Joint Learning of Representations for Web-tables, Entities and Types using Graph Convolutional Network

Aniket Pramanick
TCS Research
aniket.pramanick@tcs.com

Indrajit Bhattacharya
TCS Research
b.indrajit@tcs.com

Abstract

Existing approaches for table annotation with entities and types either capture the structure of table using graphical models, or learn embeddings of table entries without accounting for the complete syntactic structure. We propose TabGCN, which uses Graph Convolutional Networks to capture the complete structure of tables, knowledge graph and the training annotations, and jointly learns embeddings for table elements as well as the entities and types. To account for knowledge incompleteness, TabGCN's embeddings can be used to discover new entities and types. Using experiments on 5 benchmark datasets, we show that TabGCN significantly outperforms multiple state-of-the-art baselines for table annotation, while showing promising performance on downstream table-related applications.

1 Introduction

Table data abounds in webpages and organizational documents. Annotation of table entries, such as columns, cells and rows, using available background knowledge (e.g. Yago, DBPedia, Freebase, etc.), such as knowledge of entities and their types, helps in better understanding and semantic interpretation of such tabular data. The challenge, however, is that such web tables do not adhere to any standard format, schema or convention (Limaye et al., 2010). Additionally, knowledge graphs are typically incomplete - entities and types mentioned in tables may not always exist in the knowledge graph. Therefore, it becomes necessary to expand the knowledge graph with new entities (Zhang et al., 2020) and types for annotating tables.

Initial research on table annotation (Limaye et al., 2010; Takeoka et al., 2019; Bhagavatula et al., 2015) used probabilistic graphical models to capture the complete row-column structure of tables and also the knowledge graph for collective annotation. More recent approaches using embeddings (Gentile et al., 2017; Zhang and Balog, 2018; Zhang et al., 2019; Chen et al., 2019; Yin et al., 2020) only partly capture the syntactic structure of tables, and also ignore the structure of the knowledge graph. The problem of incompleteness of the knowledge representation (Zhang et al., 2020) is mostly not addressed.

In this work, we propose the TabGCN model that uses a Graph Convolutional Network (GCN) (Kipf and Welling, 2017) to unify the complete syntactic structure of tables (rows, columns and cells) and that of the knowledge graph (entities and types) via available annotations. The embeddings of the table elements as well as knowledge graph entities and types are trained jointly and end-to-end. While GCNs have been used for learning embeddings for many NLP tasks using the syntactic and semantic structure of natural language sentences (Marcheggiani and Titov, 2017; Vashishth et al., 2019), encoding tabular structure using GCNs has not been addressed before. The model and embeddings thus trained are used to annotate new tables with known entities and types, while discovering hitherto unseen entities and types. Additionally, we use the trained embeddings for tables and rows for downstream table-related tasks - identifying similar tables, and identifying the appropriate table for any row.

We demonstrate these capabilities of TabGCN using experiments on 5 benchmark web table datasets comparing against 5 existing models. We show that WebGCN significantly improves performance for entity and type annotation. For the other tasks, we show that the same embeddings show impressive performance. No existing model can perform all of these tasks.

Our contributions are as follows: **(a)** We propose a model called TabGCN based on the GCN architecture that captures the complete syntactic structure of tables as well as the knowledge representation, and learns embeddings of tables, rows, columns and cells, as well as entities and types jointly and in an end-to-end fashion. **(b)** TabGCN addresses incompleteness in the knowledge representation by discovering new entities and types. **(c)** TabGCN significantly outperforms 5 existing approaches in 5 different benchmark datasets for the task of table annotation. **(d)** The trained embeddings show impressive performance in downstream tasks such as identifying similar tables and assignment of rows to appropriate tables.

2 Related Work

Existing literature on table annotation considers two different types of tables. In general, web tables (Limaye et al., 2010; Takeoka et al., 2019; Bhagavatula et al., 2015) contain mentions of entities under every column, which need to be annotated. In contrast, relational tables (Gentile et al., 2017; Zhang and Balog, 2018; Zhang et al., 2019, 2020) refer to a single entity in each row, with a single core or anchor column and multiple attribute columns. Here, the entire row is annotated with a single entity. While both tasks are important, our focus is on the first category.

In terms of approaches, one category of work considers graphical models to capture table structure and performs joint inference for entity and type classification (Limaye et al., 2010; Takeoka et al., 2019; Bhagavatula et al., 2015). Limaye et al. (2010) uses a Markov Random Field that captures the structure of tables, and creates dependencies between entity and type annotations to jointly classify entities for cells and types for columns. The Markov Random Field (MRF) potentials capture domain knowledge about similarities between cells, between cells and entity lemmas and between entities using the type hierarchy. It cannot handle entities and types not used as labels during training. MeiMei (Takeoka et al., 2019) extends this MRF framework to handle numeric columns and focuses on multi-label classifiers for entities with multiple types. Additionally, it constructs embeddings of entities and types using knowledge graph structure to avoid expensive graph traversal for computing potential function features. However, it still requires manual construction of these features based on domain knowledge. Our GCN architecture is motivated by the structure of the graphical model in these papers. Both these models also jointly label pairs of columns with known relationships, which we do not address in our work.

The second category focuses on embeddings for tables (Gentile et al., 2017; Zhang and Balog, 2018; Zhang et al., 2019). All of these models transform table data to word sequences and then make use of neural language models. Zhang and Balog (2018) make use of RDF2vec for embedding tables, which internally uses neural language models after transforming graphs to sequences. ColNet (Chen et al., 2019) considers columns as cell sequences and uses a CNN to learn the representations of individual cells, which are used to predict entity types for columns. TaBERT (Yin et al., 2020) focuses on retrieving table rows for natural language utterances and learns a joint representation for utterances and cell sequences in table rows using BERT. In summary, these fail to capture the complete row-column structure of tables in the embeddings. In contrast, our GCN architecture captures the structure of all tables, the entities, the types and the training annotations.

For the problem of extending the knowledge graph from tables, Zhang et al. (2020) consider discovery of new entities, but only in the context of relational tables, with a single core column. Their approach make use of pre-trained neural embeddings (word2vec) and cosine similarity in addition to lexical similarity. Our focus is on web tables, where we need to discover new entities and types for all columns in a table. Our approach makes use of GCN embeddings trained using entity and type labels to detect new entities and types.

3 Problem: Table Annotation

In this section, we first define tables, the background knowledge of entities and types, annotation of tables with entities and types, and, finally, the problem of semi-supervised table annotation with incomplete knowledge.

Tables: We are given a set of tables $\mathcal{S}$. Fig.1 shows an example at the top. The k^{th} table $S^k \in \mathcal{S}$, consists of m^k rows and n^k columns of individual cells. The individual cell in the i^{th} row and j^{th} column of the k^{th} table is denoted as x^k_{ij}. We consider textual tables only, so that each x^k_{ij} takes a string as value. Also, we denote the i^{th} row as R^k_i and the j^{th} column as C^k_j.

Entities and Types: We assume background knowledge of entities and entity types (or simply,

[T12]	[T34]
Tamara [E123]	Rossiya [E789]
Rudolph [E456]	Russia [E789]

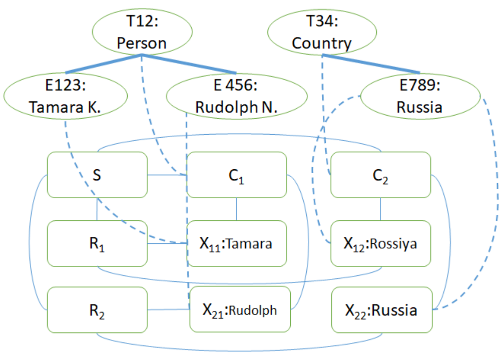

Figure 1: (a) Example table with 2 columns and 2 rows. The column annotations with types and the cell annotations with entities are shown within brackets. (b) GCN graph for example table with table nodes as boxes and entities and types nodes as ovals. Table edges are shown using fine solid lines, entity-type edges with thick solid lines and annotation edges using dashed lines. Lexical similarity edges are typically between cells in different tables and are not shown.

types). Let $\mathcal{T}$ denote the set of types, and $\mathcal{E}$ the set of entities. Each entity E is associated with a type $\mathcal{T}(E)$. For each entity E, there is also an entity description or lemma $\mathcal{L}(E)$. In Fig.1, the oval nodes shows example entities and types. Entity *E123* has associated type *T12:Person* and lemma *Tamara K.*.

Table Annotations: We assume that tables contain information about entities $\mathcal{E}$ and types $\mathcal{T}$. Specifically, each column of each table corresponds to a single type, each cell corresponds to a specific entity of that type. In our example table, cell x_{11} is annotated with entity *E123*, and column C_1 is annotated with type *T12:Person*. Let $\mathcal{T}(C_j^k)$ denote the type associated with column C_j^k, and $\mathcal{E}(x_{ij}^k)$ the entity associated with the cell x_{ij}^k. Let A_e be the set of all entity annotations of cells, and A_t that of all type annotations of columns.

Semi-supervised Table Annotation with Incomplete Knowledge: In the semi-supervised table annotation task, we are given the entire set of tables $\mathcal{S}$ but only a subset $A_e^o \subset A_e$ of the entity annotations, and a subset $A_t^o \subset A_t$ of the type annotations are observed. The task is to annotate the unannotated cells and columns of the tables, using the observed annotations as training data.

Let $\mathcal{T}^o$ denote the set of unique types seen in A_t^o, and $\mathcal{E}^o$ the set of unique entities seen in A_e^o. In the

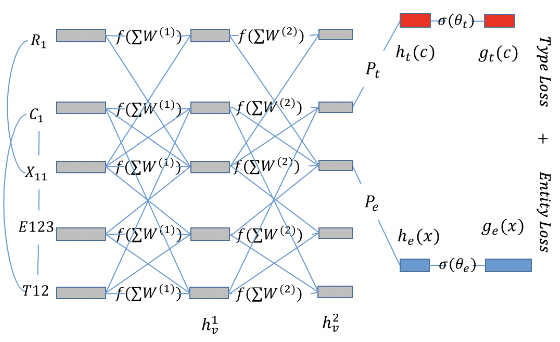

Figure 2: Model architecture showing GCN component with gray nodes, entity classification component with blue nodes and type classification components with red nodes.

incomplete knowledge setting, $\mathcal{T}^o \subset \mathcal{T}$, indicating that all the types are not seen in the training annotations. Similarly, all the entities are also not seen in training: $\mathcal{E}^o \subset \mathcal{E}$. Now, the task for the unannotated cells and columns is three-fold. The first is to decide whether these correspond to observed entities A_e^o and observed types A_t^o. We call this *novelty classification*. Next, the non-novel table columns need to be annotated with observed types $\mathcal{T}^o$, and the non-novel table cells with observed entities $\mathcal{E}^o$. We call these *type detection* and *entity detection* respectively. Finally, the columns corresponding to novel types need to be grouped according to distinct novel types, and the cells corresponding to novel entities need to be grouped according to distinct novel entities. We call these *type discovery* and *entity discovery* respectively.

4 Joint Table and Knowledge Embedding

We first present at a high level the network architecture of our model, which we call **TabGCN**. The core components TabGCN are (I) a Graph Convolutional Network (GCN), which captures the various syntactic relationships between table and knowledge elements, and then jointly learns embeddings for these via the convolution operation. The GCN embeddings of table elements contain information about both types and entities. These are fed into two parallel components: (II) the Type Classification component, and (III) the Entity Classification component. The Type Classification component first projects the GCN embedding of *table columns* to a *type space* using a *type-projection matrix*, and then uses a soft-max layer to classify this *type embedding* according to observed types. Similarly, the

Entity Classification component first projects the GCN embeddings of *table cells* to an *entity space* using an *entity-projection matrix*, and then uses a soft-max layer to classify this *entity embedding* according to observed entities. Fig.2 shows the high level architecture of our model for a subgraph of our example table graph in Fig.1. The parameters of all three components are trained jointly in an end-to-end fashion using training entity annotations for cells and type annotations for columns via back-propagation. We next describe these components in greater detail.

Graph Convolutional Network: Graph Convolutional Networks (GCN) (Kipf and Welling, 2017; Gilmer et al., 2017) extend the notion of convolution to graphs. Let $\mathcal{G} = (\mathcal{V}, R)$ be a directed graph where $\mathcal{V}$ is the set of n nodes and R the set of directed edges. An edge between nodes $u, v \in \mathcal{V}$ with label L_{uv} is denoted as $(u, v, L_{uv}) \in R$. The edge set includes an inverse edge for each edge and a self-loop for each node. An input feature matrix $\mathcal{X} \in \mathbb{R}^{m \times n}$ contains the input feature representation $x_u \in \mathbb{R}^m$ of each node $\forall u \in \mathcal{V}$ in its columns. Output embedding of a node v at k^{th} layer of GCN is given by

$$h_v^{(k+1)} = f\left(\sum_{u \in \mathcal{N}(v)} W_{L_{uv}}^{(k)} h_u^{(k)} + b_{L_{uv}}^{(k)}\right), \forall v \in \mathcal{V} \quad (1)$$

Here, $W_{L_{uv}}^{(k)}$ and $b_{L_{uv}}^{(k)}$ are label specific model parameters at k^{th} layer and $h_u^{(1)} = x_u$. For classification, a linear classification layer is added on top of final GCN layer. Function $f()$ is a non-linear activation for which we used ReLU.

GCN for Table and Knowledge Elements: Our GCN graph connects table parts, entities and types. We show the GCN graph for our example table in Fig.1. Its node set V consists of **table nodes** (boxes) and **knowledge nodes** (ovals). For each table $S^k \in \mathcal{S}$, the table nodes include one node for each cell x_{ij}^k, one node for each column C_j^k, one node for each row R_i^k, and one node for the table S_k itself. The knowledge nodes include one node for each *observed* type $T^o \in \mathcal{T}^o$ and one node for each *observed* entity $E^o \in \mathcal{E}^o$.

Recall that edges in a GCN serve to bring the embeddings of their end nodes closer, the extent being determined by their weight. With this intuition we create the edges R of different types reflecting the underlying semantics of tables and the annotations. These are **table edges** R_t, **knowledge edges** R_k, **annotation edges** R_a, and **lexical similarity edges** R_l.

Table edges capture the semantics of web tables, which do not have special anchor columns. These are of four categories: a *cell-column edge* between a cell node x_{ij}^k and its corresponding column node C_j^k; a *cell-row edge* between a cell node x_{ij}^k and its corresponding row node R_i^k; a *column-table edge* between a column node C_j^k and its corresponding table node S^k; and a *row-table edge* between a row node R_i^k and its corresponding table node S^k.

Knowledge edges connect each entity node E^o with its corresponding type node $\mathcal{T}(E^o)$.

Annotation edges are of two categories: an *entity annotation edge* for each entity annotation in A_e^o between a cell node x_{ij}^k and its labeled entity node $\mathcal{E}(x_{ij}^k)$; and a *type annotation edge* for each type annotation in A_t^o between a column node C_j^k and its labeled type node $\mathcal{T}(C_j^k)$.

Lexical similarity edges are added between pairs of cells in the same or different tables whose lexical similarity, computed using character-based Jaccard (Limaye et al., 2010), is above a threshold.

All edges are bi-directional. Each of the 8 edge categories above has its own parameters $(W_l^{(k)}, b_l^{(k)})$ for each layer in our GCN. Self loops are added for nodes associated with textual input, specifically, cells and entities with lemmas. For the input representation of such nodes, we use the pre-trained word embeddings for each of their constituent tokens, and take their mean. For this paper we used GloVe (Pennington et al., 2014).

Type Classification: The final (K^{th}) GCN layer generates an embedding $h_v^{(K)}$ for each node v. This contains information about both types and entities. For type classification of a column node c, we first get its *type embedding* $h_t(c)$ by projecting $h_c^{(K)}$ to a *type space* using a *type projection matrix* P_t: $h_t(c) = P_t h_c^{(K)}$. Then we get the probability distribution $g_t(c)$ over known types $\mathcal{T}^o$ for the type embedding $h_t(c)$ using a soft-max layer with weight matrix θ_t: $g_t(c) = \sigma(h_t(c); \theta_t)$. The type projection matrix P_t and the sigmoid weight matrix θ_t form the parameters for this component.

Entity Classification: We follow a similar approach for entity classification of a cell node x. We first project its GCN embedding $h_x^{(K)}$ to an *entity* *space* using an *entity projection matrix* P_e. The *entity embedding* $h_e(x)$ is passed through a soft-max layer with weight matrix θ_e to get the probability distribution $g_e(x)$ over known entities $\mathcal{E}^o$. The entity projection matrix P_e and the sigmoid weight matrix θ_e form the parameters for this component.

Joint Training: The parameters for all three components are trained end-to-end using available entity annotations for cells and type annotations for columns. Specifically, we consider the type predictions $g_t(c)$ for columns and minimize classification loss with the observed entity labels $\mathcal{T}(c)$. We similarly minimize classification loss between entity predictions $g_e(x)$ and observed entity labels $\mathcal{E}(x)$ for cells. We consider a weighted combination of the entity prediction loss and the type prediction loss. We have used cross-entropy as the classification loss function, and Adam (Kingma and Ba, 2015) for optimization.

5 Annotating Tables using Embeddings

Training the network in Sec.4 generates estimates of the model parameters as well as embeddings for all table and knowledge elements. In this section, we describe the use of these parameters and embeddings for the tasks defined in Sec.3. We use $h(v)$ for the GCN embedding of a node v instead of $h_v^{(K)}$ for brevity.

Novelty Classification: To decide whether an unannotated column c corresponds to any of the known types in $\mathcal{T}^o$ (novel type classification), we make use of their type embeddings. Column c corresponds to a new type if its type-space embedding $h_t(c) = P_t h(c)$ is 'far away' from the type space embedding $h_t(T) = P_t h(T)$ for all $T \in \mathcal{T}^o$. More specifically, we use $\delta(\max_{T\in\mathcal{T}^o} \cos(P_t h(c), P_t h(T)) \leq \epsilon_t)$, where ϵ_t is the *novel type threshold*, and $\delta()$ is the Kronecker delta function. A similar approach is followed for deciding if an unannotated cell x corresponds to any of the known entities in $\mathcal{E}^o$ (novel entity classification) by using the entity embeddings. Specifically, we use $\delta(\max_{E\in\mathcal{E}^o} \cos(P_e h(x), P_e h(E)) \leq \epsilon_e)$, where ϵ_e is the *novel entity threshold.*

Type and Entity Detection: Columns and cells determined to be non-novel need to be classified according to known types and entities respectively. This can be done using forward propagation in the trained network on the embeddings of the corresponding nodes. The type prediction $g_t(c)$ of a column c is obtained as $g_t(c) = \sigma(P_t h(c); \theta_t)$. Similarly, the entity prediction $g_e(x)$ of a cell x is obtained as $g_t(x) = \sigma(P_e h(x); \theta_e)$.

Type and Entity Discovery: On the other hand, columns and cells determined to be novel need to be grouped according to distinct new types and entities respectively. This is done by clustering their projections in the appropriate space. Specifically, for type discovery, we take the *type embeddings* $h_t(c)$ of all novel columns c and cluster these. Similarly, for entity discovery, we cluster the *entity embeddings* $h_e(x)$ of all novel cells x. The clustering algorithm needs to automatically determine the number of clusters in both cases. In this paper, we have used Silhouette clustering (Rousseeuw, 1987) as a representative non-parametric clustering algorithm. Other algorithms approaches such as Bayesian non-parametric techniques (Teh, 2010) may be used here.

Down-stream Inference for Rows and Tables: Training annotations are only provided for cells and columns. But embeddings are available for the rows and tables as well after training, and these can be used for different down-stream application tasks. Since these do not involve type or entity spaces, we directly use their GCN embeddings for these tasks. As examples, we define two such tasks here. **Table clustering** is the task of grouping together semantically related tables. For this, we cluster the table embeddings $h(S)$ of all tables $S \in \mathcal{S}$. For consistency, we again use Silhouette clustering. **Row to Table assignment** is the task of assigning a row to its most appropriate table. For this, we assign a row R with embedding $h(R)$ to the table S_k with the 'closest' embedding $h(S_k)$, or, more specifically, to $S^* = \arg\max_{S_k} \cos(h(R), h(S_k))$. If R is a row from a table provided during training, then its embedding is readily available. If it is a new row, then its embedding is created by convolving over the input embeddings of its constituent cells using the trained parameters W_l for cell-row edge.

6 Experiments

In this section, we first present experimental results for table annotation, and then for down-stream table-related tasks. We compare our proposed model **TabGCN** with appropriate state-of-the-art baselines. We have uploaded our source code as supplementary material for reproducibility.

Model	Wiki M		Web M		T2Dv2		Limaye		Wikipedia	
	Type	Entity	Type	Entity	Type	Entity	Type	Entity	Type	Entity
PGM	**0.55**	**0.70**	0.57	0.75	-	-	0.60	0.75	-	-
MeiMei	0.40	-	0.62	-	-	-	0.58	-	-	-
Tab2Vec	-	0.20	-	0.50	-	-	-	0.50	-	-
ColNet	0.20	-	0.47	-	0.59	-	0.47	-	0.60	-
TaBERT	0.20	-	0.49	-	0.59	-	0.47	-	0.59	-
TabGCN	0.33	0.24	**0.84**	**0.83**	**0.82**	-	**0.84**	**0.79**	**0.85**	-

Table 1: Entity and Type detection performance using micro averaged F1 for TabGCN and 5 baselines models on 5 benchmark datasets. Note that T2Dv2 and Wikipedia do not have entity annotations. Also, not all baselines can perform both entity and type detection.

Model	Wiki M		Web M		T2Dv2		Limaye		Wikipedia	
	Type	Entity	Type	Entity	Type	Entity	Type	Entity	Type	Entity
ColNet	0.76	-	0.64	-	0.77	-	0.63	-	0.60	-
TaBERT	0.76	-	0.62	-	0.76	-	0.61	-	0.60	-
TabGCN	**0.88**	**0.86**	**0.84**	**0.84**	**0.90**	-	**0.84**	**0.83**	**0.82**	-

Table 2: Type and Entity Discovery performance using normalized mutual information (NMI) for TabGCN, ColNet and TaBERT. Other baselines cannot address this task. T2DV2 and Wikipedia do not have entity annotations.

Data: We used 5 benchmark web table datasets. Their statistics are shown in Table.4. **Wiki Manual** (Limaye et al., 2010) is a small dataset of simple non-infobox tables from Wikipedia articles. **Web Manual** (Limaye et al., 2010) contains tables fetched by web-crawling using Wiki Manual tables as queries. These two tables are manually annotated with entities and types from YAGO. The tables and annotations are noisier than Wiki Manual. **Limaye** (Chen et al., 2019; Efthymiou et al., 2017) corrects many incorrect annotation in Wiki Manual using entities and types from DBPedia 2015. **T2Dv2** (Chen et al., 2019) contains tables from Common Web Crawl. **Wikipedia** is a publicly available subset of the data used by Efthymiou et al. (2017). This has HTML tables from Wikipedia 2013 with class attribute "wikitable". The last three datasets are annotated using DBPedia. But **T2Dv2** and **Wikipedia** contain *only* type annotations, and entity annotations are not available. For all datasets, we first set aside 30% of the unique entities and types, and effectively all their annotations as unseen during training. Of the remaining annotations, again 30% was removed during training.

Baselines: We compare TabGCN against three existing table annotation approaches. **PGM** (Limaye et al., 2010) uses a probabilistic graphical model and hand-crafted features. Since our entity-type set $\mathcal{T}$ is flat instead of being a hierarchical graph, we use logistic regression instead of structural SVM to estimate model parameters. **MeiMei** (Takeoka et al., 2019) also uses a Markov Random field but embeds the entities and types for fast computation of clique potentials. For both models, we ignore the relation labels for column-pairs and associated potential functions, since we do not address relation detection. **Table2Vec** (Zhang et al., 2019) learns various word2vec-based embeddings for tables for entity annotation. It does not address type annotation. It focuses on relational tables and associates a single entity with a row for its core column. We adapted this model for web tables which associate an entity for each cell. We use their *Table2VecE* version, which models a row as a sequences of all entities that appear within cells of that row. After training the word2vec model, instead of considering the embedding for the entire row as in *Table2VecE*, we create cell-specific embeddings from only the tokens for that cell. **ColNet** (Chen et al., 2019) models each column as a sequence of words from the cells under the column and uses a CNN to predict the column type. We also use an adaptation of TaBERT (Yin et al., 2020) which trains a joint language model for retrieval from tables given utterances. We adapt their approach to independently linearize each row and column of a table as a sequence of cells under that row and column, and get column and row embeddings using the mean of corresponding cell embeddings. For cells, we use pre-trained BERT embeddings.

Hyper-parameters: We used one-hot encodings as inputs for cells and entities with lemmas. We used ReLU as the nonlinear GCN activation, 2 GCN layers with 512 and 256 dimensions. As a result, the 8 GCN weight matrices were Vx512 in the first layer, where V is the vocabulary size,

Model	Wiki M		Web M		T2Dv2		Limaye		Wikipedia	
	Hits@1	MRR	Hits@1	MRR	Hits@1	MRR	Hits@1	MRR	Hits@1	MRR
Tab2Vec	0.11	0.16	0.13	0.21	0.32	0.39	0.09	0.18	0.29	0.35
TaBERT	0.13	0.18	0.51	0.58	0.32	0.40	0.46	0.57	0.38	0.46
TabGCN	**0.18**	**0.21**	**0.71**	**0.77**	**0.67**	**0.70**	**0.71**	**0.77**	**0.71**	**0.74**

Table 3: Row assignment performance using Hits@1 and MRR for TabGCN, Tab2Vec and TaBERT on all datasets.

Dataset	T	R	C	E	$\mathcal{T}$	E^*	$\mathcal{T}^*$
Wiki Man.	39	36.3	4.2	1026	49	308	15
Web Man.	403	34.4	3.7	883	48	265	14
Limaye	294	27.7	3	504	21	151	6
T2Dv2	345	44.8	4.8	-	27	-	8
Wikipedia	572	24.4	5	-	29	-	9

Table 4: Dataset Statistics: For each dataset, T indictates the number of tables, R the average number of rows per table, C the average number of columns per table, E the number of unique entities and $\mathcal{T}$ the number of unique types used for annotation, E^* the number of new unique entities and $\mathcal{T}^*$ the number of new unique types *not seen* in training annotations.

and 512x256 in the second layer. The entity and type space embeddings had dimension 256, so that the entity and type projection matrices were both 256x256. For training, we used learning rate 0.001, dropout 0.5, 1000 epochs. The weights for combining the type and entity losses was $1 : 2$ for both datasets, optimized manually. All experiments were performed on a Dual-core intel Core i5 Processor with 8GB RAM. The average training time per epoch ranges from 1.3 secs for Wiki Manual to 35.4 secs for T2Dv2.

Detection Results: We first present results for entity and type detection, addressed by most of the baselines. For this task, all models predict entity labels for all cells and type labels for all columns that are unannotated. However, evaluation is *only* for those cells and columns whose true entity and type labels are contained in the observed entities $\mathcal{E}^o$ and observed types $\mathcal{T}^o$ respectively.

For evaluation, as in earlier table annotation papers (Limaye et al., 2010; Chen et al., 2019), we use micro-averaged F1, which takes the weighted average of the F1 scores over the entities or types. This takes class imbalance into account and is therefore more appropriate than accuracy. We note that MeiMei addresses the multi-label setting with multiple possible type labels, and therefore uses ranking evaluation measures (e.g. NDCG). This is not meaningful in our single-labeled setting.

Tab. 1 shows detailed results for all models across datasets. We can see that **TabGCN** significantly outperforms all baselines on all datasets for both detection tasks. The only exception is for Wiki Manual. The graphical model based approaches with handcrafted potential functions outperform the representation learning approaches, possibly on account of the smallness of the dataset. Among the embedding based approaches, TabGCN performs the best.

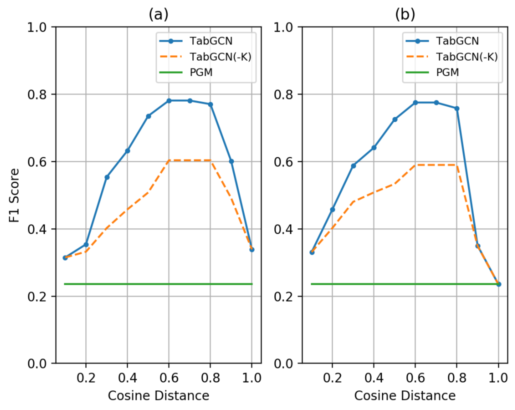

Figure 3: Novelty classification Performance on (a) types and (b) entities for TabGCN, its ablation TabGCN(-K) and PGM for Web Manual, showing F1 on the y-axis and decision threshold on x-axis. PGM does not use a decision threshold. Results on other datasets are very similar. Other baselines cannot address this task.

Novelty Classification Results: In our second experiment, we consider novelty classification. This is an unsupervised task, where a model makes a binary decision for each unannotated column (novel type classification) and for each unannotated cell (novel entity classification). Since the decision depends on the thresholds for type (ϵ_t) and entity (ϵ_e), we plot F1 score on the y-axis against the corresponding threshold on the x-axis. Of the baselines, only PGM can address this task, but outputting a NONE label for the type or entity. Fig.3 shows novelty classification performance for Web Manual. Results for Wiki Manual and Limaye are very similar. **TabGCN** reaches F1 around 0.8 for both tasks for appropriate thresholds. Across thresholds, TabGCN significantly out-

Models	Wiki M		Web M		Limaye	
	Type	Ent.	Type	Ent.	Type	Ent.
-K	0.29	0.23	**0.84**	0.82	0.83	0.78
-E	0.22	-	0.82	-	0.81	-
-T	-	0.22	-	0.63	-	0.63
-L	0.33	0.23	0.80	0.75	0.81	0.74
TabGCN	**0.33**	**0.24**	**0.84**	**0.83**	**0.84**	**0.79**

Table 5: Ablation study for entity and type detection showing micro. F1. -K removes knowledge (entity and type) nodes, -E removes entity nodes, -T removes type nodes and -L removes lexical similarity edges.

performs PGM.

Type and Entity Discovery Results: The final annotation tasks are type discovery, where all unannotated columns that do not correspond to known types in $\mathcal{T}^o$ need to be clustered into distinct new types, and entity discovery, where all unannotated cells that do not correspond to known entities in $\mathcal{E}^0$ need to be clustered into distinct new entities. We used Normalized Mutual Information (NMI) $NMI = \frac{2I(C,Y)}{H(C)+H(Y)}$ between the assigned cluster labels (Y) and the true entity or type labels (C), where $I(,)$ denotes mutual information and $H()$ denotes entropy. In Tab. 2, we see that **TabGCN** performs consistently above 80% for entity and type discovery across datasets, significantly outperforming ColNet and TaBERT.

Ablation Study: Next, we analyze the performance of **TabGCN**, using multiple ablations of the full model. **-K** leaves out the knowledge nodes and their incident edges from the GCN graph during training. **-E** focuses only on types by removing all entity nodes and entity-related edges (type-entity and cell-entity annotation) from the GCN graph. It is trained only using type loss. Note that it cannot perform tasks associated with entities, specifically entity detection, novel entity classification and novel entity discovery. Similarly, **-T** focuses only on entities by removing all type nodes and type-related edges (type-entity and column-type annotation) from the GCN graph. It is trained only using entity loss, and cannot perform tasks associated with types. Finally, **-L** removes the lexical similarity edges from the GCN graph. The results are recorded in Table. 5. We can see that all the components of the architecture contribute to performance improvements. The improvement is statistically significant (using the Wilson Interval with $\alpha = 0.05$) for all ablations other than **-K**. While **-K** performs comparably here, its performance drops significantly for novelty classification, as can be seen in Fig.3.

Model	T2Dv2	Limaye
Tab2Vec	0.66	0.49
TaBERT	0.76	0.51
TabGCN	**0.89**	**0.81**

Table 6: Table clustering performance using NMI for TabGCN, Tab2Vec and TaBERT on T2Dv2 and Limaye. Other baselines cannot address this task. T2Dv2 has 40 table categories and Limaye 15. Other datasets do not have table categories for evaluation.

Downstream Table Related Tasks: Finally, we include some results for the table and row related inference tasks defined at the end of Sec.5. This is to demonstrate how the learnt embeddings can benefit potential down-stream tasks. Note that TabGCN directly outputs table embeddings. Of the baselines, Tab2Vec and TaBERT output row embeddings. For these models, we create table embeddings by averaging the corresponding row embeddings.

In Tab. 6, we record performance for table clustering. **TabGCN** again significantly outperforms both baselines for both datasets.

We finally consider row-to-table assignment. In this task, one randomly selected row is removed from every table during training. These rows then need to be classified according to their parent table. Since the models output a ranking of tables for each row, we evaluate using two ranking related measures. *Hits@1* measures the fraction of rows with the correct table at rank 1. *Mean Reciprocal rank* (MRR) is the mean of the reciprocal rank of the correct table over all rows, and its perfect value is 1.0. In Tab. 3, we again see that **TabGCN** performs the best across datasets.

In summary, we have demonstrated the usefulness of learning embeddings of table elements and knowledge elements jointly using both entity and type losses in an end-to-end fashion for type and entity annotation on 5 benchmark datasets. In addition, we have demonstrated how the learned embeddings can be useful for downstream table-related tasks. In all cases, **TabGCN** has significantly outperformed multiple state-of-the-art baselines using probabilistic graphical models as well as other neural approaches.

7 Conclusion

We have proposed a model for that jointly learns representations of tables, rows, columns and cell, as well as entities and types by capturing the complete syntactic structure of all tables, the relevant entities and types and the available annotations using the Graph Convolutional Network. As a result, TabGCN unifies the benefits of probabilistic graphical model based approaches and embedding based approaches for table annotation. Using these embeddings, TabGCN significantly outperforms existing approaches for table annotation, as well as entity and type discovery.

References

Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. Tabel: Entity linking in web tables. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, volume 9366 of *Lecture Notes in Computer Science*, pages 425–441. Springer.

Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. 2019. Colnet: Embedding the semantics of web tables for column type prediction. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 29–36. AAAI Press.

Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. 2017. Matching web tables with knowledge base entities: From entity lookups to entity embeddings. In *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, pages 260–277. Springer.

Anna Lisa Gentile, Petar Ristoski, Steffen Eckel, Dominique Ritze, and Heiko Paulheim. 2017. Entity matching on web tables: a table embeddings approach for blocking. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, pages 510–513. OpenProceedings.org.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Thomas N. Kipf and Max Welling. 2017. 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.

Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, 3(1-2):1338–1347.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1506–1515. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL.

Peter Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65. *Journal of Computational and Applied Mathematics*, 20:53–65.

Kunihiro Takeoka, Masafumi Oyamada, Shinji Nakadai, and Takeshi Okadome. 2019. Meimei: An efficient probabilistic approach for semantically annotating tables. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 281–288. AAAI Press.

Yee Whye Teh. 2010. Dirichlet processes. In *Encyclopedia of Machine Learning*. Springer.

Shikhar Vashishth, Manik Bhandari, Prateek Yadav, Piyush Rai, Chiranjib Bhattacharyya, and Partha Talukdar. 2019. Incorporating syntactic and semantic information in word embeddings using graph convolutional networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3308–3318, Florence, Italy. Association for Computational Linguistics.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8413–8426. Association for Computational Linguistics.

Li Zhang, Shuo Zhang, and Krisztian Balog. 2019. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 1029–1032. ACM.

Shuo Zhang and Krisztian Balog. 2018. Ad hoc table retrieval using semantic similarity. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1553–1562. ACM.

Shuo Zhang, Edgar Meij, Krisztian Balog, and Ridho Reinanda. 2020. Novel entity discovery from web tables. In *Proceedings of The Web Conference 2020*, WWW '20, page 1298–1308, New York, NY, USA. Association for Computing Machinery.