TURL: Table Understanding through Representation Learning

Xiang Deng*
The Ohio State University
Columbus, OH
deng.595@osu.edu

Huan Sun*
The Ohio State University
Columbus, OH
sun.397@osu.edu

Alyssa Lees
Google Research
New York, NY
alyssalees@google.com

You Wu
Google Research
New York, NY
wuyou@google.com

Cong Yu
Google Research
New York, NY
congyu@google.com

ABSTRACT

Relational tables on the Web store a vast amount of knowledge. Owing to the wealth of such tables, there has been tremendous progress on a variety of tasks in the area of table understanding. However, existing work generally relies on heavily-engineered task-specific features and model architectures. In this paper, we present TURL, a novel framework that introduces the pre-training/fine-tuning paradigm to relational Web tables. During pre-training, our framework learns deep contextualized representations on relational tables in a self-supervised manner. Its universal model design with pre-trained representations can be applied to a wide range of tasks with minimal task-specific fine-tuning.

Specifically, we propose a structure-aware Transformer encoder to model the row-column structure, and present a new Masked Entity Recovery (MER) objective for pre-training to capture relational knowledge. We compiled a benchmark consisting of 6 different tasks for table understanding and used it to systematically evaluate TURL. We show that TURL generalizes well to all tasks and substantially outperforms existing methods in almost all instances.

1. INTRODUCTION

Relational tables are in abundance on the Web and store a large amount of knowledge. Each relational Web table has its own "schema" of labeled and typed columns and essentially forms a small structured database [3]. Over the past decade, various large-scale collections of such tables have been aggregated [3, 4, 2, 21]. Owing to the wealth and utility of these tables, various tasks such as table interpretation [2, 35, 26, 36, 14] and table augmentation [8, 33, 1, 34, 3] have made tremendous progress in the past few years.

However, previous work [33, 34, 2] often relies on heavily-engineered task-specific methods such as using simple statistical/language features or straightforward string matching. These techniques suffer from several disadvantages. First, simple features only capture shallow patterns and often fail to handle the flexible schema and varied expressions in relational tables. Second, task-specific features and model architectures require a lot of effort to design and do not generalize well across tasks.

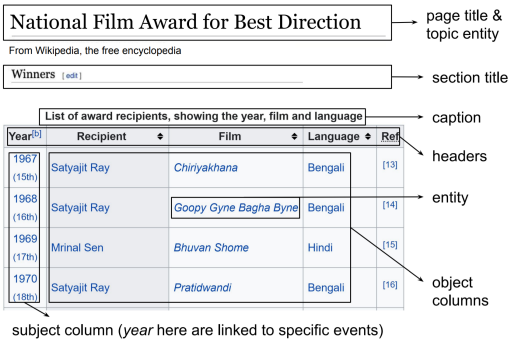


Figure 1: An example of a relational table from Wikipedia.

On the other hand, the representation learning model of [9], which uses Word2Vec [24] to learn general-purpose embedding vectors for words and entities in tables, achieved promising results on some table based tasks. However, [9] cannot generate *contextualized* representations, i.e., it does not consider varied use of words/entities in different contexts and only produces a single fixed embedding vector for word/entity. In addition, *shallow* neural models like Word2Vec have relatively limited learning capabilities, which hinder the capture of complex semantic knowledge contained in relational tables.

We propose TURL, a novel framework for learning deep contextualized representations on relational tables via self-supervised pre-training and task-specific fine-tuning. Recently, such pre-training and fine-tuning paradigms have achieved remarkable success on unstructured text data [13]. In contrast, little effort has been extended to study them on relational tables. Our work fills this gap.

There are two main challenges in the development of TURL: (1) *Relational table encoding.* Existing neural network encoders are designed for linearized sequence input and are a good fit with unstructured texts. However, data in relational tables is organized in a semi-structured format. Moreover, a relational table contains multiple components including table caption, headers and cell values. The challenge is to develop a means of modeling the row-and-column structure

*Corresponding authors.

DOI: 10.14778/3430915.3430921.
.

as well as integrating the information from different components of the table. (2) *Factual knowledge modeling.* Pre-trained language models such as BERT [13] focus on modeling the syntactic and semantic characteristics of word use in natural sentences. However, relational tables contain a vast amount of factual knowledge about entities, which cannot be captured by existing language models directly. Effectively modelling such knowledge in TURL is a second challenge.

To address the first challenge, we encode information from different table components into separate input embeddings and fuse them together. We then employ a *structure-aware* Transformer [28] encoder with masked self-attention. We explicitly model the row-and-column structure by restraining each element to only aggregate information from other structurally related elements. To achieve this, we build a visibility matrix based on the table structure and use it as an additional mask for the self-attention layer [28]. For the second challenge, we learn embeddings for each entity during pre-training, and model the relation between entities with the assistance of the visibility matrix. We then propose a Masked Entity Recovery (MER) pre-training objective: We randomly mask out entities in a table and predict the masked entities based on other entities and the table context (e.g., caption/header). *This encourages the model to learn factual knowledge from the tables and encode it into entity embeddings.* In addition, for a certain percentage of masked entities, we utilize their entity mention as additional information to base the recovery on. *This helps our model build connections between words and entities.*

We pre-train our model on around 570K relational tables from Wikipedia and fine-tune it for specific downstream tasks. A distinguishing feature of TURL is its universal architecture across different tasks - only minimal modification is needed to cope with each downstream task. To facilitate research in this direction, we compiled a benchmark that consists of 6 table understanding tasks, including entity linking, column type annotation, relation extraction, row population, cell filling and schema augmentation. Experimental results show that TURL substantially outperforms existing task-specific and shallow representation learning methods [7, 14, 26, 17, 33, 9].

This paper is a shortened version of our paper with the same title that appeared in VLDB 2021 [11]. More technical details and experimental results can be found in [11], as well as in the extended version published on Arxiv [12]. Our source code, benchmark, as well as pre-trained models are available online to facilitate future research.[1]

2. PRELIMINARY

In this work, we focus on relational Web tables and are most interested in the factual knowledge about entities. Each table T is associated with the following: (1) Table caption C, which is a short text description summarizing what the table is about. (2) Table headers H, which define the table schema; (3) Topic entity e_t, which describes what the table is about and is usually extracted from the table caption or page title; (4) Table cells E containing entities. Each entity cell $e \in E$ contains a specific object with a unique identifier. For each cell, we define the entity as $e = (e^{\mathrm{e}}, e^{\mathrm{m}})$, where e^{e} is the specific entity linked to the cell and e^{m} is the entity mention (i.e., the text string).

[1] https://github.com/sunlab-osu/TURL

(C, H, e_t) is also known as *table metadata* and E is the actual *table content*. We study self-supervised representation learning on relational Web tables, defined as follows.

Definition. Given a relational Web table corpus, we aim to learn in a self-supervised manner a task-agnostic contextualized vector representation for each token in all table captions C's and headers H's and for each entity (i.e., all entity cells E's and topic entities e_t's).

3. RELATED WORK

Representation Learning. The pre-training/fine-tuning paradigm has drawn tremendous attention in recent years. Extensive effort has been devoted to the development of self-supervised representation learning methods for both unstructured text [24, 13] and structured knowledge bases (KB) [27], which in turn can be utilized for a wide variety of downstream tasks via fine-tuning.

Despite the success of representation learning on text and KB, few works have thoroughly explored representation learning on tables. Pre-trained language models are directly adopted in [22] for entity matching. Two recent papers from the NLP community [16, 31] study pre-training on Web tables to assist in semantic parsing or question answering tasks on tables. In this work, we introduce TURL, a new methodology for learning deep contextualized representations for relational Web tables that preserve both semantic and knowledge information. In addition, we conduct comprehensive experiments on a much wider range of table-related tasks.

Table Understanding. In general, there are two types of table understanding tasks: table interpretation and table augmentation. Table interpretation aims to uncover the semantic attributes of the data contained in relational tables, and transform this information into machine understandable knowledge. This task is usually accomplished with help from existing KBs. In turn, the extracted knowledge can be used for KB construction and population. There are three main tasks for table interpretation: entity linking [2, 26, 14], column type annotation [6, 17] and relation extraction [26, 36]. Table augmentation seeks to offer intelligent assistance to the user when composing tables by expanding a seed query table with additional data. Specifically, for relational tables this can be divided into three sub-tasks: row population [8, 33], cell filling [1, 34] and schema augmentation [3, 33].

Several benchmarks have been proposed for both table interpretation [21, 23, 14, 19] and table augmentation [33, 34] over the last decade. Although these benchmarks have been used in various recent studies, they still suffer from a few shortcomings: (1) They are typically small sets of sampled tables with limited annotations. (2) SemTab 2019 [19] contains a large number of instances; however, most of them are automatically generated and lack metadata/context of the Web tables. In this work, we compile a larger benchmark covering both table interpretation and table augmentation tasks. We also use some of these existing datasets for more comprehensive evaluation. By leveraging large-scale relational tables on Wikipedia and a curated KB, we ensure both the size and quality of our dataset.

4. METHODOLOGY

In this section, we introduce our TURL framework for self-supervised representation learning on relational tables. Figure 3 presents an overview of TURL which consists of

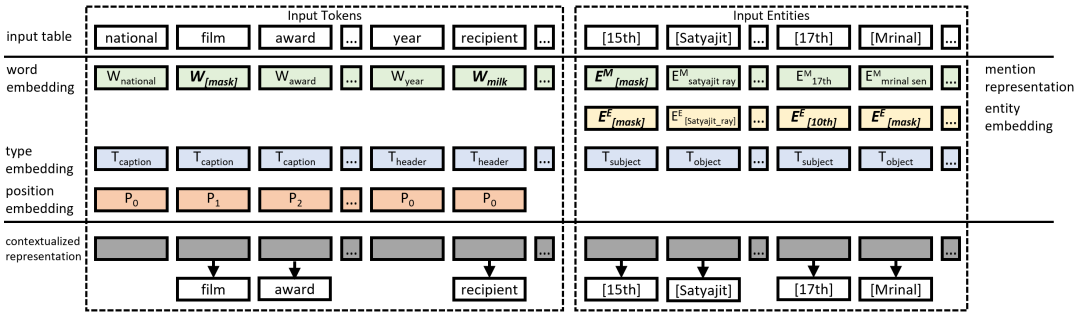


Figure 2: Illustration of the model input-output. The input table is first transformed into a sequence of tokens and entity cells, and masked for pre-training as described in Section 4.3. We then get contextualized representations for the table and use them for pre-training. Here [15th] (which means 15th National Film Awards), [Satyajit], ... are linked entity cells.

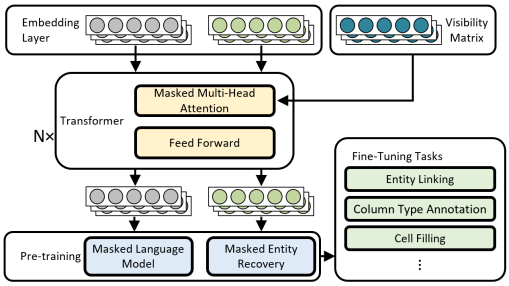


Figure 3: Overview of our TURL framework.

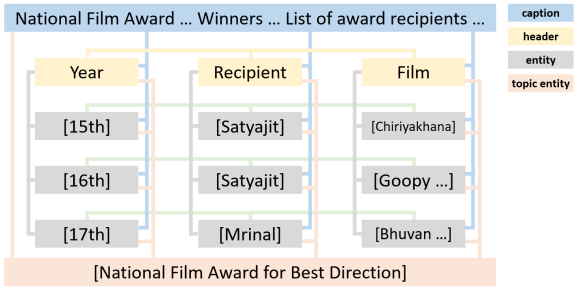


Figure 4: Graphical illustration of masked self-attention by our visibility matrix. Each token/entity in a table can only attend to its directly connected neighbors (shown as edges here, $M_{i,j} = 1$ only if two elements are connected).

three modules: (1) An embedding layer to convert different components of an input table into input embeddings, (2) N stacked structure-aware Transformer [28] encoders to capture the textual information and relational knowledge, and (3) a final projection layer for pre-training objectives. Figure 2 shows an input-output example.

4.1 Embedding Layer

Given a table $T=(C, H, E, e_t)$, we first linearize the input into a sequence of tokens and entity cells by concatenating the metadata and scanning the content row by row. The embedding layer then converts each token in C and H and each entity in E and e_t into an embedding representation.

Input token representation. For each token w, its vector representation is obtained as follows:

$$\mathbf{x}^{\mathrm{t}} = \mathbf{w} + \mathbf{t} + \mathbf{p}. \quad (1)$$

Here $\mathbf{w}$ is the word embedding vector, $\mathbf{t}$ is called the type embedding vector and aims to differentiate whether token w is in the table caption or a header, and $\mathbf{p}$ is the position embedding vector that provides relative position information for a token within the caption or a header.

Input entity representation. For each entity cell $e = (e^{\mathrm{e}}, e^{\mathrm{m}})$ (same for topic entity e_t), we fuse the information from the linked entity e^{e} and entity mention e^{m} together, and use an additional type embedding vector $\mathbf{t}^{\mathrm{e}}$ to differentiate three types of entity cells (i.e., subject/object/topic). Specifically, we calculate the input entity representation $\mathbf{x}^{\mathrm{e}}$ as:

$$\mathbf{x}^{\mathrm{e}} = \texttt{LINEAR}([\mathbf{e}^{\mathrm{e}}; \mathbf{e}^{\mathrm{m}}]) + \mathbf{t}^{\mathrm{e}}; \quad (2)$$

$$\mathbf{e}^{\mathrm{m}} = \texttt{MEAN}(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_j, \ldots). \quad (3)$$

Here $\mathbf{e}^{\mathrm{e}}$ is the entity embedding learned during pre-training. To represent entity mention e^{m}, we use its average word embedding $\mathbf{w}_j$'s. `LINEAR` is a linear layer to fuse $\mathbf{e}^{\mathrm{e}}$ and $\mathbf{e}^{\mathrm{m}}$.

A sequence of token and entity representations ($\mathbf{x}^{\mathrm{t}}$'s and $\mathbf{x}^{\mathrm{e}}$'s) are then fed into the structure-aware Transformer encoder to produce contextualized representations.

4.2 Structure-aware Transformer Encoder

We choose Transformer [28] as our base encoder block, which is composed of a multi-head self-attention layer followed by a fully connected layer. Transformer has been widely used in pre-trained language models [13, 25]. However, Transformer is originally designed for unstructured text sequences and cannot model the row-column structure, which is important for interpreting relational tables. We propose a visibility matrix M to model such structure information in a relational table.

Our visibility matrix acts as an attention mask so that each token (or entity) can only aggregate information from other structurally related tokens/entities during the self-attention calculation [28]. M is a symmetric binary matrix with $M_{i,j} = 1$ if and only if $\texttt{element}_j$ is visible to $\texttt{element}_i$. The `element` here can be a token in the caption or header, or an entity in a table cell. Specifically, we define M as follows:

- If $\texttt{element}_i$ is the topic entity or a token in table caption, $\forall j, M_{i,j} = 1$. *Table caption and topic entity are visible to all components of the table.*
- If $\texttt{element}_i$ is a token or an entity in the table and $\texttt{element}_j$ is a token or an entity in the same row or column, $M_{i,j} = 1$. *Entities and text content in the same row or the same column are visible to each other.*

4.3 Pre-training Objective

In order to pre-train our model on an unlabeled table corpus, we adopt the Masked Language Model (MLM) objec-

tive from BERT to learn representations for tokens in table metadata and propose a Masked Entity Recovery (MER) objective to learn entity cell representations.

Masked Language Model. We adopt the same MLM objective as BERT, which trains the model to capture the lexical, semantic and contextual information described by table metadata. Given an input token sequence including table caption and table headers, we simply mask some percentage of the tokens at random, and then predict these masked tokens. We adopt the same percentage settings as BERT. The data processor selects 20% of the token positions at random. For a selected position, (1) 80% of the time we replace it with a special [MASK] token, (2) 10% of the time we replace it with another random token, and (3) 10% of the time we keep it unchanged.

Given a token position selected for MLM with representation $\mathbf{h}^{\mathrm{t}}$ output by our encoder, the probability of predicting its original token $w \in \mathcal{W}$ is then calculated as:

$$P(w) = \frac{\exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{t}}) \cdot \mathbf{w}\right)}{\sum_{w_k \in \mathcal{W}} \exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{t}}) \cdot \mathbf{w}_k\right)}. \quad (4)$$

Masked Entity Recovery. We propose a novel Masked Entity Recovery (MER) objective to help the model capture the factual knowledge embedded in the table content as well as the associations between table metadata and table content. Essentially, we mask a certain percentage of input entity cells and then recover the linked entity based on surrounding entity cells and table metadata. This requires the model to infer the relation between entities from table metadata and to encode the knowledge in entity embeddings.

We also take advantage of entity mentions. Specifically, as shown in Eqn. 2, the input entity representation has two parts: the embedding $\mathbf{e}^{\mathrm{e}}$ and the mention representation $\mathbf{e}^{\mathrm{m}}$. For some percentage of masked entity cells, we only mask $\mathbf{e}^{\mathrm{e}}$, and as such the model receives additional entity mention information to help form predictions. This assists the model in building a connection between entity embeddings and entity mentions, and helps downstream tasks where only cell texts are available.

Specifically, the data processor first chooses 60% of entity cells at random. Here we adopt a higher masking ratio for MER compared with MLM, because oftentimes in downstream tasks, none or few entities are given. For one chosen entity cell, (1) 10% of the time we keep both $\mathbf{e}^{\mathrm{m}}$ and $\mathbf{e}^{\mathrm{e}}$ unchanged, (2) 63% (i.e., 70% of the left 90%) of the time we mask both $\mathbf{e}^{\mathrm{m}}$ and $\mathbf{e}^{\mathrm{e}}$, (3) 27% of the time we keep $\mathbf{e}^{\mathrm{m}}$ unchanged and mask $\mathbf{e}^{\mathrm{e}}$ (among which we replace $\mathbf{e}^{\mathrm{e}}$ with embedding of a random entity to inject noise in 10% of the time). In both MLM and MER, we keep a certain portion of the selected positions unchanged so that the model can generate good representations for non-masked tokens/cells.

Given an entity cell selected for MER with a contexutalized representation $\mathbf{h}^{\mathrm{e}}$ output by our encoder, the probability of predicting entity $e \in \mathcal{E}$ is then calculated as follows:

$$P(e) = \frac{\exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{e}}) \cdot \mathbf{e}^{\mathrm{e}}\right)}{\sum_{e_k \in \mathcal{E}} \exp\left(\texttt{LINEAR}(\mathbf{h}^{\mathrm{e}}) \cdot \mathbf{e}^{\mathrm{e}}_k\right)}. \quad (5)$$

In reality, considering the entity vocabulary $\mathcal{E}$ is quite large, we only use the above equation to rank entities from a given candidate set. For efficient training, we construct the candidate set with (1) entities in the current table, (2) entities that have co-occurred with those in the current table, and (3) randomly sampled negative entities.

We use cross-entropy loss for both MLM and MER objectives and the final pre-training loss is given as follows:

$$loss = \sum \log\left(P(w)\right) + \sum \log\left(P(e)\right), \quad (6)$$

where the sums are over all tokens and entity cells selected in MLM and MER respectively.

Pre-training details. In this work, we leverage a pre-trained TinyBERT [18] model to initialize our structure-aware Transformer encoder. We use the Adam [20] optimizer with a linearly decreasing learning rate. The initial learning rate is 1e-4 and we pre-train the model for 80 epochs.

4.4 Pre-training Dataset Construction

We construct the pre-training data based on the WikiTable corpus [2] which contains around 1.65M tables extracted from Wikipedia pages. We process and extract relational tables from the corpus, which are further partitioned into the pre-training and held-out validation/test sets.

Pre-processing. For each table, we concatenate the page title and section title with the table caption to obtain a more comprehensive description. For each cell, we leverage hyperlinks to Wikipedia pages in it to normalize different entity mentions corresponding to the same entity. To identify relational tables, we first locate entity columns that contain at least one linked cell and have meaningful headers (i.e., not *note*, digit numbers, etc.). We then identify relational tables by finding tables that have a subject column. We employ a simple heuristic for subject column detection: the subject column must be located in the first two columns of the table and contain unique entities. We further filter out tables containing less than three entities or more than twenty columns. With this process, we obtain 670,171 relational tables.

Data partitioning. We further select a high quality subset of the relational tables for evaluation: From tables that have (1) more than four linked entities in the subject column, (2) at least three entity columns including the subject column, and (3) more than half of the cells in entity columns are linked. We randomly select 10000 to form a held-out set, and partition it into validation/test sets via a rough 1:1 ratio for model evaluation. All relational tables not in the evaluation set are used for pre-training. In sum, we have 570171 / 5036 / 4964 tables respectively for pre-training/validation/test sets.

Most tables in our pre-training dataset have moderate size, with an average of 13 rows and 2 entity columns. We use the token vocabulary from BERT [13], and construct the entity vocabulary based on entities that appear more than once in the pre-training table corpus (926,135 entities).

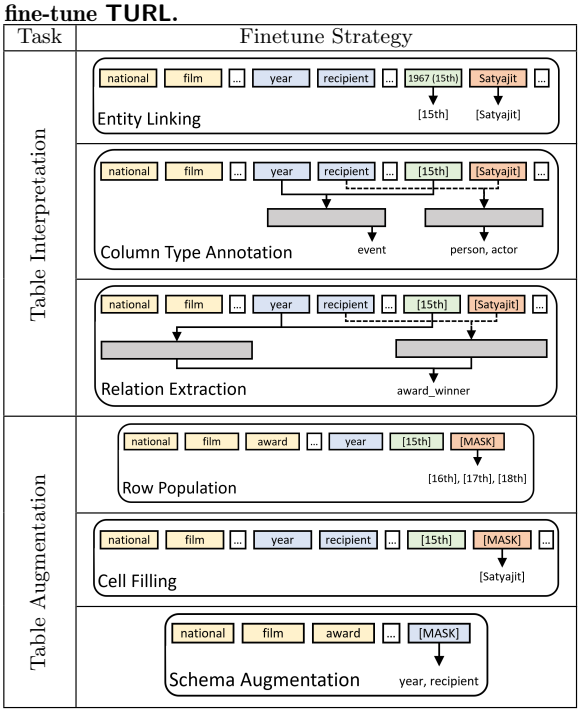
5. EXPERIMENTS

To systematically evaluate our pre-trained framework as well as facilitate research, we compile a table understanding benchmark consisting of 6 widely studied tasks covering table interpretation and table augmentation. Our pre-trained framework is general and can be applied to all the tasks with minimal task-specific fine-tuning.

5.1 Entity Linking

Definition. Given a table T and a knowledge base $\mathcal{KB}$, entity linking aims to link each potential mention in cells of T to its referent entity $e \in \mathcal{KB}$. Here we consider $\mathcal{KB}$ as structured databases that store descriptions and relations of entities in structured formats such as RDF triples.

Table 1: An overview of our benchmark tasks and strategies to fine-tune TURL.



Fine-tuning TURL. For entity linking, we first generate candidate entities using the Wikidata Lookup service. We then rank them by their matching scores with the target cell. We obtain a contextualized representation $\mathbf{h}^{\mathrm{e}}$ for each cell with its cell text (i.e., entity mention $\mathbf{e}^{\mathrm{m}}$ in Eqn. 2) and table metadata. To represent each candidate entity, we utilize the name, description and type information from a KB. Specifically, for a KB entity e, given its name N and description D (both are a sequence of words) and types T, we get its representation $\mathbf{e}^{\mathrm{kb}}$ as follows:

$$\mathbf{e}^{\mathrm{kb}} = [\texttt{MEAN}_{w \in N}\left(\mathbf{w}\right), \texttt{MEAN}_{w \in D}\left(\mathbf{w}\right), \texttt{MEAN}_{t \in T}\left(\mathbf{t}\right)]. \quad (7)$$

Here, $\mathbf{w}$ is the embedding for word w, and $\mathbf{t}$ is the embedding for entity type t. We then calculate a matching score between $\mathbf{e}^{\mathrm{kb}}$ and $\mathbf{h}^{\mathrm{e}}$ similarly as Eqn. 5. We do not use the entity embeddings pre-trained by our model here, as the goal is to link mentions to entities in a target KB, not necessarily those appear in our pre-training table corpus. The model is fine-tuned with a cross-entropy loss.

Datasets and baselines. We test with three datasets: (1) The Wikipedia gold standards (WikiGS) dataset [14], (2) our own test set from the held-out test tables mentioned in Section 4.4, (3) the T2D dataset [21] with Web tables from websites other than Wikipedia. We use names and descriptions returned by Wikidata Lookup, and entity types from DBpedia. We compare against the most recent methods for table entity linking T2K [26], Hybrid II [14], as well as the off-the-shelf Wikidata Lookup service.

Results. Results are shown in Table 2. Our model gets the best F1 score and substantially improves precision on WikiGS and our own test set. The disambiguation accuracy on WikiGS is 89.62% (predict the correct entity if it is in the candidate set). A more advanced candidate generation module can help achieve better results in the future. On the T2D dataset, all models perform much better than on the two Wikipedia datasets, mainly because of its smaller size and limited types of entities. The Wikidata Lookup baseline achieved high performance, and re-ranking using our model does not further improve. However, we adopt simple reweighting[2] to take into account the original result returned by Wikidata Lookup, which brings the F1 score to 0.82. This demonstrates the potential of using features such as entity popularity (used in Wikidata Lookup) and ensembling strong base models.

Table 2: Model evaluation on entity linking. We use the same TURL + fine-tuning model for all three datasets.

Method	WikiGS			Our Test Set			T2D		
	F1	P	R	F1	P	R	F1	P	R
T2K [26]	34	70	22	-	-	-	82	**90**	76
Hybrid II [14]	64	69	**60**	-	-	-	**83**	85	**81**
Wikidata Lookup	57	67	49	62	62	60	80	86	75
TURL + fine-tuning	**67**	**79**	58	**68**	**71**	**66**	78	83	73
+ reweighting	-	-	-	-	-	-	82	88	77

Table 3: Model evaluation on column type annotation.

Method	F1	P	R
Sherlock (only entity mention) [17]	78.47	88.40	70.55
TURL + fine-tuning (only entity mention)	88.86	90.54	87.23
TURL + fine-tuning	**94.75**	**94.95**	**94.56**
w/o table metadata	93.77	94.80	92.76
only table metadata	90.24	89.91	90.58

5.2 Column Type Annotation

Definition. Given a table T and a set of semantic types $\mathcal{L}$, column type annotation refers to the task of annotating a column in T with $l \in \mathcal{L}$ so that all entities in the column have type l. Note that a column can have multiple types.

Earlier work [26, 36] often coupled column type annotation with entity linking. More recently, [6, 7, 17] have studied column type annotation based on cell texts only. Here we adopt a similar setting, i.e., use the available information in a given table directly for column type annotation without performing entity linking first.

Fine-tuning TURL. We calculate the probability of predicting type l given column $\mathbf{h}_c$ as follows:

$$\mathbf{h}_c = [\texttt{MEAN}\left(\mathbf{h}_i^{\mathrm{t}}, \dots\right); \texttt{MEAN}\left(\mathbf{h}_j^{\mathrm{e}}, \dots\right)]; \quad (8)$$

$$P(l) = \mathrm{Sigmoid}\left(\mathbf{h}_c W_l + b_l\right). \quad (9)$$

Here $\mathbf{h}_i^{\mathrm{t}}$'s are representations of tokens in the column header, $\mathbf{h}_j^{\mathrm{e}}$'s are representations of entity cells in the column. We optimize the model with binary cross-entropy loss.

Datasets and baselines. We refer to Freebase [15] to obtain semantic types $\mathcal{L}$. For each column, we use the common types of its entities in Freebase as annotations. We further filter columns with less than three linked entities and keep only the most representative types. In the end, we get a total number of 255 types, 628,254 columns from 397,098 tables for training, 13,025 (13,391) columns from 4,764 (4,844) tables for testing (validation). We also test our model on two existing small-scale datasets, T2D-Te and Efthymiou [7]. Follow the setting in [7], we use 70% of T2D as training data (250 columns). We compare our results with Sherlock [17] and HNN + P2Vec [7].

[2]We simply reweight the score of the top-1 prediction by our model with a factor of 0.8 and compare it with the top-1 prediction returned by Wikidata Lookup. The higher one is chosen as final prediction.

Table 4: Accuracy on T2D-Te and Efthymiou for column type annotation.

Method	T2D-Te	Efthymiou
HNN + P2Vec (entity mention + KB) [7]	0.966	0.650
TURL + fine-tuning (only entity mention)	0.940	0.516
+ table metadata	0.962	0.746

Table 5: Model evaluation on relation extraction.

Method	F1	P	R
BERT-based	90.94	91.18	90.69
TURL + fine-tuning (only table metadata)	92.13	91.17	93.12
TURL + fine-tuning	**94.91**	**94.57**	**95.25**
w/o table metadata	93.85	93.78	93.91

Results. Results are shown in Table 3. Our model substantially outperforms the baselines. Further analysis for several types shows that although all methods work well for coarse-grained types like `person`, using table metadata greatly improves the results for fine-grained types like `actor` and `pro_athlete`. This indicates the importance of table context for predicting fine-grained column types. Results on T2D-Te and Efthymiou are summarized in Table 4. We can see that without using KB information, our model with entity mention and table metadata still outperforms or is on par with the baseline. However, when using only entity mention, our model does not perform as well as the baseline, especially when generalizing to Efthymiou. This is because: (1) Our model is pretrained with table metadata and entity embedding. Removing both creates a big mismatch between pretraining and fine-tuning. (2) With only 250 training instances, it is easy for deep models to overfit.

5.3 Relation Extraction

Definition. Given a table T and a set of relations $\mathcal{R}$ in KB. For a subject-object column pair in T, we aim to annotate it with $r \in \mathcal{R}$ so that r holds between all entity pairs in the columns.

Most existing work [26, 36] infers relations between columns based on entity linking results. However, such methods rely on entity linking performance and suffer from KB incompleteness. Here we aim to conduct relation extraction without explicitly linking table cells to entities. *This is important as it allows the extraction of new knowledge from Web tables for tasks like knowledge base population.*

Fine-tuning TURL. We use similar model architecture as column type annotation as follows.

$$P(r) = \text{Sigmoid}\left([\mathbf{h}_c; \mathbf{h}_{c'}]W_r + b_r\right). \tag{10}$$

Here $\mathbf{h}_c, \mathbf{h}_{c'}$ are aggregated column representations obtained same as Eqn. 8. We optimize with binary cross-entropy loss.

Datasets and baselines. We prepare datasets for relation extraction in a similar way as column type annotation. Specifically, we obtain relations $\mathcal{R}$ from Freebase. For each table in our corpus, we pair its subject column with each of its object columns, and annotate the column pair with relations shared by more than half of the entity pairs in the columns. Finally, we obtain a total number of 121 relations, 62,954 column pairs from 52,943 tables for training, and 2072 (2,175) column pairs from 1467 (1,560) tables for testing (validation). We compare our model with a text based relation extraction model [37]. Here we adapt the setting by treating the concatenated table metadata as a sentence, and the headers of the two columns as entity mentions. We also implement an entity linking based system using our entity linking model described in Section 5.1, and predict a relation if it holds between a minimum portion of linked entity pairs in KB.

Table 6: Relation extraction results of an entity linking based system, under different agreement ratio thresholds.

Min Ag. Ratio	F1	P	R
0	68.73	60.33	**79.85**
0.4	82.10	94.65	72.50
0.7	63.10	99.37	46.23

Results. From the main results in Table 5, we can see that our model outperforms the BERT-based baseline under all settings. Moreover, we notice that our model converges much faster in comparison to the BERT-based baseline during training, demonstrating that our model learns a better initialization through pre-training. Results for the entity linking based system are summarized in Table 6. We can see that it can achieve high precision, but suffers from low recall: The upper-bound of recall is only 79.85%, achieved at an agreement ratio of 0 (i.e., taking all relations that exist between the linked entity pairs as positive). As seen from Table 5 and 6, our model also substantially outperforms the system based on a strong entity linker.

5.4 Row Population

Definition. Given a partial table T, and an optional set of seed subject entities, row population aims to retrieve more entities to fill the subject column.

Fine-tuning TURL. We first generate candidate entities follow [33], which formulates a search query using either the table caption or seed entities and then retrieves tables via the BM25 retrieval algorithm. Subject entities in those retrieved tables will be candidates for row population. We then append the [MASK] token to the input, and use the hidden representation $\mathbf{h}^e$ of [MASK] to rank these candidates as shown in Table 1. We fine-tune our model with multi-label soft margin loss.

Table 7: Model evaluation on row population. Recall is the same for all methods because they share the same candidate generation module.

# seed	0		1	
Method	MAP	Recall	MAP	Recall
EntiTables [33]	17.90	63.30	42.31	78.13
Table2Vec [9]	-	63.30	20.86	78.13
TURL + fine-tuning	**40.92**	63.30	**48.31**	78.13

Datasets and baselines. We use tables in our pre-training set that have more than 3 subject entities for fine-tuning TURL and developing baseline models, and use tables in our held-out set with more than 5 subject entities for evaluation. In total, we obtain 432,660 tables for fine-tuning with 10 subject entities on average, and 4,132 (4,205) tables for testing (validation) with 16 (15) subject entities on average. We adopt models from [33] and [9] as baselines.

Results. We experiment both with and without seed entity. As shown in Table 7, our method outperforms all baselines. In particular, previous methods rely on entity similarity and are not applicable or have poor results when there is no seed entity available. Our method achieves a decent performance even without any seed entity, which demonstrates the effectiveness of TURL for generating contextualized representations based on both table metadata and content.

Table 8: Precision @ K on cell filling.

Method	P @ 1	P @ 3	P @ 5	P @ 10
Exact	51.36	70.10	76.80	84.93
H2H	51.90	70.95	77.33	85.44
H2V	52.23	70.82	77.35	85.58
TURL	**54.80**	**76.58**	**83.66**	**90.98**

Table 9: Mean Average Precision on schema augmentation.

Method	#seed column labels	
	0	1
kNN	80.16	**82.01**
TURL + fine-tuning	**81.94**	77.55

5.5 Cell Filling

Definition. Given a partial table T with the subject column filled and an object column header, cell filling aims to predict the object entity for each subject entity.

Fine-tuning TURL. Since cell filling is very similar to the MER pre-training task, we do not fine-tune the model, and directly use [MASK] to select from candidate entities same as MER (Eqn. 5). We use the same candidate generation module for TURL and baselines, which we describe in the next section.

Datasets and baselines. To evaluate different methods on this task, we use the held-out test tables in our pre-training phase and extract from them those subject-object column pairs that have at least three valid entity pairs. Finally we obtain 9,075 column pairs for evaluation. We adopt [34] as our base model. It has two main components, candidate value finding and value ranking. The same candidate value finding module is used for all methods: Given a subject entity e and object header h for the to-be-filled cells, we find all entities that appear in the same row with e in our pre-training table corpus, and only keep entities whose source header h' is related to h. Here we use the formula from [34] to measure the relevance of two headers $P(h'|h)$,

$$P(h'|h) = \frac{n(h', h)}{\sum_{h''} n(h'', h)}. \tag{11}$$

Here $n(h', h)$ is the number of table pairs in the table corpus that contain the same entity for a given subject entity in columns h' and h. We can then get the probability of the candidate entity e belongs to the cell $P(e|h)$ as follows:

$$P(e|h) = \text{MAX}\left(\text{sim}(h', h)\right). \tag{12}$$

Here h''s are the source headers associated with the candidate entity in the pre-training table corpus. $\text{sim}(h', h)$ is the similarity between h' and h. We develop three baseline methods for $\text{sim}(h', h)$: (1) **Exact**: predict the entity with exact matched header, (2) **H2H**: use the $P(h'|h)$ described above. (3) **H2V**: similar to [9], we train header embeddings with Word2Vec on the table corpus. We then measure the similarity between headers using cosine similarity.

Results. We use the same candidate value finding module for all methods, which obtains 61.45% recall with an average of 86 candidates. For value ranking, we only consider those test instances with the target object entity in the candidate set and evaluate them under Precision@K (or, P@K). From Table 8 we can see that: (1) Simple **Exact** match achieves decent performance, and using **H2H** or **H2V** only sightly improves the results. (2) Even though our model directly ranks the candidate entities without explicitly using their source table information, it outperforms other methods. This indicates that our model already encodes the factual knowledge in tables into entity embeddings through pre-training.

5.6 Schema Augmentation

Definition. Given a partial table T, which has a caption and zero or a few seed headers, and a header vocabulary $\mathcal{H}$, schema augmentation aims to recommend a ranked list of headers $h \in \mathcal{H}$ to add to T.

Fine-tuning TURL. We concatenate the table caption, seed headers and a [MASK] token as input to our model. The output for [MASK] is then used to predict the headers in a given header vocabulary $\mathcal{H}$. We fine-tune our model use binary cross-entropy loss.

Datasets and baselines. We collect $\mathcal{H}$ from the pre-training table corpus. We normalize the headers using simple rules, only keep those that appear in at least 10 different tables, and finally obtain 5652 unique headers, with 316,858 training tables and 4,646 (4,708) test (validation) tables. We adopt the method in [33] (kNN) which searches our pre-training table corpus for related tables, and use headers in those related tables for augmentation.

Results. From Table 9, we observe that both kNN baseline and our model achieve good performance. Our model works better when no seed header is available, but does not perform as well when there is one seed header. Further analysis shows that: One major reason why kNN works well is that there exist tables in the pre-training table corpus that are very similar to the query table and have almost the same table schema. On the other hand, our model oftentimes suggests plausible, semantically related headers, but misses the ground-truth headers.

6. CONCLUSION AND FUTURE WORK

We present TURL, a novel framework for learning deep contextualized representations on relational Web tables via self-supervised pre-training and task-specific fine-tuning. We propose a structure-aware Transformer encoder to model the row-column structure, and a new Masked Entity Recovery objective to capture the relational knowledge in tables. To facilitate research in this direction, we compile a new benchmark that consists of 6 different tasks for table understanding and conduct comprehensive experiments.

This paper, for the first time to our knowledge, studies the application of the pre-training/fine-tuning paradigm on relational table understanding. Although we mainly focus on relational Web tables, our techniques could be extended to other (semi-)structured data formats, such as database tables, SpreadSheets, JSON etc. In this work, we examine the utility of our approach for table interpretation and table augmentation. Some recent work also researches the potential of the pre-training/fine-tuning paradigm on other tasks such as table searching [29], table to text generation [30] and building natural language interfaces [10, 32]. More broadly, we believe that developing more advanced machine learning techniques to understand (semi-)structured data could benefit all stages of data management, from data preparation to data query and analysis. We hope our work could inspire more future work along that direction.

Acknowledgments: Authors at the Ohio State University were sponsored in part by Google Faculty Award, the Army Research Office under cooperative agreements W911NF-17-

1-0412, NSF Grant IIS1815674, NSF OAC-2112606, NSF CAREER #1942980, Fujitsu gift grant, and Ohio Supercomputer Center [5]. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

7. REFERENCES

[1] A. Ahmadov, M. Thiele, J. Eberius, W. Lehner, and R. Wrembel. Towards a hybrid imputation approach using web tables. In *BDC*, 2015.

[2] C. S. Bhagavatula, T. Noraset, and D. Downey. Tabel: Entity linking in web tables. In *ISWC*, 2015.

[3] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *VLDB*, 2008.

[4] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. In *WebDB*, 2008.

[5] O. S. Center. Ohio supercomputer center, 1987.

[6] J. Chen, E. Jiménez-Ruiz, I. Horrocks, and C. A. Sutton. Colnet: Embedding the semantics of web tables for column type prediction. In *AAAI*, 2018.

[7] J. Chen, E. Jiménez-Ruiz, I. Horrocks, and C. A. Sutton. Learning semantic annotations for tabular data. In *IJCAI*, 2019.

[8] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, 2012.

[9] L. M. Deng, S. Zhang, and K. Balog. Table2vec: Neural word and entity embeddings for table population and retrieval. In *SIGIR*, 2019.

[10] X. Deng, A. Hassan, C. Meek, O. Polozov, H. Sun, and M. Richardson. Structure-grounded pretraining for text-to-sql. In *NAACL*, 2021.

[11] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu. Turl: Table understanding through representation learning. *VLDB*, 2020.

[12] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu. Turl: Table understanding through representation learning. `https://arxiv.org/abs/2006.14806`, 2020.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[14] V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, and V. Christophides. Matching web tables with knowledge base entities: From entity lookups to entity embeddings. In *ISWC*, 2017.

[15] Google. Freebase data dumps. `https://developers.google.com/freebase/data`.

[16] J. Herzig, P. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. In *ACL*, 2020.

[17] M. Hulsebos, K. Z. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, cCaugatay Demiralp, and C. A. Hidalgo. Sherlock: A deep learning approach to semantic data type detection. *KDD*, 2019.

[18] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. Tinybert: Distilling bert for natural language understanding. *ArXiv*, 2019.

[19] E. Jiménez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, and K. Srinivas. Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems. In *ESWC*, 2020.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ArXiv*, 2015.

[21] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer. A large public corpus of web tables containing time and context metadata. In *WWW*, 2016.

[22] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan. Deep entity matching with pre-trained language models. *ArXiv*, 2020.

[23] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. In *VLDB*, 2010.

[24] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, 2013.

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[26] D. Ritze, O. Lehmberg, and C. Bizer. Matching html tables to dbpedia. In *WIMS*, 2015.

[27] Q. shan Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 2017.

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

[29] F. Wang, K. Sun, M. Chen, J. Pujara, and P. Szekely. Retrieving complex tables with multi-granular graph representation learning. *SIGIR*, 2021.

[30] T. Xie, C. H. Wu, P. Shi, R. Zhong, T. Scholak, M. Yasunaga, C.-S. Wu, M. Zhong, P. Yin, S. I. Wang, V. Zhong, B. Wang, C. Li, C. Boyle, A. Ni, Z. Yao, D. Radev, C. Xiong, L. Kong, R. Zhang, N. A. Smith, L. Zettlemoyer, and T. Yu. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *ArXiv*, 2022.

[31] P. Yin, G. Neubig, W. tau Yih, and S. Riedel. Pretraining for joint understanding of textual and tabular data. In *ACL*, 2020.

[32] T. Yu, C.-S. Wu, X. V. Lin, Y. C. Tan, X. Yang, D. Radev, C. Xiong, et al. Grappa: Grammar-augmented pre-training for table semantic parsing. In *ICLR*, 2020.

[33] S. Zhang and K. Balog. Entitables: Smart assistance for entity-focused tables. In *SIGIR*, 2017.

[34] S. Zhang and K. Balog. Auto-completion for data cells in relational tables. In *CIKM*, 2019.

[35] S. Zhang and K. Balog. Web table extraction, retrieval, and augmentation: A survey. *TIST*, 2020.

[36] Z. Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, 2017.

[37] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu. Ernie: Enhanced language representation with informative entities. In *ACL*, 2019.