DeepTable: a permutation invariant neural network for table orientation classification

Maryam Habibi · Johannes Starlinger · Ulf Leser

Received: 13 September 2019 / Accepted: 17 August 2020 / Published online: 8 September 2020
© The Author(s) 2020

Abstract
Tables are a common way to present information in an intuitive and concise manner. They are used extensively in media such as scientific articles or web pages. Automatically analyzing the content of tables bears special challenges. One of the most basic tasks is determination of the orientation of a table: In column tables, columns represent one entity with the different attribute values present in the different rows; row tables are vice versa, and matrix tables give information on pairs of entities. In this paper, we address the problem of classifying a given table into one of the three layouts horizontal (for row tables), vertical (for column tables), and matrix. We describe DeepTable, a novel method based on deep neural networks designed for learning from sets. Contrary to previous state-of-the-art methods, this basis makes DeepTable invariant to the permutation of rows or columns, which is a highly desirable property as in most tables the order of rows and columns does not carry specific information. We evaluate our method using a silver standard corpus of 5500 tables extracted from biomedical articles where the layout was determined heuristically. DeepTable outperforms previous methods in both precision and recall on our corpus. In a second evaluation, we manually labeled a corpus of 300 tables and were able to confirm DeepTable to reach superior performance in the table layout classification task. The codes and resources introduced here are available at https://github.com/Marhabibi/DeepTable.

Keywords
Information discovery · Tabular data · Table orientation classification · Deep learning · Machine learning

Responsible editor: Johannes Fürnkranz.

Electronic supplementary material
The online version of this article (https://doi.org/10.1007/s10618-020-00711-x) contains supplementary material, which is available to authorized users.

✉️ Ulf Leser
leser@informatik.hu-berlin.de

Maryam Habibi
habibima@informatik.hu-berlin.de

1 Humboldt Universität zu Berlin, Berlin, Germany
1 Introduction

An ever growing amount of information is managed and exchanged in digitized form, especially in the form of text (web pages, scientific articles, reports, newspapers, books etc.). Efficient and effective management of such large collections of text depends on the creation of computational tools for their automated analysis. A particularly important type of digital information that has seen comparably little attention in research so far are tables. Tables are a universal and intuitive means to structure information in a two-dimensional manner with a high density of information. They are used extensively in scientific articles, business reports, product descriptions, web pages, etc. Nevertheless, tables only recently were “discovered” as important first-class objects in research, mostly driven by the public availability of millions of tables on the web (Gonzalez et al. 2010; Cafarella et al. 2008a; Lehmberg et al. 2016) or from Wikipedia (Bhagavatula et al. 2013).

Tables are different from pure text as they have an explicit structure which itself carries meaning. Unlike text, using sentences and paragraphs to convey a meaning from a sequence of words, tables impose meaning mostly through the arrangement of values in columns and rows. Most tables can be classified into one of three main orientation types: Horizontal, vertical, and matrix (Crestan and Pantel 2011; Braunschweig 2015). Tables with horizontal or vertical layouts represent entities as rows or columns, respectively, where the different columns or rows contain values for attributes of this entity. In contrast, a matrix layout represents information on pairs of entities, one represented by the column, one by the row. Humans easily recognize the layout of tables by either focusing on header-like information (given as row header in row layout, as column header in column layout, and as row and column header in matrix layout), or by recognizing sets of values with similar semantics. However, this task is challenging for computer programs, as, in tables found in the wild, headers are often either missing or not clearly formatted as such, and identifying sets of similar semantics is notoriously difficult. Nevertheless, to enable automatic extraction of coherent information from tables, we first have to correctly recognize the tables’ orientation. Accordingly, this task is the first step in many types of table analysis, such as table search (Liu et al. 2007), table clustering and classification (Yoshida et al. 2001; Vilain et al. 2006), table auto-completion (Agassi et al. 2004), table fusion (Gonzalez et al. 2010), question answering from tables (Pinto et al. 2002) or filling missing values of databases (Ritze et al. 2015). For instance, imagine a user with an incomplete table of the names of mountains in the world and their heights with horizontal layout. This user could be interested in augmenting their data with the heights of further mountains and performs a similarity search for tables similar to theirs. Assume one potentially good match had a vertical layout (i.e., a table with highly similar semantics but represented with a different layout). To allow identification of the respective table even as a candidate match, table orientation classification is inevitable.

Most previous work on management and analysis of tables simply assumes all tables to have a horizontal layout if not indicated otherwise by explicit headers. This, however, covers only a fraction of real-live tables. Recently, several studies have been conducted to identify the different types of web tables, focusing on whether units within a table represent distinct entities (table type entity) or whether relationships
between individual rows or columns exist (table type relational or matrix). The earliest techniques trained a classifier for this task based on handcrafted features (Crestan and Pantel 2011; Eberius et al. 2015). To improve accuracy, Nishida et al. (2017) proposed an approach where features are automatically learned. Their approach takes inspiration from deep learning methods for image classification, considering tables analogous to images. However, the notion of structure is different in tables than in images: While for most tables changing the order of rows/columns (except header rows/columns) does not change the meaning of a table, images become completely scrambled when rows or columns are permuted. For example, imagine the table in the above example with horizontal layout, changing the order of mountain names with their heights does not change its orientation.

In this work, we propose DeepTable, a novel table orientation classification method that classifies tables into one of the three main layout types (Braunschweig 2015): Horizontal, vertical, and matrix. DeepTable is a neural network based classifier, but unlike all previous work it is order-invariant. It builds on a network architecture designed for representing sets (as opposed to lists or sequences) (Qi et al. 2017; Zaheer et al. 2017; Lee et al. 2019). Order-invariant here means that changing the order of cells within rows or columns does not change the classification results. DeepTable actually uses two such networks, where one is capable of distinguishing between horizontal and vertical layouts and the second representation contributes to determine matrix representation. We evaluate DeepTable on tables extracted from scientific articles in biomedical domain, but the method can be applied to tables from scientific articles in other domains or languages. It also does not need any formatting tags, e.g., for header recognition.

We also present a corpus for the evaluation of our method, as—to the best of our knowledge—there exists no freely available annotated corpus for table layout classification. Tables in our corpus are extracted from scientific articles in the life sciences domain. These have much better quality than web tables, which allows us to focus on the table orientation task and not on pre-processing or data cleansing. We first annotated the entire corpus (12,161 tables) using a set of heuristics defined and evaluated in Isberner (2016). This method achieves high performance, but relies on header information, which is recognized using font-based formatting clues. In contrast, DeepTable does not require any such information. Evaluating DeepTable on this large silver-standard corpus revealed that it outperforms TabNet, the current state-of-the-art (Nishida et al. 2017) in table layout classification, and several other baselines. To account for the bias this silver-standard corpus carries from the used heuristics and from the fact that only tables with explicit header information are included, we repeated our evaluation on a second corpus of 300 tables which we annotated manually. Again, DeepTable outperformed all other methods we compared to.

The paper is organized as follows. In Sect. 2, we review the state-of-the-art in table layout classification. We define the problem of table orientation classification and explain the proposed order-invariant deep neural architecture in Sect. 3. Section 4 presents data preparation, the methods we compare against, and our evaluation settings and metrics. In Sect. 5, we provide the results of our evaluation and conclude in Sect. 6.
2 Related work

Several different classification problems for tables have been discussed in the scientific literature, each concerning a specific aspect of how tables are used and presented: Genuine versus non-genuine table classification determines whether a table found in HTML documents merely serves the structure of the HTML document itself (non-genuine) or contains actual tabular data (genuine); relational versus entity type classification determines whether a table contains information about a single object only (entity) or follows a relational scheme with a header defining a set of properties and the rows containing the respective information for several individual objects (relational); and finally, table orientation classification determines the tables layout as either horizontal, vertical, or matrix (described in detail in Sect. 3). Each of these may use similar features and methods but differ in the classification result (and in the training and evaluation data used). Hence, we here discuss previous work for each of these tasks, some of which combines more than one of these tasks into a single classification.

The earliest studies have been performed by Wang and Hu (2002), where a decision tree and an SVM classifier are used to classify tables into two types (genuine and non-genuine tables), and by Cafarella et al. (2008b), where tables are classified into two types (relational and non-relational tables) using rule-based classifiers. Crestan and Pantel (2011) designed more features for the classification of web tables into eight different table types (vertical listing, horizontal listing, attribute/value, matrix, enumeration, form, navigational, and formatting). They trained a Gradient Boosted Decision Tree using content and layout features (e.g., maximum number of rows, columns and cell length, number of empty cells, number of cells containing number etc) manually extracted from tables. Eberius et al. (2015) extended Crestan and Pantel (2011) by introducing more features. The above methods utilized table formatting tags which make them limited to the subset of those tables that contain such formatting tags.

More recently, deep learning and word embeddings have been employed to automatically extract table features for table type classification. Nishida et al. (2017) classify tables into six types (vertical and horizontal relational, vertical and horizontal entity, matrix, and other tables). They proposed a deep learning architecture called TabNet using several consecutive convolution layers (CNNs) with residual units for modeling the relation among cells within rows and columns, where each cell itself is modeled by a Long Short-Term Memory (LSTM) layer with attention mechanism. Their output is fed into a softmax layer for classification. In another study, Ghasemi-Gol and Szekely (2018) proposed an unsupervised method called TabVec for clustering tables into five types (entity, list, relational, matrix, and other tables) using k-means on the table vectors trained using random indexing (Kanerva 2009) due to the lack of annotated table data for this table classification task.

The above methods assume that tables have a structure similar to that of images in terms of being a grid of data points. However, methods used for image classification are not directly suitable for tables because, in tables, the exact order of cells in a row (or column) and the order of rows (or columns) within the table most often do not carry substantial information (Xu et al. 2017; Petrovski et al. 2018)—unlike the order
of pixels in an image. In contrast, our method is tailored to assume that the contents of columns or rows have more set-like semantics and the permutation of cells inside them does not change classification output. Recently, several neural networks are proposed for set representation where a permutation-invariant neural network is developed to model the relation among the elements of a set using various aggregation strategies such as average and maximum operations (Qi et al. 2017; Zaheer et al. 2017) or self-attention layers (Lee et al. 2019). We adapt these order-invariant networks for our task. Moreover, our method does not need a difficult and expensive feature engineering process. Next to the table layout classification task explored in this paper, our method may thus be applicable to other table type classification tasks as well. Due to the lack of freely available annotated data from previous authors, however, we focus our evaluation on a corpus of tables specifically prepared for the table orientation task.

3 Table orientation classification

Tables within documents can have various layouts: horizontal, vertical or a combination of both in a matrix. The problem of determining the correct table orientation can be modeled as a classification problem that chooses the best orientation for the given table. To automatically extract relevant features for table layout recognition, we use the power of deep learning methods. In the following, we first describe three table orientation types with examples and then explain the deep neural network architecture proposed for this task.

3.1 Table orientation types illustration

We exemplify the three table orientation types in Table 1(a)–(c):

**Vertical layout** In a vertical layout as shown in Table 1(a), cells in a row are instances of a concept represented by the header in the first column. For example, “Sousa et al. 2015” and “Zhou and Xing 2015” in the first row are instances of the header concept “Reference” in the first column. Similarly, “9” and “9.5” are values of the header concept “PH” or “35” and “37” are values of the header concept “Temperature”. In this example, all columns except the first one can be swapped without changing the meaning or the message of the table. Moreover, all the rows—even the first row—can be swapped while the meaning remains unchanged. We categorize entity and relational tables with vertical layout, defined by Nishida et al. (2017) as two different classes, into a unified vertical layout class.

**Horizontal layout** In the horizontal layout, Table 1(b), each table column is a data set for an attribute in the first row. For instance, cells “60”, “75”, and “90” in the first column are the instances of the attribute “Weight (kg)” represented by the header in the first row. Similarly, “50”, “55” and “60” are the instances of the attribute “No prior CTX (mg)”. In this case, the first row containing header information can-
not be exchanged with other rows while other rows can be swapped amongst each other. All columns can be swapped as well. We uniformly consider both entity and relational tables with horizontal layout as horizontal layout.

Matrix layout Table 1(c) shows an example matrix layout, where each row is about one of the “study aspect” with its “inclusion criteria” represented by the header in the first row. Each study design is given by the header in the first column like “study design”, “intervention” and “type”. In this case both first row and first column contain headers and all rows (columns) except the first row (column) can be swapped with each other.

3.2 Table orientation neural network overview

We call the proposed deep neural network classifier for table orientation learning DeepTable. In this network, tables are modeled by two representations: (a) column-wise representation derived from an assumed horizontal layout and (b) row-wise representation from an assumed vertical layout. The two representations are concatenated and given to a softmax classifier to choose one of the classes of horizontal, vertical and matrix. One of the representations is enough to classify vertical and horizontal layouts, the second representation contributes in separation of matrix layout from the two other layouts.

DeepTable considers cells within each row and each column as items of a set. We take inspiration from neural networks proposed for set representation where dif-

| Table 1 | Table samples per layout: (a) a table with vertical layout, (b) a table with horizontal layout and (c) a table with matrix layout |
|---------|----------------------------------------------------------------------------------------------------------------------------------|
| Reference | Sousa et al. 2015 | Zhou and Xing 2015 | Zhou and Xing 2015 |
| Reactor type | Gas lift with three-phase separator | Anaerobic filter | Anaerobic filter |
| e acceptor | Sulfate | Sulfate | Sulfate |
| e donor | pH | H 2 | Formate | Ethanol |
| 9 | 9.5 | 9.5 |
| Na + conc. (M) | 1.5 | 1 | 1 |
| Temperature | 35 | 37 | 37 |
| HRT (day) | 3.3 | 1 | 1 |
| Conversion rates (mmol) | 18 | 85 | 89.5 |
| Max. sulfide conc. (mmol) | 260 | 76 | 82 |
| Side products | Formate | Acetate | Acetate / formate / lactate |
| Weight (kg) | 60 | 50 | 37.5 | 5 |
| No prior CTX (mg) | 75 | 55 | 41.25 | 5.5 |
| Prior CTX (mg) | 90 | 60 | 45 | 6 |
| Perfusion (L) | | | | |

(a) a table sample with vertical layout

(b) a table sample with horizontal layout

(c) a table sample with matrix layout
different aggregation strategies are modeled by an order-invariant neural network to derive the relation among set elements. We adapt the aggregation method proposed by Qi et al. (2017) for point cloud representation in the point cloud classification task (classification of objects represented by points in 3D space) to be applicable to table representation in the table orientation classification task. This network applies a symmetric function on transformed elements in the set: It uses a multilayer perceptron (MLP) for transformation which is shared by all points and a max pooling as a symmetric function to aggregate point representations. In Sect. 5.1, we showed that the maximum operation modeled here by a max pooling layer acquires the highest performance compared to other order-invariant networks (see Sect. 2).

We use this approach on two levels: (1) on the level of cells within columns or rows, and (2) on the level of columns or rows of a table. DeepTable first represents cells within columns or rows by a shared cell layer (the network is shared by all cells) to capture the relation among cells and then applies max pooling functions (one on rows and one on columns) to determine the best aggregated representation for each column and each row. Then it applies two shared neural networks, one on the row representation and one on the column representation to capture the relationship between rows or columns. In the following, we describe the shared cell layer, the two table representations (column-wise and row-wise representations) and the classifier used for classification.

### 3.3 Shared cell layer

The input table is modeled as a fixed-sized (truncated or padded) three dimensional array, consisting of \( N \) rows, \( M \) columns and \( T \) tokens in the cells. The content of each cell is given to a shared cell layer to capture the semantic meaning of cell contents as shown in Fig. 1. Since the network is shared by all cells, the network is capable of learning the relation among cells as well. A cell is a sequence of tokens. As these tokens are typically words (see Sect. 4.1), the order of tokens may carry semantic meaning. To model such semantic meaning, following Nishida et al. (2017), tokens of the cells are first represented by an embedding vector to map tokens into a lower-dimensional space capturing the frequencies of co-occurring adjacent tokens. Then the embedding vectors are passed to a Bidirectional Long Short-Term Memory (LSTM) layer (Bi-LSTM) to extract the dependencies among tokens in a cell. Then a multi-layer perceptron (MLP) is used to map the output of the Bi-LSTM layer into a new vectorial space in order to extract both the non-linear relation among two LSTM representations, and the relation among cells as a single copy of the MLP is shared by all cells. The three layers of Embedding, Bi-LSTM and MLP on cells are described in the following.

**Embedding layer** Each cell \( c \) is originally represented by the sequence of tokens \( T_c = \{w_1, \ldots, w_T\} \). Each token \( w_t \) is represented by a binary vector with length \( |V| \), where \( V \) is the set of all tokens in the corpus, the dictionary. All the elements of the vector are set to zero except the index of token \( w_T \) in the dictionary which is set to one. An embedding layer maps each of these binary vectors into a lower dimensional continuous vector space \( e_t = W_e w_t \), where \( W_e \) is the mapping weight matrix and \( e_t \) is the representation of token \( w_t \) in the embedding space. The weight matrix \( W_e \) is
**Bi-LSTM layer**

The output of the embedding layer for each cell is a sequence of tokens in the embedding space. Each sequence is sent into a Bi-LSTM network to capture the semantic dependencies among tokens (which are typically words as shown in Sect. 4.1) in the sequence from both directions. The Bi-LSTM layer consists of two LSTM layers, one in forward direction and one in backward direction. The output of this layer is obtained by concatenating the last LSTM units’ outputs in both directions, for each cell, in a vector with the dimension $2 \times H$, where $H$ is the size of the LSTM output unit. The LSTM unit output in forward direction is specified in Eq. 1:

$$
\begin{align*}
    g_t &= tanh(W^g_e e_t + W^g_h h_{t-1} + b^g) \\
    f_t &= \sigma(W^f_e e_t + W^f_h h_{t-1} + b^f) \\
    i_t &= \sigma(W^i_e e_t + W^i_h h_{t-1} + b^i) \\
    o_t &= \sigma(W^o_e e_t + W^o_h h_{t-1} + b^o) \\
    s_t &= i_t \odot g_t + f_t \odot s_{t-1} \\
    h_t &= o_t \odot tanh(s_t)
\end{align*}
$$

Fig. 1 The network architecture of DeepTable. The example cell content “safe and reliable” is tokenized and represented by an embedding layer. This representation is given to a Bi-LSTM layer and then to an MLP layer as the cell representation. Then column/row-wise representations are applied, modeled by first applying a max pooling function on the cell representations and then a shared MLP layer on each column/row aggregated representation. Finally the representations are concatenated and fed to a softmax classifier for table orientation classification.

initialized with a pretrained word embedding model (see Sect. 4.3) and is retrained in our network.
where $W^e_g$, $W^f_e$, $W^i_e$, $W^o_e$ are weight matrices for token $e_t$ in the embedding space, $W^g_h$, $W^f_h$, $W^i_h$, $W^o_h$ are weight matrices for the previous LSTM unit’s output, and $b^g$, $b^f$, $b^i$, $b^o$ are biases of the model. $\sigma$ is a sigmoid function, $tanh$ stands for a hyperbolic tangent function and $\circ$ expresses element-wise multiplication.

**MLP layer on cells** The multi-layer perceptron (MLP) layer combines the outputs of two LSTM layers (forward and backward) to extract the non-linear relation among the two representations. Moreover, the shared MLP network extracts the relation among all cells within the table, as all cells share a single copy of the MLP unit. We define a non-linear function for the MLP layer which is made of two sequential dense layers with a rectified linear unit (ReLU) activation function (Glorot et al. 2011) as given in Eq. 2:

$$
\begin{align*}
    u'_c &= ReLU(W_{a1}(h_{Tc} \oplus h'_{Tc}) + b_{a1}) \\
    u_c &= ReLU(W_{a2}u'_c + b_{a2})
\end{align*}
$$

where $u_c$ is the representation of cell $c$, $h_{Tc}$ and $h'_{Tc}$ are the last output of the forward and backward LSTM layers, $W_{a1}$, $W_{a2}$, $b_{a1}$ and $b_{a2}$ are the model parameters and ReLU stands for a rectified linear unit. After each dense layer, we use a dropout layer (see Sect. 4.3) to avoid overfitting.

### 3.4 Two table representations

Tables are embodied by two representations: column-wise and row-wise representations as shown in Fig. 1. In both representations a max pooling function is applied over the cell representations of each column (row) to determine the best aggregated representations for columns (rows), then a shared MLP network is applied on max pooling outputs of rows (in a row-wise representation) or columns (in a column-wise representation). This extracts the relation among columns (rows) in column-wise (row-wise) representation, as the MLP layer in each configuration is shared by rows or columns. In the following, we explain the max pooling function and the shared MLP layers used by each table representation model.

**Max pooling layer** The max pooling layer is a symmetric function (Qi et al. 2017). This makes the model invariant to cell permutations. The max pooling layer aggregates information from cells within a column or a row. We define two max pooling functions, one on columns and one on rows. The max pooling function on columns is defined in Eq. 3:

$$
l_j = \max_{c \in C_j}(u_c)
$$

where $l_j$ is the aggregated representation of column $C_j$, $c$ is a cell in column $C_j$ and $u_c$ is the cell representation (see Sect. 3.3) and $\max$ is a vector max operator that takes cell vectors in a column as input and returns a new vector of the element-wise maximum (Qi et al. 2017). Similarly, the max pooling function on rows is defined as $r_i = \max_{c \in R_i}(u_c)$, where $r_i$ is the aggregated representation of row $R_i$, $c$ is a cell in row $R_i$ and $u_c$ is the cell representation.
Shared MLP layers We define two shared MLP layers: one on the max pooling output of columns (column-wise representation) and one on the max pooling output of rows (row-wise representation). The shared MLP layer designed for column-wise representation consists of two sequential dense layers as specified in Eq. 4:

$$
    u_{l_j}' = \text{ReLU}(W_{d1}l_j + b_{d1}) \\
    u_{l_j} = \text{ReLU}(W_{d2}u_{l_j}' + b_{d2})
$$

(4)

where $u_{l_j}$ is the representation of the column $C_j$, $W_{d1}$, $W_{d2}$, $b_{d1}$ and $b_{d2}$ are the model parameters and ReLU stands for a rectified linear unit. To avoid overfitting, after each dense layer, we use a dropout layer (see Sect. 4.3). A similar shared MLP layer is applied on the rows $r_i$ as row-wise representation $u_{r_i}$ to extract the relation among rows of a table.

3.5 Classification

The two representations are first flattened and then concatenated. The output of their concatenation is fed into a three-way softmax classifier with $l_1$ and $l_2$ regularizers to avoid overfitting.

4 Data and evaluation setup

In this section, we describe the data preparation procedure, the used baseline, the model parameter settings, and the evaluation metrics.

4.1 Data

We built a corpus to evaluate our approach, as to the best of our knowledge, there is no freely available annotated data for the three-class layout classification task or any of the previous table layout classification tasks stated in the previous work section. Tables in our corpus were extracted from scientific articles published in the PubMed Central (PMC) Open Access subset\(^1\) which currently consists of approximately 1.5 million full-text articles from several domains in the area of biomedicine and life sciences. Roughly 72% of these articles contain one or more tables (Milosevic et al. 2016) which are rather easy to extract as they have explicit mark-up in PMC’s XML format.

Each table in our corpus is automatically annotated as a member of one of the three classes of horizontal layout, vertical layout and matrix: A table has two types of cells, header cells and data cells. In a horizontal layout, all the cells in the first row are header cells and the remaining are data cells. In a vertical layout, all the cells in the first column are header cells and the remaining are data cells. The matrix layout has header cells in both the first column and the first row, and the other cells are data cells.

\(^1\) See https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/
We identified header cells based on a set of heuristics defined in (Isberner 2016) using HTML table tag information. Although the method has a high performance on header identification the technique is limited to a small subset of tables with explicit font-related formatting information. A cell is assumed to be a header cell if its text is written in bold, strong or italic types. Tables whose layout is not recognizable by the above rule were eliminated from our dataset. Furthermore, we only considered the subset of those tables where all rows have the same number of cells and all columns have the same number of cells to easily annotate tables using the above heuristics. However, our classification method is also applicable to tables with merged rows (columns) by substituting each merged cell with \( k \) cells in vertical (horizontal) direction with the same content where \( k \) is the span size of the merged cell (Zhang and Balog 2018).

In total, we annotated 12,161 out of 156,030 tables with above heuristics. To balance class distribution, our experiments use 5500 tables: 1835 column-wise (horizontal layout), 1834 row-wise (vertical layout) and 1834 matrices from the annotated tables. Note that HTML table tag information is only used by the heuristic rules for automatic annotation, but not by DeepTable. We also used 300 randomly selected tables from the remaining tables where annotation is not possible with the above heuristics for manual evaluation (see Sect. 4.4).

We determined a number of statistical properties of the tables in our corpus. We were particularly interested in characterizing the size of the tables and the numbers and types of tokens found in individual cells. The latter are especially relevant with respect to DeepTable’s architecture regarding semantics of cells in terms of embeddings and sequences of words. We thus determined (a) the distribution of the number of rows/columns in a table, (b) the number of tokens per cell (tokens are separated by space), (c) the number of tokens per cell consisting of digits (alphabetic characters) only, hence resembling numeric (word) tokens, and d) the distribution of those tokens containing a combination of digits and alphabetic characters. The corresponding histograms are shown in Fig. 2. The distributions of the numbers of cells in columns and rows are very similar, varying in a broad range from three to thirteen cells. Cell length, i.e., the number of tokens found in a cell, falls within a range from one to four tokens. This shows that the tables in our corpus are quite big in comparison to web tables. More than 80% of cells contain only a single token. The distribution of tokens with numeric characters shows that more than 70% of cells contain no purely numeric tokens and of the one or more tokens in those cells that do contain a numeric token, mostly only one token is numeric. Contrarily, the distribution of tokens with alphabetic characters, i.e., word tokens, shows that about 68% of all cells don’t contain any word token, around 20% of the cells contain one word token, and the remaining 12% contain more than one word token. Finally, regarding alphanumericic tokens, around 40% of the cells have only one and around 10% contain more than one token with both digits and alphabetic characters. Alltogether, the percentage of multi-token cells containing more than one word token is about 51% of all multi-token cells.

\(^2\) [http://webdatacommons.org/webtables/2015/relationalStatistics.html]
Fig. 2  Histograms showing the distribution of the number of (a) rows and (b) columns per table, (c) the numbers of tokens per cell, and the number of (d) numeric, (e) word and (f) alphanumeric tokens per cell in our corpus.

4.2 Baseline

We compare DeepTable with several baselines: (a) deep neural network classifiers which extract features automatically and (b) traditional classifiers which require feature engineering.

**Neural network classifiers** We compared DeepTable to a method recently proposed by Nishida et al. (2017) called TabNet. TabNet uses deep learning to classify tables into six table types: vertical and horizontal relational, vertical and horizontal entity, matrix, and others. We adapted it for classification of tables into the three main layout classes of horizontal, vertical and matrix, ignoring entity and relational table types. Although our method is general and can be applied to other table type classification tasks, we were not be able to compare DeepTable with the original TabNet model which classifies tables into six table types, as there is no freely available annotated corpus for their task.

The TabNet network has the following main differences compared to our DeepTable network: (1) TabNet represents each cell with an LSTM layer with attention mechanism instead of a Bi-LSTM with a shared MLP layer, (2) it uses several consecutive convolution layers with residual units for modeling the relation among rows and columns instead of the two order-invariant representation models we use on rows and columns. As a result, the TabNet model is not invariant to cell permutations, due to the use of convolutional layers. Similar to DeepTable, TabNet feeds the output of convolutions into a softmax layer for classification. We changed the original six-way softmax into a three-way softmax to classify tables into one of the classes of vertical layout, horizontal layout and matrix.

**Other baselines** We compared DeepTable with a baseline using heuristics for automatic annotation of web tables. The method measures the standard deviation of cell length for each row and column. Tables with smaller average standard deviation for columns than for rows are considered as horizontal tables, the ones with greater aver-
age standard deviation for columns than for rows are labeled as vertical, and otherwise as matrix.

We also compared DeepTable with two traditional supervised machine learning classifiers, (1) Support-Vector Machine (SVM) (Cortes and Vapnik 1995) and (2) Random Forest (RF) (Ho 1995). The classifiers where implemented using scikit-learn\(^4\) with default hyperparameters. We trained the classifiers for the table orientation classification task by adapting the features defined by Eberius et al. (2015) for the classification of web tables into different table type classifications (genuine table detection or table layout classification). Although these features are developed for representation of web tables, we used them here for representation of tables in scientific articles for two reasons: (a) to the best of our knowledge, there is no previous work on tables in articles using traditional classifiers with manual features, and (b) they achieved the highest performance in comparison to other traditional models and were the first-ranked competitor in the evaluation of TabNet (Nishida et al. 2017). We ignored the features using format information and the ones applicable to complex tables with merged cells and kept as features the number of rows and columns, cell length with maximum length, mean and standard deviation of cell length in columns and rows, and average number of cells with digits in rows and columns. The model is trained on our training set for the table layout classification task with three distinct labels.

4.3 Evaluation settings

We used the Keras tokenizer\(^5\) for cell content tokenization. We also implemented our network using Keras in Python with a Tensorflow backend. The network is trained using Keras’ Stochastic Gradient Descent (SGD) optimizer. We set the optimizer learning rate to 0.01 for DeepTable and all of its configurations, and 0.1 for TabNet to overcome both overfitting and underfitting in the networks (see Supplementary materials section). In both cases, we train the models for 100 epochs and choose the model with the lowest loss value on the validation set.

We set \(N = M = 9\), the number of rows and columns of our network and \(T = 4\), the number of tokens in a cell, as the tables in our corpus have, on average, 9 rows, 7 columns and 2 tokens per cell. Tables with smaller (larger) sizes are handled by zero padding (truncation). We set the size of the embedding layer to 200, as biomedical topics are generally representable in a space with 200 dimensions (see below). We set the size of the LSTM hidden layer to 50 and the output of the MLP layers to 100. We set the dropout rate to 0.1. To make the number of the parameters of DeepTable and TabNet comparable, we use the same size for the layers shared by TabNet and DeepTable, i.e., embedding and LSTM the layers present in both and keep the size of the remaining layers of TabNet the same as the original version.

We initialized the embedding layer in two ways: (1) with random vectors, and (2) with word embeddings trained by the algorithm proposed by Mikolov et al. (2013). The vectors have 200 dimensions and are trained on a combination of PubMed abstracts\(^6\).

\(^4\) https://scikit-learn.org/stable/
\(^5\) https://keras.io/preprocessing/text/
\(^6\) See https://www.ncbi.nlm.nih.gov/pubmed/
(nearly 23 million biomedical abstracts), PMC articles (nearly 700,000 full texts from the biomedical domain) and approximately four million English Wikipedia articles in the general domain. The model is available from Pyysalo et al. (2013). We used it for the initialization of both DeepTable and TabNet.

### 4.4 Evaluation metrics

We evaluate the performance of DeepTable in two different ways: (a) by measuring the performance on our heuristically annotated corpus (see Sect. 4.1), and (b) by manual evaluation on a set of randomly selected tables.

In the first setting, we partitioned our corpus into training and evaluation sets in a 4-fold cross validation: In each fold, 75% (80% for training and 20% for validation) of samples are used for training, and the remaining samples are kept for testing. We measure precision, recall and F1-score per table type, and accuracy over all table types. Given a specific table orientation type X, precision measures the ratio of tables correctly predicted as table orientation type X to all tables predicted as orientation type X. Recall measures the ratio of tables correctly predicted orientation types X to all tables annotated as orientation type X. F1-score is the harmonic mean of the precision and the recall values. Accuracy of predictions is the number of correctly predicted labels divided by the total number of predictions. As scores were obtained from 4-fold cross-validation, we report the mean values and standard deviations. We also measure the execution time of DeepTable and best performing baselines and provide confusion matrices of each method for error analysis by counting the number of tables classified correctly or wrongly by classifiers.

In the second setting, we perform the evaluation with human judges. We use the entire data for training (75% for training and the remaining for validation) and evaluate the performance of the model manually on a set of 300 randomly selected tables from the set of those tables for which the orientation is not recognizable by the above heuristics. In the human evaluation, we label a table as horizontal (vertical) layout, when the cells in a column (a row) are different instances of one category or different values of one attribute represented by a column (row) header. We label a table as matrix layout, when the relation of cells in both directions is meaningful (see Sect. 3.1). Per table type, we report precision, recall, F1-score and accuracy. We used McNemar’s test to find that our method (DeepTable) significantly outperforms the baselines.

### 5 Results

We measured F1-score and accuracy for DeepTable with different configurations. Then we compared the performance of DeepTable with different initializations. We compared DeepTable with traditional classifiers using handcrafted features and TabNet as the baselines, in terms of precision, recall, F1-scores and accuracy. We report for DeepTable and high score baselines the errors the methods produced, and their execution time. Moreover, we performed human evaluation and compared DeepTable...
Table 2  F1-scores (%) of DeepTable with its various variations per table type with their standard deviation
in 4-fold cross validation

| Model       | F1-score (SD) | H    | V    | M    | Average |
|-------------|---------------|------|------|------|---------|
| DeepTable-N | 75.73 (1.82)  | 74.61 (2.66) | 68.76 (2.11) | 73.03 (1.82) |
| DeepTable-C | 71.39 (1.18)  | 71.79 (1.26) | 57.78 (2.58) | 66.93 (1.20) |
| DeepTable-R | 70.56 (1.84)  | 70.02 (0.28) | 55.93 (3.74) | 65.50 (1.62) |
| DeepTable-NP| 78.08 (3.22)  | 77.36 (1.69) | 69.59 (3.24) | 75.01 (2.54) |
| DeepTable-SA| 78.12 (1.49)  | 77.64 (2.20) | **70.70** (2.91) | 75.49 (2.07) |
| DeepTable-AP| 76.01 (1.70)  | 75.52 (2.17) | 68.58 (3.14) | 73.37 (2.01) |
| DeepTable-BL| 78.78 (2.33)  | 78.19 (2.00) | 67.80 (2.62) | 74.92 (1.92) |
| DeepTable   | **79.49** (2.22) | **79.15** (1.72) | 69.89 (3.33) | **76.18** (2.40) |

Best value per method is in bold
V, vertical; H, horizontal; M, matrix; Average, Macro average over three layouts

with the best performing baselines in terms of precision, recall, F1-score and accuracy of the models trained on samples automatically annotated, and evaluated manually on 300 tables (see Sect. 4.4).

5.1 Comparison of different DeepTable configurations

We compared different variations of DeepTable to study the impact of (a) sequential neural networks on cell representation, (b) various order-invariant models on column- or row-wise representations, and (c) column and row-wise representations on orientation classification.

Effect of sequential models on cell representation We modified DeepTable by ignoring Bi-LSTM layer and instead passing the concatenation of all embedding outputs of a cell directly to MLP layer. This variation of DeepTable denoted DeepTable-BL. F1-scores per orientation class and averaged on three classes are written in Table 2. The results show that the use of Bi-LSTM layer can slightly improve F1-score, on average, by 1.26%. This small amount of improvement can be due to the fact that the majority of cells (more than 80% of cells) are made of only a single token in our corpus (see Sect. 4.1).

Comparison across various order-invariant networks We compared the performance of DeepTable with three configurations using different strategies for cell aggregation. One configuration ignores the max pooling layer to underline the effect of a pooling layer. In this configuration denoted DeepTable-NP all the cell representations in a column/row are concatenated and given to the MLP layer. In other configurations, we replace the max pooling layer in two manners: (a) with an average pooling layer and (b) with a self-attention layer. The average pooling layer considers the contribution of all cells equally (denoted DeepTable-AP) by averaging over all cell representations instead of taking the maximum in the max pooling layer and then passing the aggregated representation to the MLP layer. The other configuration denoted DeepTable-SA uses a self-attention layer to learn a new representation for each cell in a column/row as
Table 3  Precision (%), recall (%), and F1-score (%) of DeepTable with random initialization and pretrained word embeddings initialization with their standard deviation in 4-fold cross validation

| Model                      | Precision (SD) | Recall (SD) | F1-score (SD) |
|----------------------------|----------------|-------------|---------------|
| Random initialization      | 76.66 (1.04)   | 75.32 (1.13)| 74.97 (1.34)  |
| Pretrained embeddings      | **77.34 (2.13)** | **76.59 (2.28)** | **76.18 (2.40)** |

Best value per method is in bold

a weighted average of all cells in the column/row using a self-attention layer. Then the cell representations are concatenated and given to the MLP layer. Averaged F1-scores on all classes and per class are written in Table 2. The results show that the network using average pooling performs worse than the others, by dropping the average F1-score about 3%. The networks using no pooling layer and the one using self-attention layer are ranked second by nearly, on average, 2% F1-score improvement compared to the one taking the simple averaging. This can be due to learning weights for each cell, showing its contribution in a column/row. The F-scores obtained by DeepTable (using max pooling layers) and DeepTable-SA (using self-attention layers) are very close, with a slightly higher score achieved by DeepTable. This can be due to the fact that max pooling is capable of reducing the size of network parameters and consequently network overfitting.

Impact of table representation models To investigate the impact of the use of table representations on table orientation classification, we modified DeepTable in three different manners: (a) no table layout is used, i.e., the output of the shared cell layer is flattened and fed to the classifier (which is noted DeepTable-N), (b) only the column-wise representation is fed to the classifier (DeepTable-C) and (c) only the row-wise representation is fed to the classifier (DeepTable-R). We compare the F1-score of these models with that of DeepTable in Table 2. The scores indicate that the use of only column- or row-wise representations achieves the lowest performance, even lower than DeepTable-N without layout information. This might be due to the matrix layout being ignored when the model is using only a single representation, while DeepTable-N with no designated layout component can slightly learn matrix representation using the last layer. Concatenating row- and column-wise representations in DeepTable can improve the average performance by at least 3.15% and lead to a 3.76%, 4.54%, 1.13% improvement in the F1-scores of column-wise, row-wise, and matrix layouts, respectively. This means that the two designated representation components are more powerful than the end-to-end network (DeepTable-N) with no representation layers.

Effect of pretrained word embeddings We compared DeepTable network in two different initialization settings: (a) random initialization and (b) pretrained word embeddings (see Sect. 4.3). Precision, recall and F1-scores are listed in Table 3. The results indicate that the use of pretrained word embeddings improves F1-score (precision, recall) by 0.98% (0.68%, 1.27%) compared to random initialization. Moreover, we observe in both cases a low variation relative to its mean precision, recall and F1-scores over 4-fold cross validation. The average accuracy obtained using pretrained word embeddings is 76.60%, an improvement by 1.28% compared to the model with random initialization.
Table 4  Precision (%), recall (%), F1-score (%), and accuracy (%) of the heuristic model, traditional classifiers (SVM and RF) and deep neural network classifiers (TabNet and DeepTable) with their standard deviation in 4-fold cross validation.

| Model      | Precision (SD) | Recall (SD) | F1-score (SD) | Accuracy (SD) |
|------------|----------------|-------------|---------------|---------------|
| Heuristic  | 40.47 (4.98)   | 38.48 (0.93)| 32.59 (0.79)  | 38.49 (0.92)  |
| SVM        | 66.5 (4.56)    | 65.00 (3.41)| 64.91 (3.28)  | 64.75 (3.2)   |
| RF         | 76.55 (2.36)   | 74.96 (2.29)| 74.82 (2.34)  | 74.96 (2.28)  |
| TabNet     | 75.41 (2.75)   | 73.97 (2.59)| 73.66 (2.75)  | 73.98 (2.58)  |
| DeepTable  | 77.34 (2.13)   | 76.59 (2.28)| 76.18 (2.40)  | 76.59 (2.27)  |

Best value per method is in bold.

5.2 Comparison with baseline

We compared DeepTable with three types of baselines: (a) a heuristics used for annotation of web tables based on the average standard deviation of cell lengths in rows or columns,\(^8\) (b) two traditional classifiers, SVM and RF, using handcrafted features defined by Eberius et al. (2015) and (c) TabNet, a classifier using a deep neural network proposed by Nishida et al. (2017) (see Sect. 4.2). Precision, recall and F1-score for baselines and DeepTable are listed in Table 4. The scores show that DeepTable achieves higher F1-score (precision, recall) by at least 38.86% (31.84%, 33.29%) compared to the heuristic model, by at least 1.36% (0.79%, 1.63%) compared to traditional classifiers and by 2.52% (1.93%, 2.62%) compared to TabNet (deep learning classifier baseline). We also observe that DeepTable (and most other methods) has little variation relative to its mean F1-score (precision, recall) over 4-fold cross validation.

We compared the methods’ accuracies and observed that DeepTable obtained 76.59% accuracy, an improvement of 38.10%, 11.84%, 1.63% and 2.61% in comparison to the heuristic model, SVM, RF and TabNet, respectively. The obtained results confirm that the use of a network with permutation-invariant properties compared to a network using a 2-dimensional convolutional neural network or traditional classifiers with hand-crafted features improves overall accuracy in table layout classification.

5.3 Error and execution time analysis

Error analysis To analyze the errors produced by DeepTable, TabNet and RF (the best performing traditional model), we measured the confusion matrices by averaging over the confusion matrices of all 4-fold used in cross validation. Matrices are shown in Table 5, revealing that DeepTable’s classifications are more accurate than TabNet’s for all table layouts and those of the RF model for horizontal and matrix layouts.

We observed that for all classifiers the most difficult layout to classify is the matrix layout. TabNet and DeepTable generated similar false negatives for the matrix layout equally distributed over vertical and horizontal layout misclassifications. However, RF frequently misclassifies matrices as horizontal layouts than vertical layouts.

\(^8\) http://webdatacommons.org/webtables/index.html#results-2015
All three classifiers performed better in correctly identifying vertical and horizontal layouts. The patterns of errors produced by TabNet and DeepTable on horizontal and vertical layouts are similar, the only difference is in the number of their respective true positives where DeepTable obtained higher true positive rates compared to TabNet. However, RF frequently misclassified horizontal layouts as vertical layouts and vice versa.

**Execution and training time** We measured prediction execution time of DeepTable, TabNet and RF on an Intel(R) Xeon(R) CPU E5-2637 v4 @ 3.50GHz machine. The average execution time of DeepTable is 1.68 ms per table classification, while it is 1.25 ms for TabNet and 2.1 $\mu$s for RF. RF and TabNet are 800 and 1.3 times faster than DeepTable. We also measured the training time of the methods on the same machine. The training time for RF is 79 ms while it is 899.418 and 1123.748 s for TabNet and DeepTable after 25 epochs respectively.

**Classification performance versus training size** We trained DeepTable, TabNet and RF using different training sample sizes $T_s = \{1000, 1500, 2000, 2500, 3000, 3300\}$, and tested the trained models on the same test set with 1750 tables in all cases. Models’ accuracies are presented in Fig. 3. DeepTable and TabNet test accuracies rise by increasing training sample size while this remains nearly unchanged for RF. Moreover, the gap between training and test accuracies are very high on RF, which might be due to overfitting leading to a less generalizable model, while this gap is very small on DeepTable and TabNet. This shows that deep learning based methods are more robust and given more training samples, it is possible to improve their performances.

### Table 5

| Predicted label | V | H | M |
|-----------------|---|---|---|
| True label      | H | V | M |
| H               | 362.00 | 55.75 | 40.50 |
| V               | 56.25 | 389.75 | 12.50 |
| M               | 142.50 | 36.75 | 279.00 |

V, vertical; H, horizontal; M, matrix. The values are averaged over 4-fold.
Table 6  Precision (%), recall (%) and F1-scores (%) of RF, TabNet and DeepTable using human evaluation

| Model     | Precision | Recall | F1-score |
|-----------|-----------|--------|----------|
| RF        | 66.32     | 65.67  | 64.65    |
| TabNet    | 66.67     | 64.67  | 63.84    |
| DeepTable | 73.17     | 70.70  | 69.51    |

Best value per method is in bold

5.4 Human evaluation

Precision, recall and F1-scores for DeepTable, TabNet and RF (best performing traditional method) in our second evaluation setting based on human judgment are given in Table 6, showing that DeepTable in all cases outperforms TabNet and RF in this evaluation as well. DeepTable improves the F1-score (precision, recall) by at least 4.86% (6.85%, 5.03%) on average. Moreover, DeepTable obtained an accuracy of 70.00% which is 4.33% and 5.44% higher in comparison to RF and TabNet.

We performed a statistical significance test using McNemar’s test (Dietterich 1998). McNemar’s statistical hypothesis test provides a practical solution for comparing classifier models evaluated on a single test data set. We obtained a $p$-value of 0.01 and 0.05 ($\alpha = 0.05$) in the significance test between DeepTable and TabNet and DeepTable and the RF classifier, respectively, which shows a significant difference between the two models.

6 Conclusion

In this paper, we proposed DeepTable, a table orientation classification approach based on deep learning. Our model is invariant to the permutation of cells in columns or rows and uses two types of table layout representations. Moreover, our model does not require feature engineering which makes it applicable to a variety of tables in various languages or topics independent of formatting tags.

We measured the performance of the model on a corpus of tables extracted from articles in the biomedical domain. We showed that DeepTable achieves a higher precision, recall and F1-score compared to traditional methods and a baseline which is not invariant to cell permutations. We showed that the combination of two representations outperforms configurations using no or only one type of layout information. Statistical significance tests also reveal that DeepTable significantly outperforms the previous state-of-the-art and traditional baselines on our corpus.

In future work, we intend to accelerate DeepTable execution time to make it applicable for large scale table orientation classification. Moreover, we plan to extend the applicability of DeepTable to other table types, including the distinction between relational or entity type tables, to tables with complex layouts like tables with grouped header cells or data cells, and to tables from other sources like web tables. Training neural networks for such complex tasks with larger numbers of parameters requires more training samples and we intend to build a larger gold standard corpus for this task. Moreover, we plan to investigate the performance of DeepTable in different applica-
tions for management of tabular data such as table search, table summarization, table auto-completion or table fusion.

Acknowledgements The authors are grateful to the Deutsche Forschungsgemeinschaft (DFG) for its financial support through LE 1428/7-1 and STA 1471/1-1 projects.

Author contributions The experiments are conceived and designed by MH and UL. The experiments are performed by MH. The results are analyzed by MH, JS and UL. The paper is written by MH, JS and UL. All authors read and approved the final manuscript.

Funding Open Access funding provided by Projekt DEAL.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Agassi S, Ziv U, Shulman H (2004) Auto completion of relationships between objects in a data model. US Patent 6775674 B1
Bhagavatula CS, Noraset T, Downey D (2013) Methods for exploring and mining tables on Wikipedia. In: Proceedings of the ACM SIGKDD workshop on interactive data exploration and analytics, pp 18–26
Braunschweig K (2015) Recovering the semantics of tabular web data. Ph.D. thesis, Dresden University of Technology
Cafarella MJ, Halevy A, Wang DZ, Wu E, Zhang Y (2008a) Weatables: exploring the power of tables on the web. Proc VLDB Endow (PVLDB) 1(1):538–549
Cafarella MJ, Halevy AY, Zhang Y, Wang DZ, Wu E (2008b) Uncovering the relational web. In: Proceedings of international workshop on the web and databases (WebDB)
Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297
Crestan E, Pantel P (2011) Web-scale table census and classification. In: Proceedings of the fourth ACM international conference on web search and data mining, pp 545–554
Dietterich TG (1998) Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput 10(7):1895–1923
Eberius J, Braunschweig K, Hentsch M, Thiele M, Ahmadov A, Lehner W (2015) Building the Dresden web table corpus: a classification approach. In: Proceedings of the second IEEE/ACM international conference on big data computing (BDC), pp 41–50
Ghasemi-Gol M, Szekely P (2018) Tabvec: table vectors for classification of web tables. CoRR abs/1802.06290
Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp 315–323
Gonzalez H, Halevy A, Jensen CS, Langen A, Madhavan J, Shapley R, Shen W (2010) Google fusion tables: data management, integration and collaboration in the cloud. In: Proceedings of the first ACM symposium on cloud computing, pp 175–180
DeepTable: a permutation invariant neural network

Ho TK (1995) Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition, vol 1. IEEE, pp 278–282

Isberner A (2016) Similarity search on tabular data. Diploma Thesis. Humboldt University of Berlin

Kanerva P (2009) Hyperdimensional computing: an introduction to computing in distributed representation with high-dimensional random vectors. Cognit Comput 1(2):139–159

Lee J, Lee Y, Kim J, Kosiorak A, Choi S, Teh YW (2019) Set transformer: a framework for attention-based permutation-invariant neural networks. In: International conference on machine learning, pp 3744–3753

Lehmberg O, Ritze D, Meusel R, Bizer C (2016) A large public corpus of web tables containing time and context metadata. In: Proceedings of the 25th international conference companion on world wide web, pp 75–76

Liu Y, Bai K, Mitra P, Giles CL (2007) Tableseer: automatic table metadata extraction and searching in digital libraries. In: Proceedings of the 7th ACM/IEEE-CS joint conference on digital libraries, pp 91–100

Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Proceedings of advances in neural information processing systems, pp 3111–3119

Milosevic N, Gregson C, Hernandez R, Nenadic G (2016) Extracting patient data from tables in clinical literature: case study on extraction of BMI, weight and number of patients. In: Proceedings of the 9th international joint conference on biomedical engineering systems and technologies, pp 223–22

Nishida K, Sadamitsu K, Higashinaka R, Matsuo Y (2017) Understanding the semantic structures of tables with a hybrid deep neural network architecture. In: Proceedings of the 31st AAAI conference on artificial intelligence, pp 168–174

Petrovski B, Aguado I, Hossmann A, Baeriswyl M, Musat C (2018) Embedding individual table columns for resilient SQL chatbots. In: Proceedings of the 2018 EMNLP workshop SCAI: the 2nd international workshop on search-oriented conversational AI, pp 67–73

Pinto D, Branstein M, Coleman R, Croft WB, King M, Li W, Wei X (2002) QuASM: a system for question answering using semi-structured data. In: Proceedings of the 2nd ACM/IEEE-CS joint conference on digital libraries, pp 46–55

Pyysalo S, Ginter F, Moen H, Salakoski T, Ananiadou S (2013) Distributional semantics resources for biomedical text processing. In: Proceedings of the 5th international symposium on languages in biology and medicine

Qi CR, Su H, Mo K, Guibas LJ (2017) Pointnet: deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 652–660

Ritze D, Lehmberg O, Bizer C (2015) Matching HTML tables to DBpedia. In: Proceedings of the 5th international conference on web intelligence, mining and semantics, pp 10:1–10:6

Vilain M, Gibson J, Wellner B, Quimby R (2006) Table classification: an application of machine learning to web-hosted financial documents. Technical report, MITRE

Wang Y, Hu J (2002) A machine learning based approach for table detection on the web. In: Proceedings of the 11th international conference on world wide web, pp 242–250

Xu X, Liu C, Song D (2017) SQLNET: Generating structured queries from natural language without reinforcement learning. CoRR abs/1711.04436

Yoshida M, Torisawa K, Tsuji J (2001) A method to integrate tables of the world wide web. In: Proceedings of the international workshop on web document analysis, pp 31–34

Zaheer M, Kottur S, Ravanbakhsh S, Poczos B, Salakhutdinov RR, Smola AJ (2017) Deep sets. In: Advances in neural information processing systems, pp 3391–3401

Zhang S, Balog K (2018) Ad hoc table retrieval using semantic similarity. In: Proceedings of the 2018 world wide web conference, international world wide web conferences steering committee, pp 1553–1562

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.