrating favorable initial results.

1 Introduction

While the approaches to mathematical modeling of complex systems, ranging from control theory to machine learning (ML), evolved in various independent ways, one aspect remained almost universal — the *data representation*. Irrespective of the used models, from decision trees to neural networks, virtually all ML libraries expect input samples in the form of fixed-size numeric tensors, most often just (feature) vectors. Assuming the data samples as independent points in n-dimensional spaces is extremely convenient, allowing to build directly upon the elegant foundations of linear algebra and multi-variate statistics [23]. However, actual real-world data is not stored in numeric vectors or tensors, but mostly in the interlinked structures of internet pages, knowledge graphs and, particularly, *relational databases*. Indeed, while there are numerous data storage formats, the traditional relational database management systems (RDBMs) arguably dominate the industry, from medicine and engineering to enterprise application domains [17].

In recent years, we have witnessed *deep learning* to quickly dominate all perceptual domains, from vision and speech to language, nevertheless, it remains very rare to encounter neural models on the classic *tabular data* with heterogeneous features, where standard statistical models, mainly various decision tree ensembles [13], still seem to lead the benchmarks [38]. Improving performance of the neural models, mostly the Transformer architecture [43], on tabular datasets gains increasing amounts of attention, sometimes quoted as the "last unconquered castle" for deep learning [24]. Nevertheless, extending from the tabular to the *relational* data model remains an even bigger challenge.

In this paper, we introduce a blueprint for such an extension, building on the (deep) tabular model design and insights from *relational learning* [15], concerned exactly with such relational generalizations of classic (tabular) statistical models.

Table Representation Learning Workshop at NeurIPS 2023

1.1 Related Work

While the body of work on using deep learning with relational databases themselves is extremely scarce, we discuss also the generally related areas that either use neural models on simpler data structures, or address relational structures with non-neural models.

Tabular models Tabular (neural) models [1] are concerned with transferring the deep learning strategies into the (classic) tabular data setting, currently still largely dominated by standard statistical models, such as gradient-boosted trees [13]. These commonly aim to amend the Transformer architecture [43] to better fit the complex, often heterogeneous and discrete, attribute structure of tabular data, and exploit the popular pre-training strategies in the self-supervised regime [10]. Some notable works include the TabTransformer [21], being one of the first attempts to tackle tabular data, the TUTA model [47], extending Transformers with concepts from the tree-based models, the popular TURL [9], exploiting unsupervised pre-training to learn contextualized data representations, or RPT [40], which uses tuple-to-tuple pre-training to target data preparation tasks. While these tabular models are sometimes referred to as "relational", it is only in a loose sense of the word, as they do not follow the actual relational (database) representation model.

Statistical relational learning Proper learning with actual relational representations has for decades [6] been the concern of the little-known field of Relational machine learning [8]. It builds heavily on the formalism of first-order logic (FOL) [14], in which the tabular representation and the corresponding models are effectively viewed as *propositional*, while the database representation, corresponding formally to a subset of FOL, requires *relational* generalization(s) of such models. Many such FOL-based methods have been proposed, mostly following the paradigm of Inductive Logic Programming (ILP) [32], later extended with probabilistic methods under the umbrella of Statistical Relational Learning (SRL) [15]. The most appropriate SRL works capable of learning from database representations then follow the paradigm of "lifting" [25], referring exactly to such generalization of classic statistical models into the relational setting. However, building on the FOL foundations, the SRL models typically do not scale well and, most importantly, do not offer the latent representation learning capabilities of deep learning.

Propositionalization From the SRL view, the tabular Transformers address the exact same representation expressiveness as the classic tree-based counterparts they aim to surpass. The tabular, also known as "attribute-value", data format, is an established ML representation perpetuating the whole field. While much of the real-world data structures, such as relational databases, do *not* fit into this representation, a natural urge arises to transform such structures into the expected format and proceed with the standard models. This practice, generally referred to as *propositionalization* [27], is the traditional method of choice that has dominated the industry [41, 42]. Propositionalization is essentially a data preprocessing routine where relational substructures get extracted and aggregated into standard statistical (tabular) attributes, corresponding to various select-join-aggregate (SQL) routines in the database setting. Building on decades of practice with such workflows, the resulting (statistical) models typically perform very well, however, their representation learning capabilities are principally limited, as the preprocessing (denormalization) step *necessarily* introduces an information loss.

Neuro-symbolic integration An interesting area on the intersection of both proper relational (logical) representations and deep learning principles is known as Neural-Symbolic Integration [19]. There is a (small) number of neuro-symbolic frameworks that operate with some (subset of) FOL representation, effectively covering the relational databases, while marrying the principles of neural networks through deep integration, such as Neural Theorem Provers [34], Logic Tensor Networks [37], or Lifted Relational Networks [39]. These methods are, in theory, capable of actual deep learning from relational databases, however, to the best of our knowledge, none of these methods scales to real-world database sizes due to the complexity associated with their FOL-based foundations, except for those that follow some form of the propositionalization scheme under the hood [12].

Deep relational models The closest related work is that of extending standard neural models towards relational representations. The most prominent models in this category are Graph Neural Networks [48] designed for end-to-end learning with graph-structured data. There are currently hundreds of the original GNN [35] model variants, some of which are close in spirit to our proposal,

particularly some of the hyper-graph [11] and multi-relational [36] extensions towards knowledge-graph applications [46], nevertheless, the graph-based view within this stream of research is generally not concerned with the salient features specific to relational databases, particularly the rich inner structure of the individual records. Apart from a recent vision draft [45], there are only very few works that address (some of) the database-specific aspects, particularly [7] followed by an (unsuccessful) pre-training procedure in [30], and the work of [2], which further incorporates automatic feature engineering and random architecture search to improve its performance.

The most salient contribution of our work is that, apart from proper treatment of the inter-relational structure, we also incorporate, in the spirit of the tabular models, the intra-relational (attribute) structure of the relations with the Transformer architecture, embedded end-to-end within the same general message-passing scheme, and introduce an overall more complete framework for deep learning with relational databases.

2 Background

2.1 Relational Databases

The principles of relational databases are formally based on the *relational model* [5], rooted in first-order (predicate) logic [14], providing a unified declarative specification for managing structured data, irrespective of the particular software implementation. This abstraction allows to define any database as a collection of n-ary relations defined over the domains of their respective attributes, managed by RDBM system to ensure consistency of the data with the integrity constraints of the logical database schema. The key concepts to be used in this paper are as follows.

Relation (Table): Formally, an n-ary relation $R_{/n}$ is a subset of the Cartesian product defined over the domains D_i of its n *attributes* A_i: $R \subseteq D_1 \times D_2 \times \cdots \times D_n$, where $D_i = \text{Dom}(A_i)$. Each relation R consists of a heading (signature) $R_{/n}$, formed by the set of its attributes, and a body, formed by the particular attribute values, which is commonly viewed as a *table* T_R of the relation R.

Attribute (Column): *Attributes* $\mathcal{A}_\mathcal{R} = \{A_1, A_2, A_3, \ldots, A_n\}$ define the terms of a relation R, corresponding to the *columns* of the respective table T_R. Each attribute is a pair of the attribute's name and a *type*, constraining the domain $\text{type}(D_i) \subseteq \text{dom}(A_i)$ of each attribute. An attribute *value* a_i is then a specific valid value from the respective type of the attribute A_i.

Tuple (Row): An $n-$*tuple* in a relation $R_{/n}$ is a tuple[1] of attribute values $t_i = (a_1, a_2, \ldots, v_n)$, where a_j represents the value of the attribute A_j in R. The relational can thus be defined extensionally by the *unordered* set of its tuples: $R = \{t_1, t_2, \ldots, t_m\}$, corresponding to the *rows* of the table T_R.

Integrity constraints: Besides the domain constraints $\text{Dom}(A_i)$, the most important integrity constraints are the primary and foreign *keys*. A *primary* key PK of a relation R is a minimal subset of its attributes $R[PK] \subseteq \mathcal{A}_\mathcal{R}$ that uniquely identifies each tuple: $\forall t_1, t_2 \in R, \quad (t_1[PK] = t_2[PK]) \Rightarrow (t_1 = t_2)$. A *foreign* key FK_{R_2} in relation R_1 then refers to the primary key PK of another relation R_2: $\forall t \in R_1, \quad t[FK] \in \{t'[PK] \mid t' \in R_2\}$. This constitutes the inter-relations in the database, with the RDBMs managing the referential integrity of $T_{R_1}[FK] \subseteq T_{R_2}[PK]$.

2.2 Deep Learning

Deep learning [16] is a popular paradigm characterized by the use of *gradient descent* to optimize parameters of nested functions, commonly viewed through their parameterized computation graphs, referred to as *neural networks*. The main conceptual idea lies in learning *latent representations* of the data, corresponding to the inner layers of the networks, generally constrained to the form of fixed-size numeric tensors, which restricts from directly applying deep learning to relational databases. While passing beyond that limitation, we will generalize upon concepts known from two neural architectures that address two forms or related (simpler) structured representations of *sequences* and *graphs*.

[1] the ordering is instantiated through the naming of the attributes

Transformers The Transformer [43] is a popular *sequence-to-sequence* model, relying primarily on the "*attention*" mechanism for interrelating the sequence tokens $x_1, \ldots, x_n$. Each input token x_i here is *embedded* into a continuous vector representation: $E(x_i) \in \mathbb{R}^d$, and combined with a "positional encoding" capturing its positional role: $E'(x_i) = E(x_i) + \text{PosEnc}(x_i)$. The attention mechanism then inter-relates all pairs of the input tokens to update their values as $X' = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) V$, where Q, K, and V are the "query", "key", and "value" matrix projections (roles) of the input embeddings $E'(X)$. This efficient matrix computation can be further computed in parallel with separate Q, K, V projection matrices. The updated values X' then position-wise pass through two feed-forward layers: $MLP(x'_i) = W_2 \cdot \text{ReLU}(W_1 \cdot x'_i + b_1) + b_2$, followed by layer normalization to reduce internal covariate shift and residual connections for improved gradient propagation.

Graph Neural Networks Graph Neural Networks [48] (GNNs) are a general class of neural models aimed at *graph-structured* data using the concept of (differentiable) *message-passing*. Given an input graph $G = (\mathcal{V}, \mathcal{E})$, with a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$, let $h_v^{(l)} \in \mathbb{R}^{d^{(l)}}$ be the vector representation (embedding) of node v at layer l. The general concept of GNNs can then be defined through a (i) message $M^{(l)} : \mathbb{R}^{d^{(l)}} \times \mathbb{R}^{d^{(l)}} \to \mathbb{R}^{d_m^{(l)}}$, (ii) aggregation $A^{(l)} : \{\mathbb{R}^{d_m^{(l)}}\} \to \mathbb{R}^{d_m^{(l)}}$, and (iii) update $U^{(l)} : \mathbb{R}^{d^{(l)}} \times \mathbb{R}^{d_m^{(l)}} \to \mathbb{R}^{d^{(l+1)}}$ functions as:

1. Compute messages for each edge $(u, v) \in E$: $\quad m_{u \to v}^{(l)} = M^{(l)}(h_u^{(l)}, h_v^{(l)})$
2. Aggregate the messages for each $v \in V$: $\quad M_v^{(l)} = A^{(l)}\left(\{m_{u \to v}^{(l)} : (u, v) \in E\}\right)$
3. Update representation of each $v \in V$: $\quad h_v^{(l+1)} = U^{(l)}(h_v^{(l)}, M_v^{(l)})$

The particular choice of the message, aggregation, and update functions then varies across specific GNN models, which are commonly composed of a predefined number L of such layers, enabling the message-passing to propagate information across L-neighborhoods within the graph(s). Note that the attention module of the Transformer follows the same schema (assuming a fully-connected graph).

3 The Blueprint

In this section, we describe our proposed learning data representation and the respective neural message-passing architecture(s) suitable for end-to-end deep learning from relational databases.

3.1 Data and Learning Representations

Schema detection We aim at direct learning from raw database storage, with as little preprocessing as possible, while retaining the proper relational model semantics [5]. However, the current RDBMs do not necessarily preserve all the semantic information required for deep learning. For instance, for integer-type columns, the information on whether the data contained are of ordinal or nominal nature is missing and, similarly, string-type columns may either contain actual text, or represent nominal categories. [2] A distinction must also be made about attributes that form the key constraints, as to whether they convey actual information or serve merely the referential purpose. To resolve such issues while avoiding manual data preprocessing, we built a procedure that attempts to determine all such information from the database, based on a combination of simple heuristics on top of select statistics of the data. The output from the procedure is an annotated, deep learning-compatible database schema, which may optionally be (manually) amended if some additional domain knowledge for the given database is present. This is also useful for correcting databases with invalid designs, e.g. missing key specification(s). Once the schema is detected with all the attribute domains determined, we may proceed with their *embedding* in some standard fashion [1]. Particularly, we choose a simple embedding (no pre-training) of the detected categorical types while merely expanding the numeric types into the respective dimensionality.

Multi-relational hypergraphs In order to directly follow the inductive bias of the relational database model, we consider the learning representation of a database as a *two-level multi-relational*

[2] Using the SQL data type, e.g. 'text' or 'varchar', to differentiate categorical data is not sufficient.

hypergraph, where each relation R forms n-ary hyperedges corresponding to the n-tuples over its attributes $\{t_i = (a_1, a_2, \ldots, a_n)\}$, and each pair R_1, R_2 of such relations related through the foreign key constraints $R_1[FK_{R_2}] \subseteq R_2[PK]$ forms another set of hyperedges from the particular tuple pairs $\{(t^1 \cup t^2) \mid t^1 \in R_1, t^2 \in R_2,\ t^1[FK_{R_2}] = t^2[PK]\}$. We consider generally all the tuple attributes $(a_1^1, \ldots, a_n^1 \cup a_1^2, \ldots, a_m^2)$ to form the link, and not just their keys, as these may also be composite, possibly spanning the whole tuple as a corner case $R[PK] = \mathcal{A}_\mathcal{R}$, hence the forming of hyperedges instead of just edges here. Additionally, for each such foreign-key tuple pair (t^1, t^2), we also consider the "reverse" hyperedge (t^2, t^1) to be able to fully propagate learning representations throughout the database, irrespective of the (ad-hoc) ordering choices of the database designer. We then use the tuple pairs of (t^1, t^2) and (t^2, t^1) to build a bi-directional bi-partite hypergraph, connecting the tuples of the individual relations $R_{1/n}, R_{2/m}$, for each foreign key constraint in the database schema.

Data loading For deep learning, we need to establish what constitutes the learning samples (x_i, y_i) in the given relational setting. In this paper, we consider the standard (self-)supervised scenario where a single attribute A_j of a single target relation R forms the output labels y_i. Nevertheless, the input samples x_i can no longer be considered as i.i.d. tuples, since they take the form of the multi-relational hypergraphs defined above. There are generally two cases, either (a) the database contains separate relational samples where each row t_i of the target table T_R belongs to a single learning instance x_i, or (b) the database cannot be split into such separate components, with x_i possibly spanning the whole hypergraph structure. To extract the learning samples (x_i, y_i) in both cases, we follow a simple breadth-first-search (BFS) procedure, starting from each row t_i of the target table T_R, and expanding over all the tables related through the foreign key constraints, in both the referenced and the referencing directions, while checking for loops. Due to the possible interdependence between the samples, care must be taken to prevent information leakage about the labels, for which we mask out all target labels from the target column A_j of T_R when processing the samples.[3]

A salient feature of relational databases is that they can be very large, for which we allow to run the sample loading natively *in-database* through recursive SQL (self-)joins,[4] with which the data may be fetched into memory on demand in a lazy fashion. To further make sure that the resulting samples themselves fit into memory, particularly in the (b) case, we optionally bound the BFS with a depth limit on the number of joins between the tables. In inductive learning settings, this limit can be set to correspond to the perimeter of the relational receptive field of the subsequent neural message-passing, corresponding e.g. to the number of layers in GNN models (Sec. 2), without loss of information.

3.2 Neural Architecture Space

To natively facilitate deep learning on the two-level hypergraph structure of the relational model, we introduce a general two-level neural message-passing scheme composed of differentiable parameterized operations defined on the levels of (i) individual *attributes* (ii) and (sets of) related *tuples*. We further divide these operations w.r.t. their input-output characteristics into *transformations*: $X \overset{1:1}{\rightarrow} Y$, n-ary *combinations* $(X_1, X_2, \ldots, X_N) \overset{N:1}{\rightarrow} Y$, and generic permutation-invariant *aggregations* $\{X_1, X_2, \ldots, X_M\} \overset{M:1}{\rightarrow} Y$, where the X_i, Y may refer to either the attributes a or the tuples t. This can be seen as a generalization of the message-aggregate-update GNN scheme (Sec.2).

Building on the *atomicity* assumption of the database model [5], we consider the relations' *attributes* a as the minimal necessary processing unit. Every instantiation of the neural blueprint thus starts with embedding of the individual relation $R_{/n}$ attribute values $E(a_1), \ldots, E(a_n)$, resulting into a tuple of n vectors $t_i^{(0)} \in (\mathbb{R}^{d_1}, \ldots, \mathbb{R}^{d_n})$ per each original tuple t_i from $R_{/n}$. The attribute embedding dimensions $d_1, \ldots, d_n$ within and across the relations may generally differ, so as to accommodate the possibly varying information loads in the tables, but we set them to be the same here for simplicity. Each such tuple $t^{(0)}$ then undergoes either (i) an attribute *combination* $C_a : (a_1, a_2, \ldots, a_n) \overset{n:1}{\mapsto} t^{(1)}$ that merges the attribute embeddings into a joint tuple embedding $t^{(1)} \in \mathbb{R}^{d_{t^{(1)}}}$ or (ii) *transformation* $T_a : (a_1, \ldots, a_n) \overset{1:1}{\mapsto} (a'_1, \ldots, a'_n) = t^{(1)}$ that keeps the attribute embeddings separate as $t^{(1)} \in (\mathbb{R}^{d_{a^{(1)}}}, \ldots, \mathbb{R}^{d_{a^{(1)}}})$. In either case, the resulting tuple representation $t^{(1)}$ subsequently enters the

[3] Overlooking this precaution led to some inappropriate accuracy reports in some of the related works.

[4] Note that the purpose of the joins here is *not* to preprocess (denormalize) the data into a single (universal) relation (as in the propositionalization), but merely to explore the primary-foreign key sub-structures.

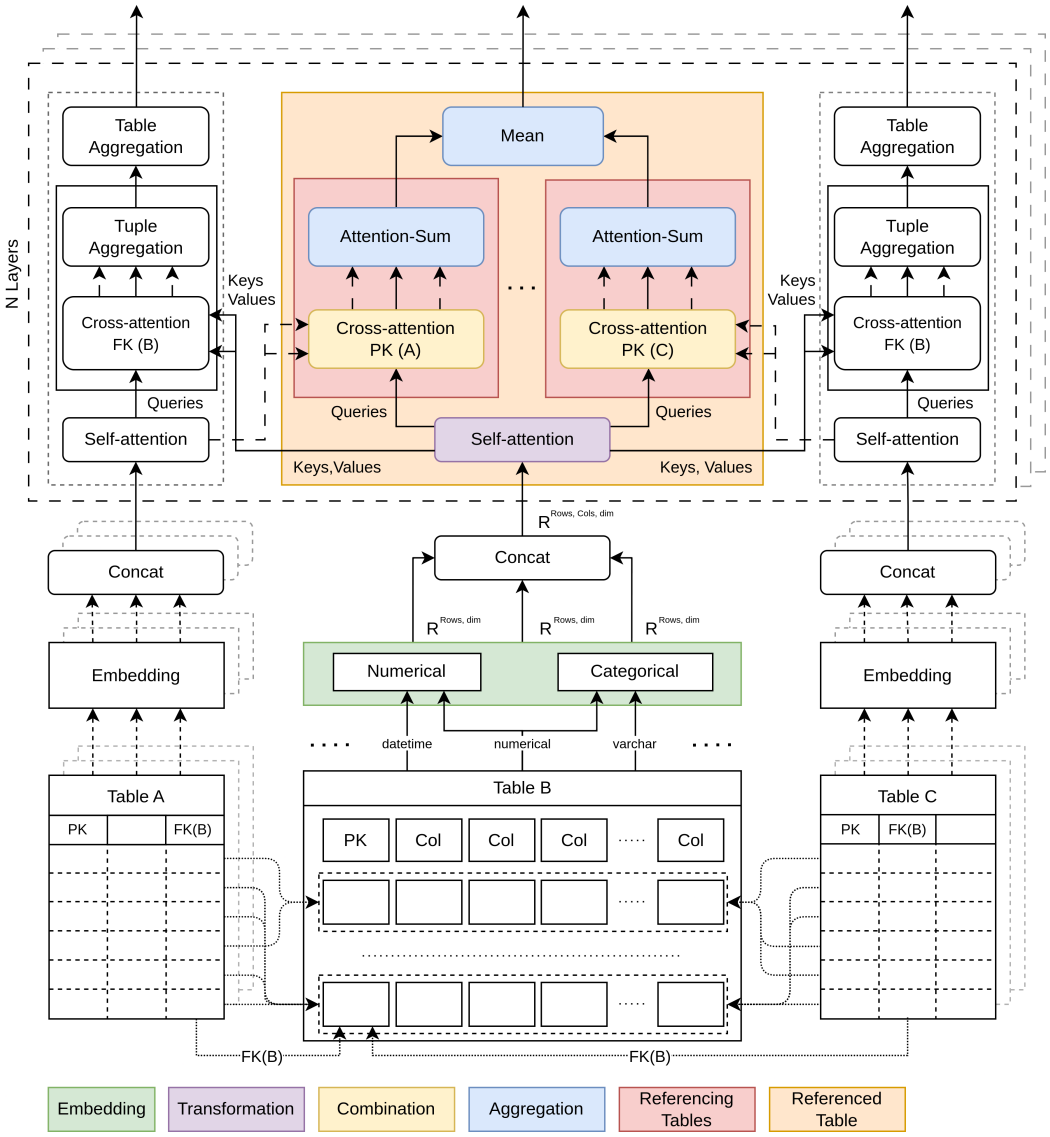

Figure 1: The architecture blueprint, instantiated with the self/cross-attention operations.

second level of neural computation where it gets combined with all the tuples related through the second type of hyperedges (Sec. 3.1). Particularly, each $t_i^{(1)} \in R_1^{(1)}$ undergoes a tuple *combination* $C_t : (t_i^{(1)}, t_j^{(1)}) \overset{2:1}{\mapsto} t_{i_{R_2}}^{(2)} \in \mathbb{R}^{d_{t^{(2)}}}$ with each $t_j^{(1)} \in R_2^{(1)}$ where $t_i[FK_{R_2}] = t_j[PK_{R_2}]$, resulting into a set of $\{t_{i_{R_2}}^{(2)}\}$ representations for each such pair of $t_i \in R_1$ and the related R_2. Each such set of the combined representations then undergoes a tuple *aggregation* $A_t : \{t_{i_{R_2}}^{(2)}\} \overset{m:1}{\rightarrow} t_{i_{R_2}}^{(3)}$ where $m = |\{t_{i_{R_2}}^{(2)}\}|$ to obtain one $t_{i_{R_2}}^{(3)}$ representation. Finally, we *aggregate* all such tuple representations $A_{t_R} : \{t_{i_{R_k}}^{(3)}\} \overset{l:1}{\rightarrow} t_i^{(4)} \in \mathbb{R}^{d_{t^{(4)}}}$ from all the $l = |\{R_k\}|$ linked relations back into a single final tuple representation $t_i^{(4)}$ for each $t_i \in R_{/n}$. The same computation is performed simultaneously for all relations $R_{/n}$ in the database, and the resulting representations may be used as input into subsequent layers of the same computation blueprint, in the classic spirit of deep learning.

Cross-attention table joins Technically, any differentiable parameterized operations that satisfy the corresponding input-output interface of the transformation, combination, and aggregation operators

can be used in their respective places within the blueprint, some of which are presented in our experiments (Sec. 4.1). Here we just highlight one interesting instantiation with the attention module we deem particularly suitable due to its (intra-)relational character. Particularly, one may use *self-attention* in place of any tuple *transformation*, in the standard spirit of the tabular Transformers [1] but, additionally, we may also use the *cross-attention* operation in place of the tuple *combination*. We hypothesize that the cross-attention module used in this place might be able to extract the necessary *latent* relational features, as exploited with the successful propositionalization methods (Sec. 1.1), but in a fully end-to-end fashion through gradient descent. Based on the notable expressiveness of Transformers [29], the select-join-aggregate operations normally used to construct such relational features should be well within the hypothesis space of the resulting architecture, in which we assume the query, key, and value roles of the input tokens to correspond to the foreign-key, primary-key, and column-value roles of the individual attributes, respectively. This particular instantiation of the blueprint is depicted in Figure 1. The idea here is that the self-attention (attribute transformation) firstly transforms the attributes w.r.t. each other within the tables, the cross-attention (tuple combination) then learns their contextual interactions with attributes from the referenced tuples, and the attention-sum (tuple aggregation) finally weights all their importances w.r.t. the referencing tuple.

4 Experiments

We test a few selected instantiations of the proposed blueprint against representative models from the distinct related work categories (Sec. 1.1) through (standard) supervised data classification tasks across a range of diverse relational database datasets.

Datasets While RDBMs are some of the most widespread data storages, publicly available relational database datasets are considerably scarce. Perhaps the most complete resource in this area is the CTU Prague Relational Learning Repository [31], maintaining dozens of database datasets from various domains. This repository also covers some of the older ILP[5] and SRL[6] dataset collections. We focus on datasets with more than two tables and more than two columns to emphasize the *relational* expressiveness of the proposed approach. The selected datasets are of diverse characteristics w.r.t. their sizes, schema structures, and application domains.

4.1 Models

Related work As a baseline we consider a simple *tabular* model operating solely on the target table, i.e. ignoring all the other tables related through the foreign keys. This naive strategy is particularly useful in revealing whether the given dataset task is indeed relational in nature or not. From statistical *relational learning*, we choose the state-of-the-art RDN-boost [33] which, following the lifting strategy (Sec. 1.1), can (very roughly) be seen as a relational version of the popular gradient-boosted trees [13]. As a *propositionalization* method we select the FastProp algorithm followed by XGBoost [3] – a battle-proof combination as implemented in [42], which leads a number of the relational dataset scoreboards.[7] To cover the *neuro-symbolic* area, we emulate the popular CILP++ method [12] similarly by connecting propositionalization with a feed-forward neural network. We were unable to put any of the few recent deep relational learning proposals (Sec. 2) into operation, but we attempt to emulate the GNN work of [7] as one of the blueprint ablations.

Blueprint instantiations As the space of all the possible neural architectures within the blueprint is very large, we present only a few selected instantiations. This means selecting some particular parameterized differentiable operations in place of the attribute a and tuple t transformations $T_{a/t}$, combinations $C_{a/t}$, and aggregations $A_{a/t}$. Generally, we consider a simple space of $T = \{\text{linear}, \text{MLP}, \text{self-attention}\}$, $C = \{\text{linear}, \text{MLP}, \text{cross-attention}\}$, and $A = \{\text{conv-sum}, \text{attention-sum}, \text{mean}\}$, where "conv-sum" is a classic GNN convolution [26] and "attention-sum" is a sum weighted by the self-attention coefficients [44], which is natively inside the attribute-level attention transformation, but we also use it for the generic (permutation-invariant) tuple aggregation (without PosEnc). For instance, the model from Fig. 1 can be characterized as

[5] `https://www.doc.ic.ac.uk/~shm/Datasets/`

[6] `https://starling.utdallas.edu/datasets/`

[7] `https://relational.fit.cvut.cz/statistics`

Table 1: Test accuracies of the selected blueprint instantiations (I_1, I_2, I_3) compared to the representative models from the related areas (Sec. 1.1) over a range of relational databases from [31].

category:	Tab.	Rel.	Prop.	NeSy	Ours		
datasets	**MLP**	**RDN-b**	**getML**	**CILP**	**I_1**	**I_2**	**I_3**
PTE	N/A	44.94%	**100.00%**	100.00%	**100.00%**	83.05%	**100.00%**
university	81.82%	81.82%	54.55%	81.82%	**100.00%**	**100.00%**	**100.00%**
NCAA	**100.00%**	47.50%	**100.00%**	78.75%	67.92%	71.69%	67.92%
cs	N/A	63.33%	96.67%	96.67%	**100.00%**	**100.00%**	**100.00%**
UTube	N/A	84.15%	98.93%	**99.39%**	98.16%	98.16%	98.16%
mutagen	87.50%	85.71%	82.86%	92.86%	**94.59%**	**94.59%**	**94.59%**
Dunur	N/A	23.17%	**97.56%**	**97.56%**	94.54%	94.54%	94.54%
MuskSmall	N/A	77.78%	74.07%	66.67%	**83.33%**	77.77%	50.00%
WebKP	N/A	82.51%	**83.04%**	65.40%	68.57%	51.99%	65.14%
DCG	N/A	72.57%	65.17%	61.06%	73.89%	65.92%	**79.20%**
Pima	N/A	32.17%	**77.11%**	75.65%	58.82%	73.20%	74.50%
CiteSeer	N/A	**66.16%**	47.41%	37.36%	50.15%	51.51%	37.76%
Carcinogen.	N/A	53.06%	62.07%	**65.31%**	64.61%	63.07%	60.00%
Toxicology	N/A	63.73%	57.02%	**72.55%**	61.76%	67.64%	61.76%
Chess	40.91%	34.09%	33.64%	48.86%	**50.84%**	**50.84%**	**50.84%**
Atheroscler.	26.72%	18.10%	22.41%	28.45%	**33.76%**	32.46%	31.16%

$A^{\text{mean}}_{t_R}(A^{\text{attention-sum}}_{t}(C^{\text{cross-attention}}_{t}(T^{\text{self-attention}}_{a})))$. For the experiments, we select 3 such representative instances as: $I_1 = A^{\text{mean}}_{t_R}(A^{\text{conv-sum}}_{t}(C^{\text{linear}}_{t}(T^{\text{linear}}_{a})))$, $I_2 = A^{\text{mean}}_{t_R}(A^{\text{attention-sum}}_{t}(C^{\text{linear}}_{t}(T^{\text{MLP}}_{a})))$, and $I_3 = A^{\text{mean}}_{t_R}(A^{\text{mean}}_{t}(C^{\text{cross-attention}}_{t}(T^{\text{self-attention}}_{a})))$.

Parameterization We set simple default parameters across all the methods. For MLP (linear) modules in all the models, we use a single hidden layer with half of the input dimensionality. The input number of relational features in the models based on propositionalization ranges about 200, depending on the depth of a custom BFS procedure which we implemented to improve the base feature extraction. The dimensionalities of the embeddings and the attention modules range from 10 to 64, depending on dataset size. We do not use any pre-trained embeddings nor share parameterization across different attributes or relations, which might be appropriate in many cases. We optimize all the gradient descent-based models with $lr = 0.0001$ in a default ADAM setting, and the boosting methods with the default $lr = 0.1$ and 100 base estimators. We evaluate all models on a random $80 : 20$ train-test split of the data w.r.t. the target table rows.

4.2 Results

Our results with the selected models (Sec. 4.1) are summarized in Tab. 1. We see that most of the datasets are not accessible to the tabular models (Tab.), as the target table does not contain any informative attributes, nevertheless, in the few cases where it does, simple tabular models (MLP) perform very well [24].[8] The RDN-boost is a sophisticated SRL (Rel.) method that does capture the relational dependencies for which it however needs to set up "modes" [33], which we implemented in a rather straightforward fashion, possibly explaining its generally weaker performance.[9] The getML system [42], on the other hand, performed very well out-of-box in its default setting (Fastprop+XGBoost), validating the strength of the propositionalization (Prop.) practice [28]. Similarly, the neuro-symbolic (NeSy) approach of CILP++ [12] emulated by combining propositionalization with MLP performed very strongly, too. Finally, our instantiations of the blueprint also showed some solid performance with close to no tuning across a range of operator configurations, some of which can be seen as a generalization of the popular graph-convolutional [26] (I_1) and graph-attention [44] (I_2) models from graphs to relational databases, close in spirit to the work of [7], and some of which are better viewed as extending the Transformer [43] (I_3) from sequences to relations.

[8] Their baseline performance can be expected to be improved with the more recent tabular Transformers, such as TabPFN [20], nevertheless with the same limited scope of applicability.

[9] A similar, very recent, work in this category that might be more appropriate in this context is JoinBoost [22].

5 Conclusions

We introduced a general blueprint for deep learning from relational databases, based on a custom message-passing scheme that respects the relational model of the common RDBMs storages. Our experiments with a (small) number of instantiations of the proposed concept demonstrate its viability, favorable properties, and competitive performance w.r.t. existing methods from various fields.

Future work Among open challenges is the ambiguous presentation of the data (Sec. 3.1), depending on choices of the database schema designer. As a corner case, even effectively non-relational data are sometimes stored in the database format (Sec. 4), and there is a general lack of large enough *relational* datasets to exploit the main benefits of deep representation learning. Nevertheless, incorporating self-supervised pre-training, in the spirit of the tabular models (Sec. 1.1), for domain transfer across different *databases* seems a promising avenue. However, despite the efficient data loading techniques introduced (Sec. 3.1), scaling to very large databases may be out of reach of the explicit message-passing, which might better be approximated with sampling [18] or indirectly emulated through gradient descent iterations [4] in transductive settings. Last but not least, incorporating the cross-attention to emulate the (latent) relational features opens interesting possibilities for interpretability and learning [29], such as initializing the blueprint parameters to emulate the popular propositionalization schemes.

Acknowledgments and Disclosure of Funding

This work has received funding from the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149.

References

[1] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics*, 2023.

[2] Jinze Bai, Jialin Wang, Zhao Li, Donghui Ding, Ji Zhang, and Jun Gao. Atj-net: Auto-table-join network for automatic learning on relational databases. In *Proceedings of the Web Conference 2021*, pages 1540–1551, 2021.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[4] Yihong Chen, Pushkar Mishra, Luca Franceschi, Pasquale Minervini, Pontus Lars Erik Saito Stenetorp, and Sebastian Riedel. Refactor gnns: Revisiting factorisation-based models from a message-passing perspective. *Advances in Neural Information Processing Systems*, 35:16138–16150, 2022.

[5] Edgar F Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[6] Andrew Cropper, Sebastijan Dumančić, and Stephen H Muggleton. Turning 30: New ideas in inductive logic programming. *arXiv preprint arXiv:2002.11002*, 2020.

[7] Milan Cvitkovic. Supervised learning on relational databases with graph neural networks. *arXiv preprint arXiv:2002.02046*, 2020.

[8] Luc De Raedt. *Logical and relational learning*. Springer Science & Business Media, 2008.

[9] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, 51(1):33–40, 2022.

[10] Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. Table pre-training: A survey on model architectures, pre-training objectives, and downstream tasks. *arXiv preprint arXiv:2201.09745*, 2022.

[11] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3558–3565, 2019.

[12] Manoel VM Franca, Gerson Zaverucha, and Artur Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Machine learning*, 94(1):81–104, 2014.

[13] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[14] Jean H Gallier. *Logic for computer science: foundations of automatic theorem proving*. Courier Dover Publications, 2015.

[15] Lise Getoor and Benjamin Taskar. *Introduction to Statistical Relational Learning*. 2007. ISBN 0262072882.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[17] Terry Halpin and Tony Morgan. *Information modeling and relational databases*. Morgan Kaufmann, 2010.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[19] Barbara Hammer and Pascal Hitzler. *Perspectives of neural-symbolic integration*, volume 77. Springer, 2007.

[20] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

[21] Xuanwei Hu, Weijing Tang, Cheng-Kang Hsieh, and Shuaiwen Shi. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

[22] Zezhou Huang, Rathijit Sen, Jiaxiang Liu, and Eugene Wu. Joinboost: Grow trees over normalized data using only sql. *arXiv preprint arXiv:2307.00422*, 2023.

[23] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. *An introduction to statistical learning*, volume 112. Springer, 2013.

[24] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Advances in neural information processing systems*, 34:23928–23941, 2021.

[25] A Kimmig, L Mihalkova, and L Getoor. Lifted graphical models: a survey. *Machine Learning*, 99(1):1–45, 2015.

[26] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[27] Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. *Relational data mining*, pages 262–291, 2001.

[28] Mark-A Krogel, Simon Rawles, Filip Železný, Peter A Flach, Nada Lavrač, and Stefan Wrobel. *Comparative evaluation of approaches to propositionalization*. Springer, 2003.

[29] David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. *arXiv preprint arXiv:2301.05062*, 2023.

[30] Shengchao Liu, David Vazquez, Jian Tang, and Pierre-Andre Noel. Flaky performances when pre-training on relational databases with a plan for future characterization efforts. In *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, 2022.

[31] Jan Motl and Oliver Schulte. The ctu prague relational learning repository. *arXiv preprint arXiv:1511.03086*, 2015.

[32] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19, 1994.

[33] Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, and Jude Shavlik. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 86:25–56, 2012.

[34] Tim Rocktäschel and Sebastian Riedel. Learning knowledge base inference with neural theorem provers. *Proceedings of the 5th Workshop on Automated Knowledge Base Construction (AKBC)*, pages 45–50, 2016.

[35] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[36] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[37] Artur Serafini, Luciano, Garcez, Luciano Serafini, and Artur S. d'Avila Garcez. Learning and reasoning with logic tensor networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10037:334–348, 2016. ISSN 16113349.

[38] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[39] Gustav Šourek, Vojtěch Aschenbrenner, Filip Železný, Steven Schockaert, and Ondřej Kuželka. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100, 2018.

[40] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. Rpt: relational pre-trained transformer is almost all you need towards democratizing data preparation. *arXiv preprint arXiv:2012.02469*, 2020.

[41] The Alteryx. Featuretools. URL `https://www.featuretools.com`.

[42] The SQLNet Company GmbH. getml. URL `https://getml.com`.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Neural Information Processing Systems*, 2017.

[44] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[45] Liane Vogel, Benjamin Hilprecht, and Carsten Binnig. Towards foundation models for relational databases [vision paper]. *arXiv preprint arXiv:2305.15321*, 2023.

[46] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29 (12):2724–2743, 2017.

[47] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790, 2021.

[48] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.