



Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Table2Vec: Neural Word and Entity Embeddings for Table Population and Retrieval

Master's Thesis in Computer Science

by

Li Deng

Supervisors:

Krisztian Balog

Shuo Zhang

Spring 2018

Abstract

Tables contain a significant amount of valuable knowledge in a structured form. In recent years, a growing body of studies related to tables has been conducted in different application domains. To the best of our knowledge, utilizing neural embeddings regarding table corpus is rather unexploited. In this thesis, our goal is to employ neural language modeling approaches to embed tabular data into vector spaces, which are leveraged and contributed to table-related tasks. Specifically, we consider different tabular data, such as sequences of words, table entities, core column entities, and heading labels in relational tables, for training word and entity embeddings.

These embeddings are utilized subsequently in three particular table-related tasks, i.e., row population, column population, and table retrieval, by incorporating them into existing retrieval models as additional semantic similarity signals. The main novel contribution of Table2Vec is a neural method for performing multiple table-related tasks developed specially on table corpus.

We further conduct an evaluation of table embeddings on the task level. The results show that Table2Vec can significantly and substantially improve upon the performance of state-of-the-art baselines. In the best case, Table2Vec outperforms the corresponding baseline by 40%.

Acknowledgements

I would like to thank my thesis supervisors Prof. Krisztian Balog and Shuo Zhang from the Department of Electrical Engineering and Computer Science at University of Stavanger. Whenever I ran into a trouble spot or had a question about my research or writing, they can always give me useful suggestions against my doubts and steer me to the right direction. Besides, they have credited me great trust and understanding in the process of my research. Without their passionate participation and input, the research work could not have been successfully conducted or even completed. And I am gratefully indebted to their very valuable support on the researching and writing of this thesis.

Contents

Acknowledgements	v
1 Introduction	1
1.1 Approach and Contributions	3
1.2 Outline	4
2 Overview of Neural Retrieval and Table-Related Applications	5
2.1 Fundamental Concepts of Retrieval	5
2.1.1 Text-Based Retrieval Tasks	5
2.1.2 Evaluation Metrics	7
2.1.3 Traditional Retrieval Models	8
2.2 Neural Language Modeling in IR	10
2.2.1 Neural Retrieval Models	11
2.2.2 Unsupervised Term Embeddings	12
2.3 Table-Related Retrieval Applications	13
2.3.1 Table Extension	13
2.3.2 Table Mining	17
2.3.3 Table Search	19
2.3.4 Neural Models in Table-Related Application	21
3 Training Table2Vec Embeddings	23
3.1 Neural Models for Training Embeddings	23
3.1.1 Continuous Bag-of-Words Model	24
3.1.2 Basic Skip-gram Model	24
3.1.3 Optimization	25
3.2 Content Extraction	28
3.2.1 A Brief Introduction of Tables	28
3.2.2 Four Variants	29
4 Utilizing Table2Vec Embeddings	33
4.1 Introduction	33
4.2 Row Population	34
4.2.1 Baselines	34
4.2.2 Using Table2Vec Embeddings	35
4.3 Column Population	36

4.3.1	Baseline	36
4.3.2	Using Table2Vec Embeddings	36
4.4	Table Retrieval	37
4.4.1	Overview	37
4.4.2	Baseline	37
4.4.3	Using Table2Vec Embeddings	38
5	Evaluation	41
5.1	Experimental Setup	41
5.1.1	Data	41
5.1.2	Constructing Groundtruth	42
5.2	Row Population	44
5.2.1	Experimental Results	44
5.2.2	Analysis	45
5.3	Column Population	47
5.3.1	Experimental Results	47
5.3.2	Analysis	49
5.4	Table Retrieval	49
5.4.1	Experimental Results	49
5.4.2	Analysis	50
6	Conclusion and Future Work	55
6.1	Conclusion	55
6.2	Future Work	57
A	Resources	59

Chapter 1

Introduction

In recent years, we have witnessed significant improvements regarding the performance of speech recognition, machine translation, and completion prediction tasks. These achievements were largely credited to the use of neural network models [1–3]. A growing body of research, which is about introducing these neural approaches to information retrieval (IR), has been conducted with the goal of advancing the state of the art or even achieving breakthrough performance as in these other fields. Vector representations are fundamental to retrieval models in which terms are usually the smallest unit of representation. Therefore, many retrieval models, both non-neural and neural ones, focus on learning good vector representations of terms.

One-hot vector representations Traditionally, people tend to represent data with high-dimension (up to millions of dimensions) vectors, for example, in audio and image data. In natural language modeling, terms, e.g., words and entities, are represented by one-hot vectors which are used to distinguish each term from every other term in the vocabulary. Each vector consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify the term, see Fig. 1.1(a). One-hot vector representations provide no useful information regarding the relationships that may exist between the individual terms, and furthermore lead to data sparsity. Consequently, more samples are needed to train a model.

Semantic vector representations Neural language modeling learns the semantics of terms from term sequences and embeds them to a continuous vector space. In semantic representations, terms are represented in a form of more dense vectors that can illustrate the relationships between different terms. Values are real numbers in these vectors. Figure 1.1(b) shows an example of semantic representations with high values shadowed. Given the term “dog”, we can see “cat” is more similar to it than “pineapple”, because

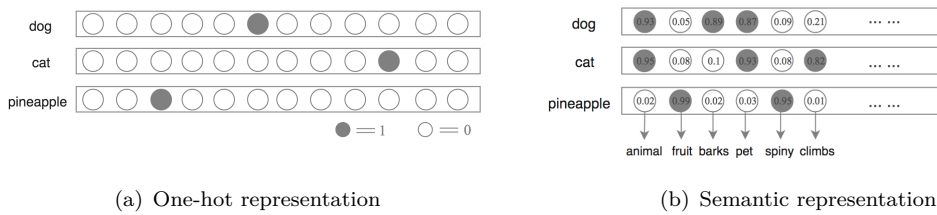


Figure 1.1: Illustration of two vector representations.

they both are animals and pets. Apparently, semantic representations can leverage the relationships between terms and resolve the issue of data sparsity efficiently.

Traditional retrieval models work in a way of extracting the useful information from a huge corpus of text documents. According to our observation, there are hundreds of millions of tables in web pages. These tables are much richer sources of structured knowledge than free-format text. Therefore, it is important to introduce IR techniques to table-related applications. Recently, a body of interesting research has developed around exploiting web tables for various tasks: (i) table search or table mining [4–6], (ii) table extension or completion [7–10], (iii) knowledge base (KB) construction [11, 12].

In this thesis, we focus on three particular table-related tasks: row population, column population, and table retrieval. These tasks consider *relational tables*, which describe a set of entities placed in a core column, along with their attributes in additional columns. Specifically, we use Wikipedia Tables corpus [13]. The corpus has been extracted from Wikipedia and consists of 1.6 million high-quality relational tables [5, 14].

Table population is the task of populating a given seed table with additional elements. Specifically, we address the *row population* and *column population* tasks proposed in [7]. The former aims to complement the core column of a relational table with additional entities, while the latter aims to complement the header row with additional column headings, see Fig. 1.2 as an illustration. *Table retrieval* is the task of returning a ranked list of tables for a keyword query, see Fig. 1.3.

Prior table-related work has considered embeddings, both pre-trained ones and task-specific ones. For example, Zhang and Balog [4] use pre-trained word and entity embeddings for table retrieval. Ghasemi-Gol and Szekely [15] develop table embeddings for table classification and Gentile et al. [16] train table embeddings for web table entity matching. To the best of our knowledge, no studies have been conducted on training table embeddings specifically for table population and retrieval tasks. To fill the gap, we propose Table2Vec, a novel approach that introduces neural language modeling to map different table elements into semantic vector spaces, which can benefit these tasks.

List of United States university campuses by enrollment/2009-10

University	Ranking	Location	Enrollment	+
Arizona State University	1	Tempe, Arizona	55,552	
The Ohio State University	2	Columbus, Ohio	55,014	
University of Central Florida	3	Orlando, Florida	53,537	
University of Minnesota	4	Minneapolis/Saint Paul, Minnesota	51,659	
+				

Add column ✕
 1. Population
 2. References
 3. Homepage

Add entity ✕
 1. University of Texas
 2. University of Florida
 3. Indiana University

Figure 1.2: Illustration of table population. Leftmost column and column heading labels are shadowed as grey. The user can add additional rows or columns by clicking the corresponding button, followed by auto-returning a list of suggestions.


1.1 Approach and Contributions

In this study, we train four variants of table embeddings by utilizing different table elements. Specifically, word embeddings (Table2VecW) consider all the terms within a table, and are leveraged for table retrieval. Two different entity embeddings are obtained by considering only core column entities (Table2VecE*) and all table entities (Table2VecE). The former is employed for the row population task, while the latter is employed in table retrieval. Heading embeddings (Table2VecH) are generated for the column population task by utilizing table heading labels. In summary, based on Wikipedia Table corpus, we have designed different embeddings for various tasks. The following research questions are addressed through the experiment:

- RQ1** Can Table2Vec improve table population performance against the state-of-the-art baselines?
- RQ2** Would different training datasets affect the embeddings, thus the retrieval results?
- RQ3** Which of the semantic representations performs better in table retrieval?

We further summarize the main contributions of this thesis as follows:

- We employ neural language modeling to train word and entity embeddings on Wikipedia Tables corpus by utilizing different table elements.
- We involve information retrieval techniques in various table-related tasks instead of traditional document retrieval.
- We develop new methods by employing the trained embeddings for table population and retrieval.

football clubs 

[Forbes' list of the most valuable football clubs - Wikipedia, the ...](https://en.wikipedia.org/wiki/Forbes'_list_of_the_most_valuable_football_clubs)
https://en.wikipedia.org/wiki/Forbes'_list_of_the_most_valuable_football_clubs

Team | Manchester United | Real Madrid | Arsenal | Liverpool |

Show less (26 rows / 8 columns total) - Export data

Rank	Team	Country	Value (\$M)	Debt as % of value
1	Manchester United	England	1,870	54
2	Real Madrid	Spain	1,353	23
3	Arsenal	England	1,200	107
4	Bayern Munich	Germany	1,110	0
5	Liverpool	England	1,010	59

[Home](#) | [Official Site](#) | [Chelsea Football Club](#)
<https://www.chelseafc.com/>

Team | Leicester City | Tottenham Hotspur | Arsenal | West Ham United |

Show less (6 rows / 5 columns total) - Export data

Pos	Team	P	GD	Pts
1	Leicester City	30	22	63
2	Tottenham Hotspur	30	29	58
3	Arsenal	29	16	52
4	Manchester City	29	21	51
5	West Ham United	29	12	49

Figure 1.3: Illustration of table retrieval. Given the keyword query, the system responds with a ranked list of tables.

- We perform evaluation on the embedding and task level, and provide further insights and analysis.

1.2 Outline

The content of this thesis is organized in the following manner: Chapter 2 presents an overview regarding the area of retrieval incorporating neural language modeling. It also covers the recent studies in table-related application domains. At the end of this chapter, we further introduce those table-based research that has employed neural language modeling. In Chap. 3, we describe the neural language models and their optimization methods in detail, followed by introducing our four variants of table embeddings as inputs to the chosen neural language model. We present new methods for table population and retrieval tasks in Chap. 4, by involving the Table2Vec embeddings that were derived before. In Chap. 5, we further report our experimental setup, results of table population and retrieval, and a detailed analysis for each individual task. Finally, a summary of the thesis and description about the future work are presented in Chap. 6.

Chapter 2

Overview of Neural Retrieval and Table-Related Applications

In this chapter, we first introduce some fundamental concepts of retrieval, such as retrieval tasks, evaluation metrics and traditional retrieval models, in Sect. 2.1. Section 2.2 gives an overview of neural retrieval. In Sect. 2.3, we present the related work for different table-based tasks.

2.1 Fundamental Concepts of Retrieval

Information retrieval is the process of entering a keyword into a system, and the system in turn responds with a list of ranked results from the data collection, see Fig. 2.1 as an illustration of the basic retrieval process. The retrieved results are ranked according to their relevancy to the query. An example of the real-world information retrieval system is the search engine, where search results may be passages of text or full text documents. In this section, we present an overview of text-based IR, such that we can refer to them in subsequent sections.

2.1.1 Text-Based Retrieval Tasks

There are two different application domains of text-based IR, i.e., ad hoc retrieval systems and question answering systems.

Ad Hoc Retrieval. Document retrieval is a classic problem in text-based IR. It has been not only reported as the main task in the Text Retrieval Conference [17], but also implemented by commercial search engines such as Google, Bing, and Firefox. Document

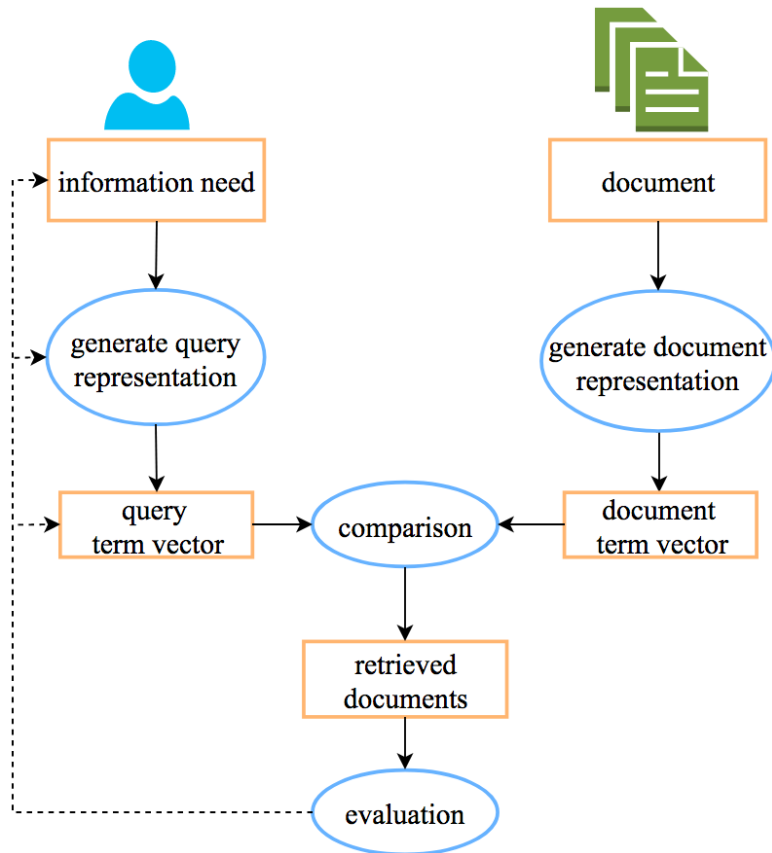


Figure 2.1: Basic information retrieval process.

retrieval is “ad hoc” because the number of possible queries is huge. Given a query that textually describes a user’s information need and a collection of textual documents, the goal of ad hoc retrieval is to find the relevant documents, and ideally, the top ranking ones are the documents satisfying the information need of the user.

Question Answering. Many research has been conducted in the area of question answering, which is the task of (i) ranking spans of text or passages. Voorhees and Harman [17] introduced the IR systems to retrieve spans of text in response to given questions, rather than documents, (ii) choosing between multiple choices. Based on deep neural networks, Hermann et al. [18] proposed a new methodology that can learn to read real documents and answer complex questions with minimal prior knowledge of language structure, or (iii) synthesizing textual answering by gathering evidence from one or multiple sources, Nguyen et al. [19] designed and developed of a new comprehensive real-world dataset of its kind in both quantity and quality, named Ms Marco, for the same task of reading comprehension and question answering.

2.1.2 Evaluation Metrics

Evaluation is key to build effective and efficient information retrieval systems, because the effectiveness, efficiency and cost of the IR process are related. IR systems respond users with a list ranked results, and the users are more likely to pay attention to these top-ranked ones. IR evaluation metrics, therefore, focus on rank-based comparisons of the retrieved result. These metrics are typically calculated at a rank position, k , and then averaged over all queries in the test set. In the following parts, we describe some of these standard metrics used most frequently in IR evaluations.

Precision and Recall. Precision and recall both compute the fraction of relevant documents retrieved for a query q . We refer A_q as the set of relevant documents and B_q as the set of retrieved documents from data corpus respectively, and formulate:

$$Precision = \frac{|A_q \cap B_q|}{|B_q|} .$$

$$Recall = \frac{|A_q \cap B_q|}{|A_q|} ,$$

Mean Average Precision. The average precision for a ranked list of results against query q is given by:

$$AvgP(q) = \frac{\sum_{k=1}^n P_q(k) \times rel_q(k)}{|A_q|} ,$$

where where k is the rank in the sequence of retrieved documents, n is the number of retrieved documents. $rel_q(k)$ is an indicator function equaling 1 if the item at rank k is a relevant document, 0 otherwise. $P_q(k)$ is the precision at cut-off k in the returned ranked list of results. The Mean Average Precision for a set of queries is the mean of the average precision scores for each query:

$$MAP = \frac{\sum_{q=1}^Q AvgP(q)}{|Q|} , \quad (2.1)$$

where Q is the total number of queries.

Mean Reciprocal Rank. Mean reciprocal rank is computed as the reciprocal rank of the first relevant document averaged over all queries:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^Q \frac{1}{rank_i} , \quad (2.2)$$

where $rank_i$ refers to the rank position of the first relevant document for the i -th query.

Normalized Discounted Cumulative Gain. Discounted cumulative gain (DCG) is a popular measure for evaluating web search and related tasks. It is based on two

assumptions, (i) highly relevant documents are more useful than marginally relevant document, (ii) the lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined. DCG uses a graded relevance judgment of documents from the result set to evaluate the gain, of a document based on its position in the result list. Gain is accumulated starting at the top of the ranking and may be reduced, or discounted, at lower ranks. DCG is the total gain accumulated at a particular rank position p :

$$DCG_p = rel_1 + \sum_{i=2}^p \frac{rel_i}{\log_2 i},$$

where rel_i is the graded relevance level of the document retrieved at rank i . An alternative formulation of DCG emphasizes on retrieving highly relevant documents, and given by:

$$DCG_p = \sum_{i=1}^p \frac{2^{rel_i} - 1}{\log(1 + i)}, \quad (2.3)$$

Since result set may vary in size among different queries or systems, we introduce the normalized version of DCG (NDCG) to compare performances. In NDCG, numbers are averaged across a set of queries at specific rank values, typically at rank 5, 10, 15, 20, e.g., DCG at rank 5 is 6.11 and at rank 10 is 7.28. Usually, DCG values are normalized by comparing the DCG at each rank with the DCG value for the perfect ranking, which makes averaging easier for queries with different numbers of relevant documents. Formally,

$$NDCG_p = \frac{DCG_p}{IDCG_p}, \quad (2.4)$$

where $IDCG_p$ represents the ideal DCG. $IDCG$ is computed the same way as Eq. (2.3), but assuming an ideal rank order for the documents up to position p .

2.1.3 Traditional Retrieval Models

In this section, we present a few traditional information retrieval models, which usually serve as the state-of-the-art baselines for comparison purpose against these modern models that involving neural methods.

The Importance of Terms. Intuitively, terms that appear more frequently in a document should get higher weights, e.g., the more often a document contains the term “phone”, the more likely that the document is “about” phones. Besides, terms that appear in many documents should get low weights, e.g., “a”, “the”, and “is”. There are two ways to capture the term importance mathematically: (i) term frequency (tf), which reflects the importance of a term in a document (or query); (ii) inverse document

frequency (idf), which reflects the importance of the term in the collection of documents. The more documents that a term occurs in, the less discriminating the term is between documents, consequently, the less useful for retrieval tasks. Formally,

$$idf_t = \log \frac{N}{n_t} ,$$

where N represents the total number of documents and n_t is the number of documents that contain term t . We further combine tf and idf weights to formulate:

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t ,$$

BM25. BM25 was created as the result of a series of experiments [20]. It is a popular and effective ranking algorithm. The reasoning behind BM25 is that good term weighting is based on three principles, i.e., inverse document frequency, term frequency, and document length normalization. Formally, BM25 is given by:

$$BM25(d, q) = \sum_{t \in q} \frac{f_{t,d} \cdot (1 + k_1)}{f_{t,d} + k_1(1 - b + b \frac{|d|}{avgdl})} \cdot idf_t , \quad (2.5)$$

where $b(b \in [0, 1])$ refers to the document length normalization. b equals to 0 and 1 represent no normalization at all and full length normalization respectively. k_1 works in a way of calibrating term frequency scaling. $k_1 = 0$ corresponds to a binary model, and large values of k_1 correspond to using raw term frequencies. Empirically, k_1 is set between 1.2 and 2.0, and a typical value is 1.2. BM25 combines the contributions from individual terms but ignores any phrasal or proximity signals between the occurrences of the different query terms in the document. BM25F, an extension of BM25, managed to incorporate multiple fields in the model, e.g., title, body, and anchor texts.

Language Model. Language model has been widely used in various real-world applications, e.g., speech recognition, machine translation, and completion prediction. It is based on the notion of probabilities and processes for generating text. In standard language modeling approach, we rank documents d according to their likelihood of being relevant given a query q and formulate:

$$P(d|q) = \frac{P(q|d) \cdot P(d)}{P(q)} \propto P(q|d) \cdot P(d) ,$$

where $P(d)$ is the probability of the document d being relevant to any query. $P(d|q)$ represents the query likelihood given by:

$$P(q|d) = \prod_{t \in q} P(t|\theta_d)^{f_{t,q}} ,$$

where $f_{t,d}$ is the number of times t appears in q . $P(t|\theta_d)$ is a multinomial probability distribution Smoothing parameter over the vocabulary of terms. Most formulations of language modelling based retrieval typically incorporate some form of smoothing [21] by sampling terms from both the document d and the full collection D . The two common smoothing methods are:

Jelinek-Mercer smoothing.

$$P(t|\theta_d) = (1 - \lambda)P(t|d) + \lambda P(t) ,$$

where λ is the smoothing parameter and same amount of smoothing is applied to all documents.

Dirichlet smoothing.

$$p(t|\theta_d) = \frac{f_{t,d} + \mu \cdot p(t)}{|d| + \mu}$$

where smoothing parameter is μ , and smoothing is inversely proportional to the document length.

Both BM25 and language modelling based approaches estimate document relevance according to the occurrences of only the query terms in the document. The position of these occurrences and the relationship with other terms in the document are not considered.

2.2 Neural Language Modeling in IR

In recent years, the use of neural language modeling has significantly benefited both the research and real-world applications. A growing body of research about incorporating these neural approaches to the field of information retrieval have been conducted, with the goal of advancing the state-of-the-art systems or even achieving performance improvements as in these other fields. According to Fig. 2.1, there are three key operations in the information retrieval process, i.e., (i) generate query representation, (ii) generate document representation, and (iii) comparison metric. Incorporating neural approaches to information retrieval is fundamentally using neural network models in these three key operations. Figure 2.2 shows different examples of utilizing neural language models in IR and we will discuss these in detail in the following sections.

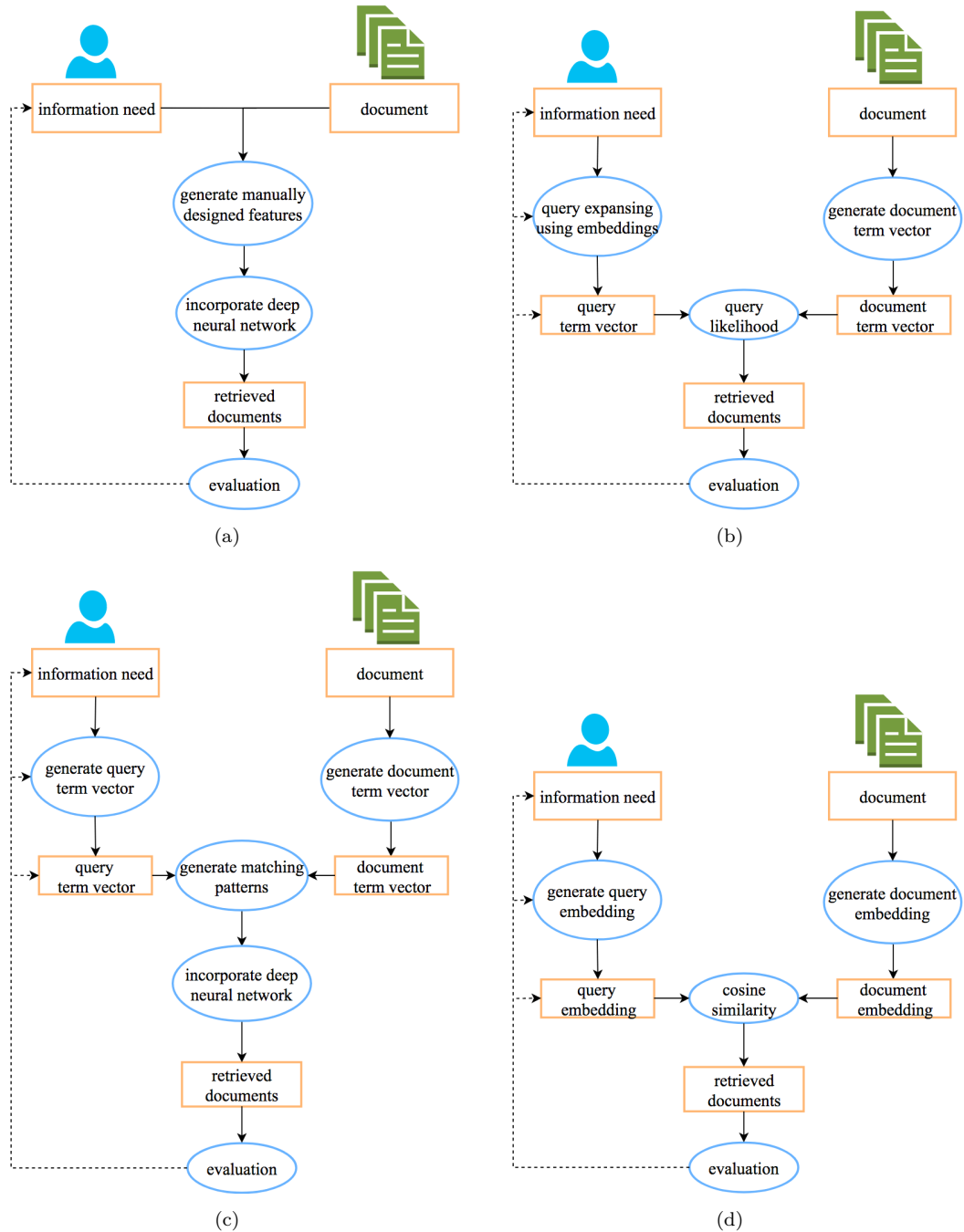


Figure 2.2: Illustration of four different neural language models employed in IR.

2.2.1 Neural Retrieval Models

Many retrieval problems are by nature ranking problems. Learning to rank is a task of automatically constructing a ranking model by using training data, such that the model can sort new objects according to their degrees of relevance [22]. Figure 2.2(a) illustrates a learning to rank neural retrieval process using manually designed features. A deep neural network is then introduced as a comparison metric to assess the relevance against

query-document joint representations. Traditionally, features of learning to rank models in information retrieval can be concluded into three categories: query-level features, document-level features and query-document features.

Neural methods can also be used to expand the query before employing the basic information retrieval model [23, 24], see Fig. 2.2(b) as an illustration. Kuzi et al. [25] have presented a suite of query expansion methods, which are based on word embeddings, to expand the query with terms that are semantically related to it as a whole or to its terms. In addition, they achieved meaningful improvement over the performance by integrating the query expansion methods with a pseudo-feedback-based query expansion approach.

In traditional retrieval models, terms have discrete or local representations, and the relevance of a document is determined by the exact matches of query terms in the body text. Unlike traditional learning to rank model, the model in Fig 2.2(c) depends less on manually generated features and introduces semantic features to derive a good matching pattern. Mitra et al. [26] have proposed a novel document ranking model composed of two separate deep neural network sub-models, one that matches using a local representation of text, and another that learns embeddings before matching, both of which have achieved significantly improvements over traditional retrieval baselines.

Many neural retrieval models depend on learning useful low-dimensional embeddings of query and document text, and then use them within traditional retrieval models or in conjunction with simple similarity metrics. Figure 2.2(d) shows the neural retrieval model that focus on learning effective representations of text by incorporating neural methods. We further note that this model employs neural approaches in all the core operations that we have introduced at the beginning of the section. As we discover, it can also learn the embeddings by optimizing directly for the retrieval tasks [27] or in an unsupervised setting [28].

2.2.2 Unsupervised Term Embeddings

So far we have presented a comparison regarding the different neural retrieval approaches. According to our observation, term embeddings are incorporated into these approaches for inexact matching. There are two different types of involvement against the term embeddings, (i) using embeddings to compute query-document relevance, (ii) use embeddings to generate suitable query expansion candidates from a global vocabulary and then perform information retrieval based on the expanded query. We will discuss both of them in the remainder of this section.

Query-Document Relevance Assessment. In this scenario, each individual terms from the vocabulary is represented as an embedding vector. The query and the document are subsequently represented as a dense vector respectively. The query and the document embeddings themselves can be compared using a variety of similarity measurement metrics, such as cosine similarity,

$$\text{sim}(q, d) = \text{cosine}(\vec{v}_q, \vec{v}_d) = \frac{\vec{v}_q \cdot \vec{v}_d}{\|\vec{v}_q\| \|\vec{v}_d\|},$$

Query Expansion. Rather than computing the query-document relevance directly, this approach first uses term embeddings to find good expansion candidates from a global vocabulary, and then retrieves documents using the expanded query. One of the ways to compute the relevance between query q and a term candidate t_{candi} is given by:

$$\text{score}(q, t_{candi}) = \frac{1}{|q|} \sum_{t_q \in q} \text{cos}(\vec{v}_q, \vec{v}_{t_{candi}}),$$

2.3 Table-Related Retrieval Applications

There are a vast amount of tables in web pages. These tables contain useful information, and has raised great interest in information retrieval field. Although web tables have proved to be useful sources, retrieving useful information from millions of tables on web is a problem on its own account. An increasing amount of research has been conducted to show the value of leveraging tabular data in various applications, including table extension, table mining, and table search. In the remainder of this section, we will introduce these applications and the related research work in detail.

2.3.1 Table Extension

Table extension is the task of populating a seed table with additional elements (e.g., rows and columns) or filling the missing values in a seed table (e.g., empty table cells), based on the corpus of tables. The task of row population is also related the problem of entity set completion.

Entity Set Completion

Entity set completion is to extend seed entities with additional entities that are returned by the retrieval systems (or algorithms) [8]. These entities are arranged in a descending order against their relevance with seed entities. An example system that does set

expansion is Google Sets, which carries out the set expansions task using propriety algorithms. The system was discontinued in 2011.

Wang and Cohen [29, 30, 31] have addressed the set completion problem by presenting the SEAL (Set Expander for Any Language) system, which expands entities automatically by incorporating multiple examples from the Web. SEAL is capable of handling various languages and there are two components of the system: (i) Extracting. SEAL constructed the character-level wrappers for each web page and extracting the context sequences that contains all seed entities. These context are subsequently applied on their source pages to extract candidate entities in addition to the given seed entities. (ii) Ranking. In the SEAL system, web pages, wrappers and candidate entities are modeled as nodes in the graph, and random walk techniques are used to rank candidates during iterations. Similar iterative phrases have been employed by He and Xin [32], which has proposed a method that completing the entity sets by deriving other entities belonging to the same concept set against seed entities. Multiple web data sources have been exploited to discover such relevant entities, including lists extracted from web pages and user queries from a web search engine. Instead of using random walk ranking, they proposed a new general framework based on iterative similarity aggregation.

Yakout et al. [9] presented the Infogather system to automate information gathering tasks. It comes with three core operations: (i) augmenting entities with attribute name. (ii) augmentation by example. Instead of providing the augmenting attribute name, the user provides the query table with some complete records as examples. (iii) discovering important attributes of a given set of entities. These operations are based on entity-attribute tables which are also referred as relational tables and 2-dimensional tables [33]. The key contribution of Infogather system is that it can obtain much higher precision and coverage by leveraging indirectly matching tables in addition to the directly matching ones. Specially, Infogather system addresses the problem of spuriously matched tables by developing a holistic matching framework based on topic sensitive pagerank. An augmentation framework that aggregates predictions from multiple matched tables is also employed by the system. In addition, a novel architecture for Infogather system that leverages preprocessing in MapReduce to achieve extremely fast response times at query time is proposed. The experiments are based on real-world datasets and 573M web tables from a crawl of Microsoft Bing search engine. The results show that the approach proposed has significantly improved the precision and coverage and achieved four orders of magnitude faster response times against the state-of-the-art baseline. The related tables that matched can be employed in both row and column population tasks.

Detecting related tables has been proved a powerful tool for extending seed tables with additional data and enables effective reuse of available public data. Das Sarma et al. [8]

have performed a task of detecting highly related tables on Wikipedia Tables corpus given an input table, by incorporating algorithms of *entity complement*. The key function of these algorithms is to determine that the entities in a candidate table are a coherent expansion of the entities in a seed table. Hence, the entity set completion is based on the coherency. Specifically in their paper, they have proposed two approaches to compute entity consistency score, (i) For each additional entity in table T_1 , compute its relatedness to each entity in T_2 , and then aggregate the pairwise entity relatedness; (ii) take the set of additional entities in T_1 as a whole, and directly compute its consistency with respect to T_2 .

An aspect-based framework named QBEEES that conducts searching similar entities based on one or more example entities has been proposed by Metzger et al. [34, 35, 36]. The core idea of this model is aspect-based similar entity retrieval. Given an RDF triple, (subject, predicate, object) and an entity e , we consider all the arcs that are incident with q in knowledge graph, thus we have the triples $(e, \text{predicate}, \text{object})$. An aspect of e is then given by the duple (predicate, object). There are three different aspect-based entity characterization models, (i) type aspect, where the set of all type aspects of an entity e reflects the set of all types. (ii) relational aspects, which captures the information which relations an entity is involved with. (iii) factual aspects, which means the rdf triple contains entity e presents a fact about it, e.g., (Paris, LocatedIn, Europe), It is a fact that “Paris” is located in “Europe”.

Instead of limiting the focus on entities themselves, Bron et al. [37] have employed additional textual descriptions of the candidate entities in addition to seed entities. They combine a text-based language model with a structure-based model derived from different aspects about the entity, i.e., type aspect and factual aspect. The text-based language model for an entity is constructed from terms appearing in facts about that entity and in descriptions of types and other entities connected to it. In the structure-based model, entities are represented by their facts with a uniform weight. Given a set of seed entities, types and facts that appear in many seed entities have higher weight against the retrieving results. Both [34, 37] have emphasized the use of entity aspects. The latter does not include a term component and in particular does not assume a textual description of the target entities as in [37]; Besides, the structure-based features in the aspects is more general in the latter paper, which also includes priors incorporating the entity importance as well as assigning different weights to different features.

Table Extension

We have already introduced related work of table extension that regarding set completion in §2.3.1. Table extension focus on columns is a task of extending the seed table with new columns, which are retrieved based on the given seed table. The Mannheim Search Join Engine by Lehmberg et al. [38] operates on a large amount of web tables and performs such table extension operations automatically. The Mannheim Search Join Engine searches the web table corpus for additional tabular data describing the entities contained in the seed table. The discovered data is used to rank the candidate tables, and relevant columns from the top- k ranked tables are then used to join with the seed table. Schema matching and data fusion techniques were employed in the whole process. At last, the user is provided with an extended table by the Mannheim Search Join Engine. The evaluation of the Mannheim Search Join Engine was operated on the table corpus derived multiple sources, which contains HTML tables, Linked Data and Microdata annotations in tabular form. The Mannheim Search Join Engine achieved very prominent results for the tasks of extending tables describing diverse entities, such as cities, companies, countries.

Cafarella et al. [5] have proposed an efficient relational data integration system called OCTOPUS, which consists of a sequences of operators, i.e., SEARCH, CONTEXT, EXTEND. Specifically, the user starts with the *Search* operator, which returns a cluster of relevant and related tables, he then choose two extracted relational tables and uses *Context* operator on both tables and modifies them to contain additional columns, using data derived from the source Web page embeds these two tables. At last *Extend* operator enables the user to add more columns to a table by performing a join with the other one. There are two different underlying algorithms developed for EXTEND operation in OCTOPUS, i.e., *JoinTest* and *MultiJoin*. The former looks for an extracted web table that is “about” the topic and has a column that can join with the indicated join column and it relies on a combination of Web search and key-matching to perform schema matching. The latter attempts to join each tuple in the source table with a potentially different table and addresses the problem of schema matching via a column-matching clustering algorithm.

Zhang and Balog [7] have proposed a smart Assistance that helps the user to extend the tables with additional rows and column labels, concentrating on a particular family of web tables, i.e., these that have an entity focus in their leftmost column. Specifically, they have proposed two specific tasks for providing intelligent assistance with tables: row population and column population. The former is the task of generating a ranked list of entities to be added to the leftmost column of a given seed table, while the latter is about generating a ranked list of column labels to be added to the column headings

of a given seed table. They have developed generative probabilistic methods for both tasks and enhanced the performance by combining the approaches from the literature with their novel components. This is the most related research to this thesis. Instead of using probabilistic methods for measuring relevance, we incorporated neural methods to deriving term embeddings for row and column population tasks in this thesis.

Data completeness is one of the most important indicators of data quality. Data completion is an essential premise for many subsequent data-related work. The completeness of tabular data is particularly critical because tables are highly organized and each cell represent different content against the table topic. Another type of table extension is the task of filling the empty cells in a seed table. Traditionally, statistical techniques are employed for filling the missing values. Ahmadov et al. [10] proposed a hybrid data imputation approach based on external data source such as relational web tables. This approach takes the characteristics of the incomplete dataset into account, for the purpose to look up missing values, or using a statistical approach such as regression analysis, or combine both approaches to find the most qualified data. Two keyword subqueries were introduced based on the input table to search entities and attributes separately.

2.3.2 Table Mining

Tables contains a vast amount of structural information that can be potentially useful for many application areas, e.g., knowledge base extension or completion. Table is a not only fundamental problem on its own, but also broadly used as a core component in other tasks, such as table extension, many existing table extension approaches using table mining as the premise of their experiment [2, 5, 8]. Recently, a growing body of research has been carried out in the area of table mining [11, 39–46].

The database corpus that is contained within the raw HTML tables is particularly valuable, it consists of data from millions of websites and a vast amount of topics. Cafarella et al. [14] pioneered in the area of extracting and leveraging the relational information embedded in HTML tables and proposed the WEBTABLES system. In the process, 14.1 billion HTML tables are extracted from Google’s web crawl and 154M that contain high-quality relational data are mined for later usage. The WEBTABLES system mines the good relations by combining hand-written detectors and statistically-trained classifiers, and uses a human marked test set to evaluate the mining performance. They further recovered the relations of some tables which are valuable for a knowledge base construction.

A neural network system being trained in an end-2-end fashion for natural language questions-answering on knowledge base tables, Neural Enquirer, has been introduced

by Yin et al. [47]. Given examples of queries and answers, the system can learn to understand queries and execute them on a knowledge base table in a supervised manner. Specifically, given a natural language query q and a knowledge base table T , Neural Enquirer executes q against T based on their embeddings and yields a ranked list of answers. The different embeddings are generated from a *query encoder* and *table encoder* for q and T separately, which are then as input to a series of neural network *executor*. An executor yields intermediate execution results, referred to as *annotations*, which are saved in the external memory of the executor. A query is executed sequentially through a stack of *executors*, and only the last one outputs the probability of an entry in T being the answer. Such a cascaded system enables the model to answer more complex queries.

Data in tables can be efficiently leveraged to enrich existing knowledge bases such as DBpedia [48], Freebase [49], YAGO [50]. KB is typically a large, directed graph that utilizes RDF triples to represent the relations between different nodes. Knowledge bases have information of a vast number of open domain entities and have been widely used against entity retrieval area. So far, many approaches have been proposed to enlarge the population of entities in a KB. Although the size of KBs keeps growing along with these efforts, we still have a limited coverage of entities against the number of real-world entities. The current Web contains a huge amount of tables, among which millions of tables contain high-quality relational data. Of these high qualified tables, there are many entity-attribute tables that contain information of some entities of the same type. An instance-based schema mapping solution is employed in Zhang et al. [12] to find the effective mapping between an entity-attribute table and a knowledge base via some matched data examples. Besides they also proposed efficient approaches for finding the matched data examples as well as the overall mapping of a table and a KB.

The web contains vast sources of structured data, such as HTML tables and spread sheets, both of these can be used for a knowledge base augmentation. The semantics of these structured data are usually ambiguous, presenting us from extracting triples against the web tables. Sekhavat et al. [11] have provided a probabilistic approach for extending an existing knowledge base (YAGO) with facts from tabular data by leveraging a web text corpus and natural language patterns associated with relations in the knowledge base. Prior approaches use mainly natural language understanding to determine whether two entities are related, [11] assumed all entities in the same row of a table are related by construction, and also labeled pairs of columns in the table with relations coming from an established knowledge base. Similar research has been described in other paper as well, e.g., Wang et al. [51] have described an approach for building a comprehensive knowledge base using linguistic patterns.

2.3.3 Table Search

Traditional retrieval model is particularly targeting the document retrieval problems which is by nature text-based retrieval problems. Tables are highly structured as well as contain massive information, hence they are of huge value. It is potentially for a user in need of structured data or information to find a table from database that fulfills the needs. The existing text-based retrieval model performs poorly in terms of structural data like tables. Table search is a fundamental problem on its own, as well as often used as the first step in other tasks [9, 38, 52]. Numerous studies have been developed around table search.

Cafarella et al. [14] proposed an approach for the table search, which performs a keyword relation search over a corpus of relational tables. Specifically, given a keyword query, the underlying idea is utilizing the top-ranked results (documents or passages) returned by a web search engine, and then extract the top- k tables from those results. Table search is also important in integrating web data. The data integration model, OCTOPUS system in [5] introduced an SEARCH operator that takes a search-style keyword query and returns a set of relevance-ranked and similarity-clustered web tables. More specifically, The SEARCH operator takes as input an extracted set of relations S and a user's keyword query string q . It returns a sorted list of clusters of tables in S , ranked in a descending order against their relevance to q . In this case, the set of relations S can be considered all the relations found on the Web. To summary, the table search approach here is as a modification of document search, by adding new signals to ranking documents, such as hits on the schema elements and left-hand columns. The weights of the new signals were determined by machine learning techniques.

Traditional retrieval is in a fashion of answering keyword query, and returns a list a ranked results. Pimplikar and Sarawagi [6] propose an ad hoc table search system that based on exploiting multiple sources of web structured data, which takes the keyword query with description about each column of the table as input and returns the user with a multi-column table. The table search can be viewed as a schema matching problem between the query column descriptions and a Web table. Schema matching [53, 54] has traditionally been applied for integrating databases that contain a consistent and clean set of tables and the main challenge is in managing the complex alignment between the large number of schema elements on each side. In contrast, [6] matches a few query columns to a large number of unlinked and noisy web tables.

The amount of web data is vast and keeps growing, raising the importance of techniques in terms of searching the Web. Web tables allows a user to get information in a structured form, which arising the attention of web table search. Vinyals and Le [1] have proposed

techniques to support a user in browsing and exploring a result for Web Table search. they focus on presenting the user with effective results, and target this problem in two approach, (i) table selection, It combines the relevance scores obtained from web table search with measures for similarity of the schema and the data tuples of web tables, thereby accounting for diversity in the presented result; (ii) table summerization, which selects a set of representative tuples of a table that induce little information loss with respect to non-selected tuples and preserve regularities of underlying data.

In recent years, people have looked at the problem of discovering semantics of tables in the context of web tables, where the goal is to have a better web table understanding to benefit web table search. Venetis et al. [55] described an approach for recovering the semantics of web tables, by leveraging the text on the web. Specifically, table search was aided by these annotations on the table gnerated in the semantic recovering process. Given a query in a form of combining class name and property, i.e., $q = (C, P)$, it consider tables in the corpus that have the class label C in the top- k class labels, and then performs ranking on these tables based on a weighted sum of the following signals collected through different table elements: occurrences of the property P on the tokens of the schema row, occurrences of P on the assigned binary relations of the table, page rank, incoming anchor text, number of rows and tokens found in the body of table and the surrounding text. The weights were derived by training on a set of examples.

In a most recent trend, neural language models are introduced for table-related tasks to catch semantics, such as table search, and achieved significant success. Zhang and Balog [4] propose an ad hoc seach by leveraging the semantics similarity between table-query pairs. Given a keyword query, the ranking of the results is established based on different degrees of semantic relevancy. Instead of using traditional lexical matching, they represent both queries and tables in some semantic (vector) space, and measuring the similarity of those vector representations. They have introduced various semantic representations to complete this mission, and focus on representing single terms (e.g., words, entities) rather than table and query themselves: (i) *bag-of-concept*, two different semantic representations are generated by leveraging entities and categories form DBpedia respectively. (ii) *embeddings*, two pre-trained embeddings model are introduced for the table search task, i.e., word embedding [56] derived on Google News Data, and graph embedding [57] trained on DBpedia. This is the work most related to this thesis, in contrast, instead of using pre-trained models, we train embeddings based on WikiTables specifically for table search task.

2.3.4 Neural Models in Table-Related Application

As we mentioned above, web tables have proved to be useful sources in information retrieval and knowledge extraction. Leveraging data within these tables is difficult because of the wide variety of structures, formats and data encoded in these tables. Neural language models have been developed as an alternative to represent raw texts as a bag of terms in the natural language processing field. In neural networks, terms are represented as vectors in the embedding space. To enhance the performance of retrieval, a growing body of studies that combine neural network and retrieval have been developed around. Some of these particularly focus on web tables.

Tables are in a form of structured data. Table type is essential premise for exploring the power of Web tables, and it is important to understand the semantic structures of tables in order to utilize them in various tasks. Nishida et al. [58] proposed a supervised deep learning method (TabNet) using a hybrid deep neural network architecture for table type classification, based on Hierarchical attention networks. More recently, Ghasemi-Gol and Szekely [59] present a neural method, TabVec, to resolve the problem of discovering data tables on the web and determining how data are organized within these tables on a large corpus of web data. The underlying idea of TabVec is (i) generating semantic vector representation for a tables cell, (ii) embedding tables themselves into a vector space by leveraging multiple context definition in a table, (iii) utilizing these table embeddings to support table type classification, such as categories of entity, relational, matrix, list, and non-data respectively. User annotations are used for examining these clusters, but not the training data. They further implement the evaluation of TabVec on four different real-world datasets, three of which are from unusual domains and the other is a sample from Common Crawl. In our thesis, Table2Vec also have employed semantic vector representation for tables, but it focus in terms of table terms (entity and words) embeddings instead of the whole table embeddings. As for the application, we use Table2Vec for table population and retrieval tasks rather than table classification. At last we have evaluated its performance over Wikipedia Tables corpus.

Many classic blocking methods are derived for data from relational databases with clearly defined schemas. Web tables are an interesting data source for many knowledge intensive tasks, and data from web tables more likely without an explicit schema, which arise the importance and challenge of partitioning the web tabular data. Gentile et al. [16] propose an effective approach for entity matching on web tables, which is the task of identifying records that refer to the same entity, by incorporating neural language model to block these data and subsequently reduce the comparison complexity. The general idea is to use word embedding which gives a lower vector dimensionality to generate a latent representation of table, instead of using classic bag-of word representation. And then

measure the similarity between tables by using cosine similarity. They further assume that the header rows, and attribute value relationship is known in order to create the context sentences for words within tables.

Based on popularity of Linked Open Data in data mining and information retrieval areas, RDF2Vec is proposed by Ristoski and Paulheim [57]. The underlying idea is to use neural language approaches for unsupervised feature extraction from sequences of entities. Converting RDF (Resource Description Framework) graphs to sequences are done by using (i) graph walks [60], and (ii) Weisfeiler-Lehman Subtree RDF Graph Kernels [61]. They derive the RDF graph embedding from two different knowledge graphs, i.e., DBpedia, which is extracted from structured data in Wikipedia, and Wikidata [62], which is a free collaborative knowledge graph operated by the Wikimedia foundation which also hosts various language editions of Wikipedia. The evaluation proved these models are capable of outperforming standard feature generation approaches in various tasks, such as machine-learning classification and regression, document similarity and entity relatedness, content-based recommender systems. Constructing embeddings from such huge KB graphs seems costly, but the embeddings can be reused on various tasks, e.g., Zhang and Balog [4] have employed the pre-trained graph embeddings based on DBpedia to serve an ad hoc table search task.

To the best of our knowledge, utilizing neural embedding methods regarding web tables is rather unexploited. Specifically, no work before has trained embeddings particularly for table-related tasks. Some research has employed the vectorization methods [4, 16, 59] by incorporating pre-trained embedding models. As we observed, these research only focus on one specific task, which gives us no proof whether the methods can benefit other table tasks. To fill in this gap, we focus on training neural embeddings from tables themselves, and then use these embeddings to exploit up to three different table tasks in this thesis.

Chapter 3

Training Table2Vec Embeddings

In this chapter, the content is arranged in the following manner. We introduce the neural language models for training embeddings and their optimization methods in Sect. 3.1, and then detail the four variants of table embeddings in Sect. 3.2.

3.1 Neural Models for Training Embeddings

Traditionally, many natural language processing systems represent terms in context with one-hot vectors, in which no semantic information been captured. This choice has its own advantages such as simplicity, robustness and high efficiency in terms of simple models. When it comes to a larger dataset or more complex models, we start to experience the defects of this method, such as computationally inefficient and low performance. With the progress of neural language modeling techniques in recent years, it has become possible and effective to train more complex models on much larger data set. One of the most successful concept is to use distributed representations of words [63] and they typically outperform the traditional models [64–66].

We base the training of our table embeddings on the *Word2Vec* [56] neural network model. It is proved to be a computationally efficient two-layer (with one hidden layer) Neural Language Model that learns the semantic meaning of terms from raw sequences and projects those terms to a vector space where similar terms are close to each other. There are two predictive models for *Word2Vec*, i.e., continuous bag-of-words(CBOW) and skip-gram. Since learning term representations is essentially unsupervised, methods are needed to “create” labels to train the model. Skip-gram and CBOW are two ways of creating the “task” for the neural network. Figure 3.1 shows both two architectures of *Word2Vec* and we further discuss both models in detail in the following subsections.

3.1.1 Continuous Bag-of-Words Model

CBOW uses continuous distributed representation of the context and works in a way that using context to predict the input. The input is comprised from all the surrounding terms in a given window area, e.g., the given window size in Fig. 3.1(a) is 3; Specifically, the vector in the projection layer is the average by all input vectors which are retrieved by the input weight matrix; Then further utilizing the weights from the output weight matrix to calculate a score for each term in the vocabulary, which represents the probability of the term being a target. Formally, given a sequence of training terms $t_1, t_2, t_3, \dots, t_n$, the objective of the CBOW model is to maximize the average log probability:

$$\frac{1}{n} \sum_{i=1}^n \log p(t_i | t_{i-c} \dots t_{i+c}) ,$$

where c refers to the window size and the probability $p(t_i | t_{i-c} \dots t_{i+c})$ is computed by:

$$p(t_i | t_{i-c} \dots t_{i+c}) = \frac{\exp(\vec{v}^\top \vec{v}'_{t_i})}{\sum_{t=1}^V \exp(\vec{v}^\top \vec{v}'_t)} ,$$

where V is the size of vocabulary, and \vec{v} is the average of all input vectors that represent the surrounding terms, formally,

$$\vec{v} = \frac{1}{2c} \sum_{-c \leq j \leq c, j \neq 0} \vec{v}_{i+j} ,$$

3.1.2 Basic Skip-gram Model

Skip-gram model does the inverse of CBOW by using a given input to predict the nearby terms, see Fig. 3.1(b). The input vector is retrieved by the input weight matrix; Then skip-gram further utilizes the weights from the output weight matrix to calculate a score for each term in the vocabulary, which represents its distance from the input term. More formally, given a sequence of training terms $t_1, t_2, t_3, \dots, t_n$, the objective of the skip-gram model is to maximize the average log probability:

$$\frac{1}{n} \sum_{i=1}^n \sum_{-c \leq j \leq c, j \neq 0} \log p(t_{i+j} | t_i) ,$$

where c is the size of training context, and the probability $p(t_{i+j} | t_i)$ is calculated using the following softmax function:

$$p(t_o | t_i) = \frac{\exp(\vec{v}'_{t_o}^\top \vec{v}_{t_i})}{\sum_{t=1}^V \exp(\vec{v}'_t^\top \vec{v}_{t_i})} ,$$

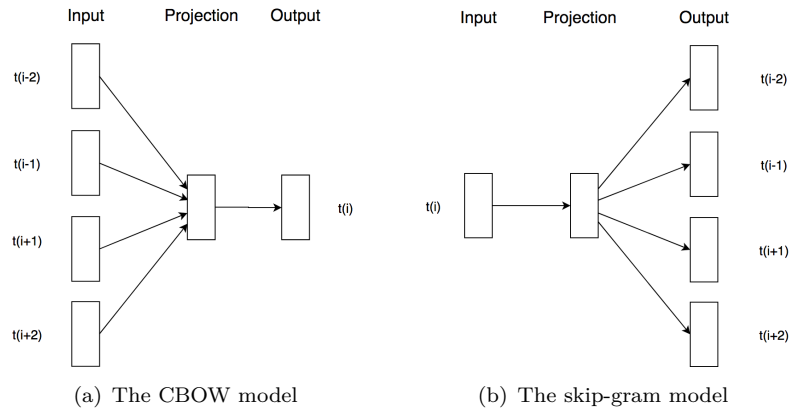


Figure 3.1: Word2vec neural network architecture.

where V is the size of vocabulary, and \vec{v}_{t_i} and \vec{v}'_{t_o} are the input and output vector representations of term t , respectively. Note that there are actually two representations of a term (apart from the one-hot vector), \vec{v}_t is the embedded vector for t as the center word, and \vec{v}'_t is the vector representation for t as the context word. Semantically similar terms share more similar vector representations, and the dot product between those vectors results in higher values, which means higher probabilities after softmax.

In our scenario, we consider terms to be words, entities, or heading labels in a table. According to [64], skip-gram model works well with small amount of the training data, represents well even rare terms or phrases. CBOW is several times faster to train than the skip-gram, and obtains slightly better accuracy for the frequent words; Which makes sense since with skip gram, you can create a lot more training instances from limited amount of data, instead of increasing the size of training corpus to deal with the data sparsity problem; As for CBOW, you will need more data for deriving the neural network, since you are conditioning on context, which can get exponentially huge. Hence, we employ the skip-gram model in this thesis for training our table embeddings because it can leverage limited data within a table and form more training examples compared to CBOW. The basic skip-gram model is impractical due to its large vocabulary size V results in large training sets and training time consumes. Thus we also employ optimization method to make the training of our models computationally more efficient.

3.1.3 Optimization

As we have mentioned above, the size of our term vocabulary V means that the skip-gram model has a vast amount of weights to be tuned, which means that training this model is going to be a considerably tough task. Recently, optimization methods for improving

the training performance of such model have been proposed and achieved very noticeable results and we will specifically discuss these methods in the following parts:

Sub-sampling of Frequent Terms

In the skip-gram model, infrequent terms usually matter more than frequent terms, e.g., “an”, “the”, and “of”, since frequent terms reveal much less useful information. Given a context, “many of the students like the library”, the co-occurrence of “students” and “library” benefits more for the skip-gram model than the co-occurrence of “the” and “students”. By sub-sampling frequent terms, not only the vocabulary size V becomes smaller but also the quality of the embeddings is improved. In sub-sampling, we grant each term a probability that formulated by:

$$p(t_i) = 1 - \sqrt{\frac{\tau}{f(t_i)}},$$

where t_i is a term from the vocabulary, τ is the threshold which usually empirically set as 10^{-5} , and $f(t_i)$ refers the term counts of t_i in the training corpus. and $p(t_i)$ represents the probability of keeping term t_i .

Hierarchical Softmax

Hierarchical softmax is very interesting from a computational point-of-view compared with the full softmax in which the probability of any one output depends on a number of model weights that is only logarithmic in the total number of outputs. More in detail, rather than evaluating V output nodes in the neural network to obtain the probability distribution, it is needed to evaluate only approximate number of $\log_2 V$ nodes.

$$O(V) \rightarrow O(\log_2 V),$$

which is significantly faster than the full softmax. The $\log_2 V$ nodes are selected by the binary tree, where leaves represent probabilities of terms; Each of the terms can be reached by a path from the root through the inner nodes, which represent probability mass along that way. Formally, the Hierarchical Softmax is given by:

$$p(t|t_i) = \prod_{j=1}^{L(t)-1} \sigma(\langle n(t, j+1) = ch(n(t, j)) \rangle \vec{v}_n^\top \vec{v}_{t_i}),$$

Where angled braces represent boolean checking if the case is true or false; $L(t)$ is the depth of the tree; $ch(n)$ is the child node of n , and $n(t, j)$ refers the j -th node on the

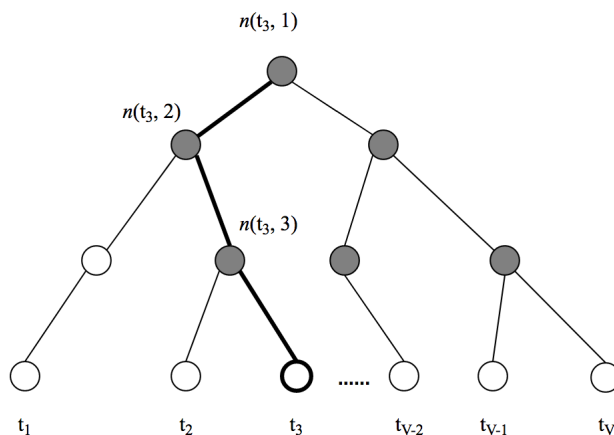


Figure 3.2: Illustration for Hierarchical Softmax.

path from the root to term t . And specifically, $\sigma(x)$ is formulated by:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Figure 3.2 shows an example of the Hierarchical Softmax. We can see $n(t_3, 1)$ is the root node, t_3 is the corresponding leaf node; it is obvious that we are performing three steps of computations, which is a sufficient decrease in the number of operations.

Negative Sampling

. We know that in the basic skip-gram model, all weights would be updated slightly by every single one of our training instances, which is extremely computationally inefficient. The negative sampling handles this issue in a way of having each training instance only modify a small percentage of the weights, instead of all of them. More in detail, with negative sampling, we are going to randomly select just a few “negative” terms to update their weights for.

The “negative samples” are chosen using a “unigram distribution”. Essentially, the probability for selecting a term as a negative sample is related to its frequency, with more frequent terms being more likely to be selected as negative samples. In the *Word2Vec*, you can see the equation* for this probability as follows,

$$P(t_i) = \frac{f(t_i)^{\frac{3}{4}}}{\sum_{j=0}^n (f(t_j)^{\frac{3}{4}})}$$

We can see each term is given a weight equal to its frequency (term count) raised to the $\frac{3}{4}$ power. The probability for selecting a term is just its weight divided by the sum of weights for all terms. The decision to choose the frequency powered by $\frac{3}{4}$ appears

to be empirical. And according to experiments by Mikolov et al. [56], the number of negative samples in the range 5 to 20 are useful for small training datasets, while for large datasets the number can be as small as 2 to 5.

To summary, Hierarchical Softmax and negative sampling do not seem to be exclusive; Specifically, we employ negative sampling for deriving all our table embeddings, and sub-sampling when we base the training on single words. It's also worth noting that by sub-sampling frequent terms and using negative sampling, we can not only reduce the compute complexity of the training process, but also improve the quality of their corresponding table embeddings.

3.2 Content Extraction

3.2.1 A Brief Introduction of Tables

Tables are highly structured. A table is a collection of related data arranged in a highly structured format within a database. It consists of columns, rows and cells. Table elements in a web table include (see also Fig. 3.3):

1. *pgTitle*, the main text that describes the web page which embeds the table.
2. *secondTitle*, the title of the web page section that contains the table.
3. *caption*, the title of the table which gives a brief description of content within the table cells, i.e., topic of the table.
4. *colHeadings*, a list column heading labels, usually corresponding to the first row.
5. *tableCells*, content of the table cells, include the heading row.

As we discovered, content from the same table is most likely related and shares some similar semantic information. Besides, content from different elements of the same table are usually differs from each other, because they focus on different aspects of the table topic, e.g., given a table that describes countries, then we know the content inside the table, *tableCells*, is related to the topic “country”; But each column is most likely about different attributes like names, populations, areas, etc., while each row is probably about information of one single country; Moreover terms in *tableCells* might be numbers, symbols, or even link to entities. To summarize, How to leverage different table elements and extract the rightful content for table raw representation that used for training table embeddings are problems themselves.

Article [Talk](#) [Read](#) [Edit](#) [View history](#)

2018 in film

From Wikipedia, the free encyclopedia

There are numerous films set to be released in 2018. While some films have announced release dates but have yet to begin filming, others are in production but do not yet have definite release dates.

Highest-grossing films [[edit](#)]

See also: *Lists of box office number-one films § 2018*

The top films released in 2018 by worldwide gross are as follows:^[f1]

Rank	Title	Distributor	Worldwide gross
1	<i>Avengers: Infinity War</i>	Disney	\$1,966,131,254
2	<i>Black Panther</i>		\$1,345,407,624
3	<i>Deadpool 2</i>	20th Century Fox	\$598,156,703
4	<i>Ready Player One</i>	Warner Bros.	\$580,360,031
5	<i>Operation Red Sea</i>	Huaxia Film	\$579,220,560

Avengers: Infinity War has grossed over \$1.9 billion worldwide, and is the 4th highest-grossing film of all time. *Black Panther* has grossed over \$1.3 billion worldwide, and is the 9th highest-grossing of all time.

Figure 3.3: An example of Wikipedia page.

3.2.2 Four Variants

In this thesis, we employ skip-gram of *Word2Vec*, a neural language model which has been proved semantically efficient to capture term features. Such model learns the semantics of terms from raw sequences and represent terms in a continuous vector space where semantically similar ones from the raw context are mapped to nearby points. We propose that tables can be represented as sequences of terms, (t_1, t_2, \dots, t_k) , because content from the same table is most likely topically coherent.

In terms of the table elements, we select four types of raw representations to represent the table and for training different neural embeddings. Table 3.1 lists the input (second column), which corresponds to the four types of embeddings (first column). Correspondingly, we train four different types of table embeddings by using different raw representations as input; These are well illustrated in Fig. 3.4. Note that all embeddings are trained using the same neural model, but they differ in (i) what constitutes as a term and (ii) which table elements are used for training the model. We further introduce the four variants in the following.

Table2VecW It is conventional to only use words for training neural networks. In this scenario, tables/queries are represented as sequences of individual words, which are qualified inputs of the neural language model. In a real-world web pages, many words are exclusive to the tables, and do not appear in the *tableCells*, hence, Our method Table2VecW takes the words appearing in all elements of a table into

Table 3.1: Table2Vec embeddings.

Method	Input	Semantic repr.
Table2VecW	all table data	word embeddings
Table2VecH	heading labels	heading embeddings
Table2VecE	all entities	entity embeddings
Table2VecE*	core column entities	entity embeddings

Region	Release Date	Label	Release Format
United Kingdom	22 September 2008	Super Records	DVD
Ireland	<i>pageTitle</i> : Radio:Active <i>secondTitle</i> : Release history <i>caption</i> : Release history	Records	DVD
Japan		mix	DVD
Argentina		18 May 2009	EMI Music
Singapore	12 June 2009	Warner Music	DVD
Spain	1 December 2009	EMI Music Spain	Digital Download

(a) Table2VecW

Region	Release Date	Label	Release Format
United Kingdom	22 September 2008	Super Records	DVD
Ireland	22 September 2008	Super Records	DVD
Japan	11 February 2009	Avex Trax	DVD
Argentina	18 May 2009	EMI Music	Digital Download
Singapore	12 June 2009	Warner Music	DVD
Spain	1 December 2009	EMI Music Spain	Digital Download

(b) Table2VecH

Region	Release Date	Label	Release Format
United_Kingdom	22 September 2008	Super Records	DVD
Ireland	22 September 2008	Super Records	DVD
Japan	11 February 2009	Avex_Trax	DVD
Argentina	18 May 2009	EMI	Music_Download
Singapore	12 June 2009	Warner_Music_Group	DVD
Spain	1 December 2009	EMI Music Spain	Music_Download

(c) Table2VecE

Region	Release Date	Label	Release Format
United_Kingdom	22 September 2008	Super Records	DVD
Ireland	22 September 2008	Super Records	DVD
Japan	11 February 2009	Avex_Trax	DVD
Argentina	18 May 2009	EMI	Music_Download
Singapore	12 June 2009	Warner_Music_Group	DVD
Spain	1 December 2009	EMI Music Spain	Music_Download

(d) Table2VecE*

Figure 3.4: Illustration of different Table2Vec embeddings.

consideration, so that we can capture the topical information of the table to the utmost extent. Specifically, it considers the page title, section title, table caption, table heading labels, and all table cells; see Fig. 3.4(a) as an illustration. Note only in this table embedding the sub-sampling of frequent words is considered.

Table2VecH Some words together holds a much different meaning than individual words, e.g., “New York” is a word pair that represents name of a city in America, but if we want to predict “York” from “New” with Table2VecW model, most likely the result is not pleasant. So it makes sense to treat “New York” as a word itself and has its own vector representation. Hence we propose Table2VecH. Instead of using single words, our method further leverage both the table structure and word-pairs by representing tables as sequences of table column labels extracted from column headings. Note that each column heading cell is treated as a single term, as is shown in the shadowed area in Fig. 3.4(b). The raw input is: “*Region Release_Date Label Release_format*” for Table2VecH in the illustration.

Table2VecE An entity is something that exists as itself, it need not be of material existence. Tables, especially relational tables, often contain entities which are semantically more meaningful than words. Moreover, the number of entities is much

less than that of the words, such that we can have a lighter training process. Hence we propose Table2VecE which takes sequences of entities as input, by extracting all entities that appear within *tableCells*; see the shadowed area in Fig. 3.4(c). According to the example, the input for Table2VecE is: “*United_kingdom DVD Ireland DVD ... Spain Music_Download*”

Table2VecE* Relational tables describe a set of entities as well as their attributes in the columns. These entities are placed in the *core column*. Table2VecE* only concern entities from first column of the table, as is shown in Fig. 3.4(d). The raw representation of the example in this scenario is: “*United_kingdom Ireland ... Singapore Spain*”.

So far, we have introduced all the four variants, and we will further discuss the application of those representations in the following chapter.

Chapter 4

Utilizing Table2Vec Embeddings

In this chapter, we present new methods for table population and retrieval tasks by involving the Table2Vec embeddings that were introduced in Sect. 3.2.

4.1 Introduction

For all tasks, we keep our focus on relational tables. Specifically, the table population task is considered in two flavors: row population and column population, and conducted based on entity-focused tables, which only have entities, more precisely unique entities as values in the left most column; It is also assumed that entities mentioned in the table are recognized and disambiguated by linking them to entries in a knowledge base [13]. Formally, let us define an $N \times M$ Table T :

$$T = \{c_{i,j}; 1 \leq i \leq N, 1 \leq j \leq M\} ,$$

where $c_{i,j}$ denotes the table cell. If a table is an entity-focused table, then:

$$c_{i1} = \{e_i; 1 < i \leq N\} ,$$

where (e_1, e_2, \dots, e_N) are entities different from each other.

We shall refer the input table for our table population tasks as a *seed table*, \mathcal{T} , which represents the table that has been edited by the user. Specifically, we assume the *seed table* already has the elements such as a *caption*, and some heading labels in the *colHeadings* and some entities in the left most column. Note that we do not employ the values in the other table cells in our task. In the *seed table* the set of entities from the

core column are referred as seed entities E , and the set of column labels are denoted as seed labels L .

4.2 Row Population

Row population is a task of returning a list of entities, based on their likelihood of being added to the core column of the seed table \mathcal{T} as the next row. The ranking is established based on the similarity score of a candidate entity e to the *seed table* entities E . And the one with the highest score is most likely to be the target added to a new row. In this task, we measure entity similarity by two approaches: using a knowledge base and using Table2Vec embeddings.

4.2.1 Baselines

We employ three probabilistic ranking methods from [7] as our baselines, which rank candidate entities according to $P_{kb}(e|E)$. Candidate entity selection method is also consistent with that in the source paper. Intuitively, candidates are identified from two sources: knowledge base and table corpus. Candidates from knowledge base are selected by the overlapping of types and categories properties between a candidate the seed entities, formally, we formulate:

$$score(e, E) = |\mathcal{P}_e \cap (\cup_{i=1}^n \mathcal{P}_{e'_i})| ,$$

where \mathcal{P}_e is a set of properties against entity e , and e' represents a seed entity. We identify candidates from table corpus according to how similar the *caption* is compared with that of the *seed table* based on BM25 retrieval algorithm, see Eq. (2.5). Tables contain seed entities also have a higher chance to be selected. We further describe our baseline methods in detail:

BL1 Entity similarity is measured based on the similarity of relations of e , obtained from RDF triples, and those of the seed tables entities E . Formally, given a subject-predicate-object triple, (s, p, o) , a relation for an entity e is then defined as:

$$\hat{e} = \{(p, o) \cup (s, p)$$

where (p, o) means in the triple s is an entity, (s, p) refers to o is an entity; Each entity is then represented as a set of relations, $(\hat{e}_1, \hat{e}_2, \dots, \hat{e}_n)$; Specifically, the relevance is measured based on the occurrence of each relation of candidate entity

e over the set of relations generated by *seed table* entities. In this case,

$$P_{kb}(e|E) = \sum_{r \in \hat{e}} \frac{\sum_{i=1}^n \langle r = \hat{e}_i \rangle}{|\Upsilon_E|},$$

where angled braces represent boolean checking if the case is true or false, which is true if r occurs in the representation of \hat{e}_i and is false otherwise. $|\Upsilon_E|$ denotes the total number of relations by all seed entities.

BL2 It uses the Wikipedia Link-based Measure [67] to estimate the semantic relatedness of entities based on their outgoing links (in the knowledge base).

$$P_{kb}(e|E) \propto sim(e, E) = 1 - \frac{\log(\max(|S_e|, |S_E|)) - \log(|S_e \cap S_E|)}{\log(|\Theta|) - \log(\min(|S_e|, |S_E|))}$$

where Θ denotes the sum of entities in the knowledge base. S_e and S_E are the sets of outgoing links from e and E respectively.

BL3 It relies on the Jaccard similarity between outgoing links of entities.

$$P_{kb}(e|E) \propto sim(e, E) = \frac{|S_e \cap S_E|}{|S_e \cup S_E|}$$

4.2.2 Using Table2Vec Embeddings

Recall that we have two entity embeddings, Table2VecE and Table2VecE*. The former is trained on all entities contained in the *tableCells*, while the latter considers only entities in the core column. Given that the row population task focuses on the core column, we employ the Table2VecE* embeddings here. Note that our candidates are selected by returning top- k ranked entities based on relevance between e' and e . We measure the similarity of each candidate entity e , against the seed entities $e' \in E$, using the cosine similarity of their respective embedding vectors:

$$sim(e, E) = \frac{1}{|E|} \sum_{e' \in E} sim(e, e') = \frac{1}{|E|} \sum_{e' \in E} \frac{\vec{v}_e \cdot \vec{v}_{e'}}{\|\vec{v}_e\| \|\vec{v}_{e'}\|}, \quad (4.1)$$

where $|E|$ is the size of seed entity set, and \vec{v}_e and $\vec{v}_{e'}$ are the embedding vectors of the candidate and seed entities, respectively.

We then combine the baseline similarity with the Table2Vec-based similarity using the following linear mixture:

$$P(e|E) = \alpha P_{kb}(e|E) + (1 - \alpha) P_{emb}(e|E),$$

where P_{kb} is the similarity measured using the knowledge base and P_{emb} is based on table embeddings, and equals to Eq. (4.1).

4.3 Column Population

Column population is the task of returning a ranked list of labels, l_1, l_2, \dots, l_k , given a seed table \mathcal{T} . The returned column heading labels are ranked based on their relevance to the seed labels L . Similarly to row population, we consider two label similarity measures. Note that we treat all the content in a table heading cell as a label.

4.3.1 Baseline

For comparison we employ the column labels candidate selection method by [7]. Candidates are identified from the tables that hold similarity with the *seed table* in seed column labels. Similarly we use BM25 algorithm to get a rank of tables from the table corpus. The baseline method, using a table corpus, is taken from [7]. First, relevant tables are retrieved from the table corpus. Then, the probability of a candidate label being relevant $P(l|L)$ is estimated based on the occurrences of that label in relevant tables. Formally, the probabilistic method against table relevance assessment is given by:

$$P(T|L) = \frac{|T_L \cap L|}{|L|}$$

where T means tables from corpus, T_L denotes the set of labels of a table from corpus. Based on the occurrence of column heading labels in the relevant table, and we define a label likelihood function as:

$$P(l|T) = \begin{cases} 1, & \text{if } l \text{ is in } T \\ 0, & \text{otherwise} \end{cases}$$

where l refers as a column label. Our ultimate goal is to assess the relevance between a label and seed labels which is calculated by:

$$P(l|L) = \sum_T P(l|T)P(T|L)$$

4.3.2 Using Table2Vec Embeddings

We utilize embeddings trained on table headings, Table2VecH, for column label relevance estimation. Similarly to row population, we employ cosine similarity based on the

Table 4.1: Summary of features used by table retrieval task. Note that the baseline features include query features, table features, and query-table features [4].

Baseline Features	
q_{len}	Sum of terms in the query q
idf_f	Sum of query idf scores in the field f
numRows	Number of table rows
numCols	Number of table columns
numEmptyCells	Number of empty tabular cells
PMI	The Attribute Correlation Statistics Database based schema coherency score
inLinks	Number of in-links to the page that contains the table
outLinks	Number of out-links from the page that contains the table
pageViews	Number of page views
tableImportance	The inverse of number of tables on the page
tablePageRatio	Ratio of table size to page size
#hitsLC	Total query term frequency in the leftmost column cells
#hitsSLC	Total query term frequency in second-to-leftmost column cells
#hitsB	Total query term frequency in the table data
qInPgTitle	Ratio of number of query terms occurrences in page title to total number of terms
qInTableTitle	Ratio of number of query terms occurrences in table title to total number of terms
yRank	Rank of the table's Wikipedia page given by Web search engine for the query
simInMLM	score given by Mixture Language Models between query and different table fields

embedding vectors of the candidate label l and seed labels $l' \in L$. Then, we combine these baseline estimate with the embedding-based similarity using:

$$P(l|L) = \alpha P_{kb}(l|L) + (1 - \alpha) P_{emb}(l|L) .$$

where the computing of $P_{emb}(l|L)$ follows analogously to Eq. 4.1.

4.4 Table Retrieval

4.4.1 Overview

Table retrieval is the task of returning a ranked list of tables in response to a keyword query q , based on their relevance to q . For this task, we employ a feature-based method as a baseline, which is referred to as the LTR method in [4]. This method considers a rich set of query, table, and query-table features. We utilize the word-based and entity-based table embeddings, Table2VecW and Table2VecE, to compute additional semantic matching features. Specifically, each type of embedding contributes four features, for each of the similarity methods in [4]. For performance comparison, we also employ pre-trained Graph2Vec [57] and Word2Vec embeddings [56].

4.4.2 Baseline

LTR employs all the baseline features listed in Table 3. We further explain some important features in detail, note that:

$$idf_f(q) = \sum_{t \in q} idf_f(t)$$

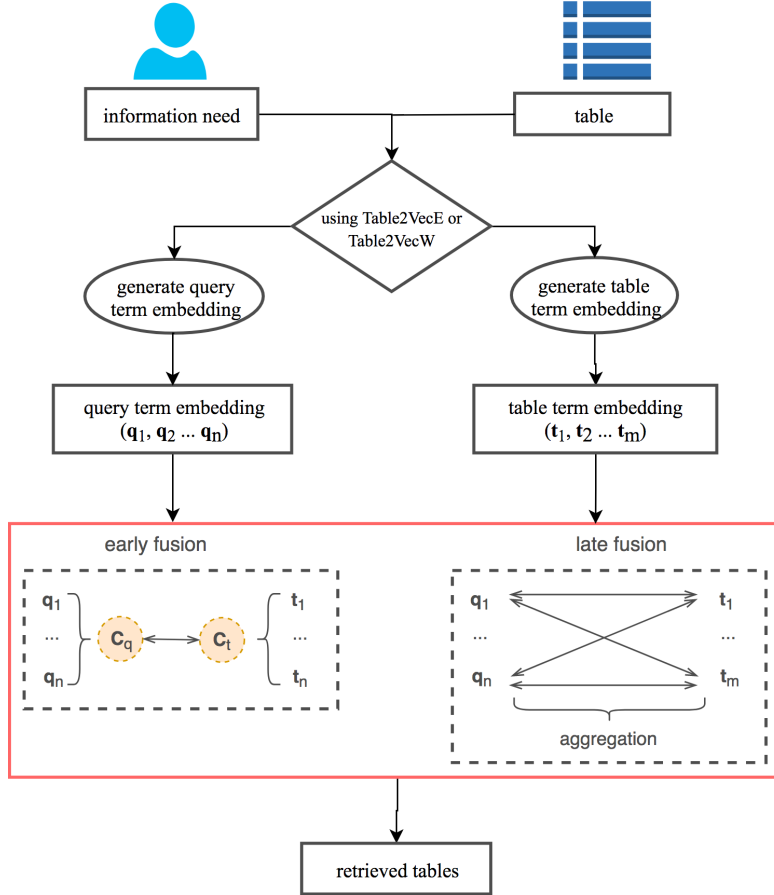


Figure 4.1: Computing query-table similarity using semantic representations.

where $idf_f(t)$ is the idf score of term t in field f which are listed in Table 1. Another important feature is point-wise mutual information (PMI) which is calculated by:

$$PMI(l_i, l_j) = \log(P(l_i, l_j) / (P(h_i)P(h_j)))$$

4.4.3 Using Table2Vec Embeddings

Now the remaining problem is how to compute the query-table relevance based on the different semantic representations separately. Given that both the table and query are vectors now, we introduce cosine similarity as the relevance assessment method. For comparison purpose, we employed the methods by [4]: *early fusion* and *late fusion*. Let us start with *early fusion* and some terms definition, \vec{C}_q refers to the centroid of query term vectors, similarly, \vec{C}_t denotes the centroid of table term vectors, and a term refers to an entity in entity embeddings, a word in word embeddings. More formally,

$$\vec{C}_q = \sum_n \vec{q}_i / n, \quad i = 1, 2, \dots, n$$

$$\vec{C}_t = \sum_m \vec{t}_j / m, \quad j = 1, 2, \dots, m$$

where n, m are the number of terms in query and table respectively, \vec{q}_i and \vec{t}_j are term vectors. Note that for word embedding, *tf-idf* weighting is employed to each term vectors in the above calculation. And the query-table relevance is given by:

$$sim(q, T) = cosine(\vec{C}_q, \vec{C}_t) = \frac{\vec{C}_q \cdot \vec{C}_t}{\|\vec{C}_q\| \|\vec{C}_t\|},$$

The *late fusion* method conducts pairwise cosine similarity between table terms and query terms first, and then aggregates those results. The query-table relevance is then given by an aggregation function, which consists of three aggregators: (i) maximum of $sim(\vec{q}_i, \vec{t}_j)$, (ii) sum of $sim(\vec{q}_i, \vec{t}_j)$ (iii) average of $sim(\vec{q}_i, \vec{t}_j)$. Note that *late fusion* has no exception for word embeddings. In this thesis, we combine the results computed by all four methods to yield the final similarity score, see Fig. 4.1 for an illustration.

We have further introduce Graph embeddings by [57] and use it on Wikipedia Tables corpus. And employ pre-trained Word2Vec embeddings with 300 dimensions, derived on Google News data, for performance comparison.

Chapter 5

Evaluation

In this chapter, we present our experimental setup, and the results for individual task followed by further analysis. We consider several standard retrieval evaluation metrics. For table population, we use Mean Average Precision (MAP, see Eq. (2.1)) as the main metric and Mean Reciprocal Rank (MRR, see Eq. (2.2)) as a supplementary metric for performance evaluation. Table retrieval performance is evaluated by Normalized Discounted Cumulative Gain (NDCG, see Eq. (2.4)) with a cut-off at 5, 10, 15 and 20. To test significance, we use a two-tailed paired t-test and write \circ to denote not significant, and \dagger/\ddagger to denote significance at the 0.05 and 0.01 levels, respectively.

5.1 Experimental Setup

5.1.1 Data

We use the Wikipedia Tables corpus [7], which contains 1.6 million high-quality relational tables in our experiment. The information provided by a single individual table can be categorized into two fields: textual fields like page title, second page title, table caption, table headings, table data, and statistical fields like number of rows, number of columns, number of numeric columns and number of header rows.

We use contents from only textual fields both for training the Table2Vec embeddings and for the retrieval experiments. And we have detailed the raw representations for training in Sect. 3.2.2. Additionally, for the word-based embedding, Table2VecW, we have filtered out empty strings, numbers, HTML tags, and stopwords from the raw text during training to obtain a better representation. For Table2VecH, we employ no normalization for the labels, i.e., “year(s),” “year:,” and “year” will be treated as different labels in our experiment. Specifically, Word2Vec is trained on Google News data with

Table 5.1: Statistics for Table2Vec embeddings. Neg is short for negative sampling.

Embedding	Total terms	Unique terms	Neg	Win_size
Table2VecW	200,157,990	1,829,874	25	5
Table2VecH	7,962,443	339,433	25	20
Table2VecE	24,863,683	2,159,467	25	50
Table2VecE*	5,367,837	1,285,708	25	50

Table 5.2: Statistics for Wikipedia Table corpus. tables* represents the tables that have more than 5 rows and 3 columns. Core column refers to the left most column.

Core column	Tables in total	Tables* in total
existing entities	726,913	212,923
60% entities	556,644	139,572
80% entities	483,665	119,166
100% entities	425,236	78,611
100% unique entities	376,213	53,354

300 dimension, and Graph2Vec is trained on DBpedia 2015-10, which contains 4,641,890 instances and 1,369 mapping-based properties, with 200 dimensions.

Table 5.1 shows the statistics of different Table2Vec embeddings trained by skip-gram 200-dimensional models. The decision to choose 200-dimensional model is both empirical and for obtaining as good as possible results quickly. It seems like that we are facing time constrained optimization problem in terms of choosing embedding size, as it can be expected that using higher dimensional word vectors will improve the accuracy. Mikolov et al. [68] has experimentally proved that after some point, adding more dimensions against the same training dataset provides diminishing improvements. Note DBpedia is used as our knowledge base, which is consistent with the original experiments.

Recall that for table population tasks, we employ a specific type of tables, entity-focused tables, which contains only unique entities in the left most column. We further constrain our experiment on the entity-focused tables that have more than five rows and three columns, so that we can obtain enough samples to simulate a real-world problem. Table 5.2 reports the statistics of Wikipedia Tables corpus. It is obvious that many tables have an entity focus. To be able to perform an automated evaluation without any human intervention, we employ entity focused tables that have at least six rows and four columns. According to Table 5.2, there are 53k tables in total that meet our requirements.

5.1.2 Constructing Groundtruth

The test inputs and ground truth assessments are obtained for the three tasks as follows:

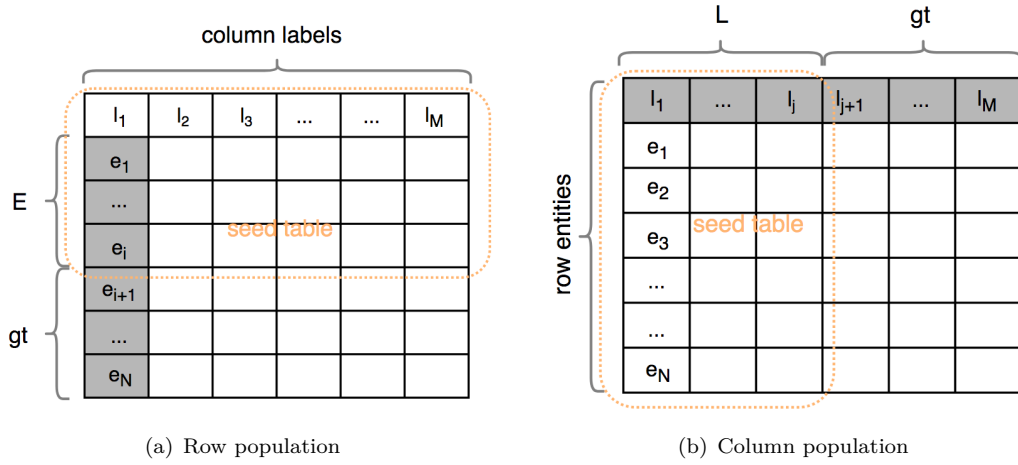


Figure 5.1: Illustration of evaluation methodology against table population. E and L represent the seed entities and seed labels respectively, and gt is the corresponding groundtruth.

- *Row population:* we use the test set from [7]. It contains 1000 relational tables ($N + 1$ rows and M columns), of which each table has at least six rows and four columns. Note that the first row is column heading labels, and we refer remaining N rows as data rows. For evaluation, we take entities from the first i data rows ($i \in [1..5]$) as seed entities, and the remaining ($N - i$) entities as ground truth. The test set contains 21,502 unique entities, see Fig 5.1(a).
- *Column population:* we use the test set from [7], consisting of 1000 relational tables. We take column heading labels from the first j columns ($j \in [1..3]$) as seed labels, and the rest labels as ground truth. There are a total of 7,216 unique column heading labels in this test set. Figure 5.1(b) illustrates the methodology of column population.
- *Table retrieval:* we use a set of 60 queries from 2 independent sources (30 queries from each, see Table 5.3) and corresponding ground truth relevance labels from [4], which amount to a total of 3,120 query-table pairs with a three-degree relevance assessment: (i) 0(non-relevant), which indicates the topic related to the table is unclear or different from the given key-word query topic. (ii) 1(relevant), which means some part of the table(cells and values) are related to the given query topic. (iii) 2(highly-relevant), which means large blocks or several values could be used directly from it when creating a new table on the query topic. Out of all the queries, 377 are labeled as highly relevant, 474 as relevant, and 2269 as non-relevant.

Table 5.3: illustration of two query sets.

No.	QuerySet 1	QuerySet 2
1	world interest rates table	football clubs city
2	2008 beijing olympics	healthy food cost
3	fast cars	capitals attractions
4	clothing sizes	diseases mortality
5	phases of the moon	cigarette brands market share
6	usa population by state	apples market share
7	prime ministers of england	healthy food nutritional value
8	ipod models	hormones effects
9	bittorrent clients	household chemicals strength
10	olympus digital slrs	lakes altitude
11	composition of the sun	laptops cpu
12	running shoes	asian countries currency
13	fuel consumption	diseases risks
14	stock quote tables	external drives capacity
15	top grossing movies	baseball teams captain
16	nutrition values	maryland counties population
17	state capitals and largest cities in us	countries capital
18	professional wrestlers	diseases incidence
19	company income statements	eu countries year joined
20	dog breeds	irish counties area
21	ibanez guitars	cereals nutritional value
22	used cellphones	erp systems price
23	world religions	cats life span
24	stocks	broadway musicals director
25	academy awards	infections treatment
26	2008 olympic gold medal winners	food type
27	currencies of different countries	board games number of players
28	science discoveries	google products reviews
28	pga leaderboard	constellations closest constellation
30	pain medications	games age

5.2 Row Population

We present the results of row population task in Sect. 5.2.1, followed by further analysis in Sect. 5.2.2.

5.2.1 Experimental Results

Recall that we have introduced different candidate entities selection methods in 4.2.1, i.e. from entity category by KB, table caption, and table entities, and from embeddings. Correspondingly, we explore the row population performance on four alternative baselines in our task: (i) using relations of entity, (ii) estimating the semantic relatedness of entities based on outgoing links of them, (iii) based on Jaccard similarity, (iv) based on nearby words: Table2VecE*. Note that the former three baselines are introduced from literature.

The overall row population results are listed in Table 5.4. The top three lines show the results of the literature baselines. The middle line illustrates the result of our approach

Table 5.4: Row population performance. Statistical significance is tested against the respective baseline.

Method	#Seed entities ($ E $)									
	1		2		3		4		5	
	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
BL1	0.4360	0.5552	0.4706	0.5846	0.4788	0.5856	0.4786	0.5779	0.4711	0.5618
BL2	0.2612	0.4779	0.2778	0.4887	0.2845	0.4811	0.2846	0.4808	0.2817	0.4689
BL3	0.2912	0.5024	0.3024	0.4927	0.3028	0.4815	0.2987	0.4780	0.2910	0.4609
Table2VecE*	0.4982 [‡]	0.7623 [‡]	0.5522 [‡]	0.8081 [‡]	0.5598 [‡]	0.7993 [‡]	0.5543 [‡]	0.7787 [‡]	0.5476 [‡]	0.7744 [‡]
BL1+Table2VecE*	0.5581[‡]	0.7414 [‡]	0.6147[‡]	0.8141 [‡]	0.6400[‡]	0.8424 [‡]	0.6524[‡]	0.8427[‡]	0.6533[‡]	0.8372 [‡]
BL2+Table2VecE*	0.5461 [‡]	0.7710[‡]	0.6027 [‡]	0.8317[‡]	0.6187 [‡]	0.8440[‡]	0.6217 [‡]	0.8389 [‡]	0.6223 [‡]	0.8410[‡]
BL3+Table2VecE*	0.5461 [‡]	0.7710[‡]	0.6027 [‡]	0.8317[‡]	0.6187 [‡]	0.8440[‡]	0.6217 [‡]	0.8389 [‡]	0.6223 [‡]	0.8410[‡]

Table2VecE*. The bottom three lines are the results of combining the baselines with Table2VecE*. Out of the three literature baselines, BL1 performs far better than the other two in terms of both MAP and MRR. This indicates relations given by RDF triples are more beneficial for capturing entity similarity information. We further notice that Table2VecE* significantly outperforms all other baseline methods ($p < 0.01$).

We further combine Table2VecE* with three literature baselines to enhance the performance. Note that the combination involves a mixture parameter α (cf. Eq. (4.2.2)). To understand the potential of using table embeddings, we perform a grid search in steps of 0.1 for the value of α , and report results using the α value that yielded the best MAP score, see Fig. 5.2. The best performing α values for BL1, BL2, and BL3 are 0.4, 0.0, and 0.0, respectively. This means that the latter two baselines do not contribute at all to the combination. That is, the bottom two rows of Table 5.4 are based only on Table2VecE*, hence the scores in these two rows are identical. It also indicates Table2VecE* benefits from the candidate selection method by three literature baselines, given the performance is higher than Table2VecE* itself in terms of both MAP and MRR.

Overall, we find that the combined methods outperform the respective baselines substantially and significantly ($p < 0.01$). BL1 + Table2VecE* yields the best performance in terms of MAP, which means Table2VecE* and BL1 are complementary with each other. It is worth pointing out that the performance of this combined methods improves more with more seed entities than the baseline BL1, which flattens out already after two seed entities. This indicates the seed entities are better utilized in our embedding-based method.

5.2.2 Analysis

We continue to present our analysis of how the combination methods at bottom block of Table 5.4 influence individual tables. Figure 5.3 shows the result of Average Precision(AP) difference over individual tables in terms of input at #5. Note that BL1 and E1* refer to the AP difference of combination method(Table2VecE* & BL1) against baseline BL1 and baseline Table2VecE* respectively. When the difference(Δ) is smaller than 0.05, we

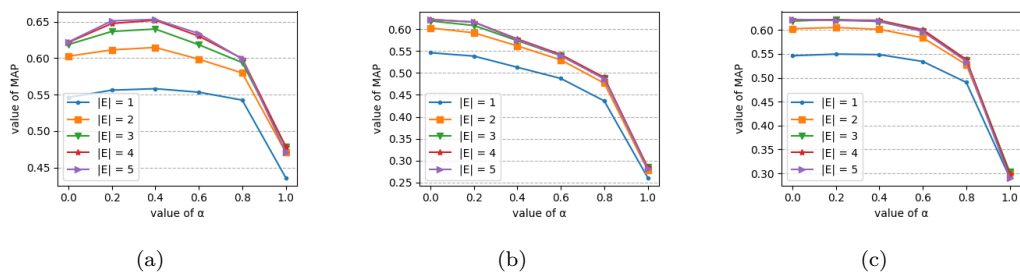


Figure 5.2: Effect of varying the interpolation parameters for combination methods in row population. Note that from left to right, each subfigure is corresponding to combination methods involving BL1, BL2, BL3 respectively.

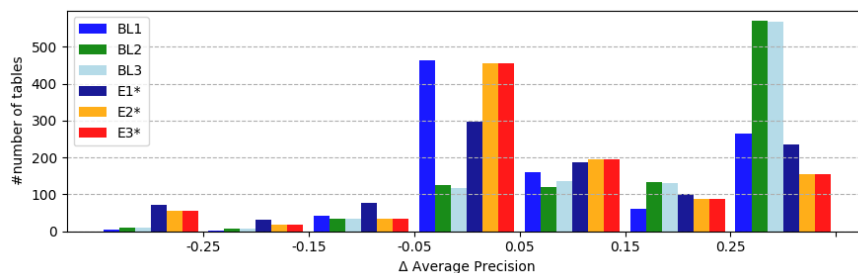


Figure 5.3: Row population performance of individual tables in terms of Average Precision difference at $\#E = 5$.

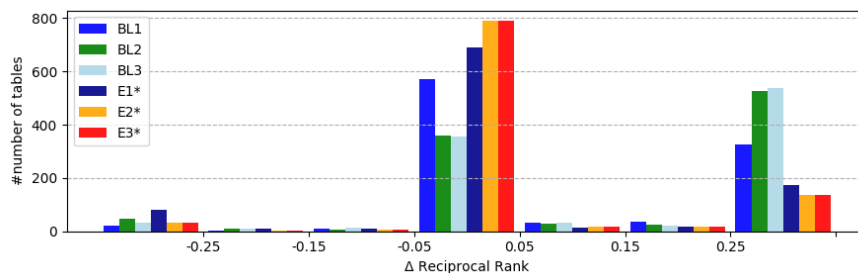


Figure 5.4: Row population performance of individual tables in terms of Reciprocal Rank difference at $\#E = 5$.

consider no change in terms of AP. The left and right bar groups refers to the number of tables that have negative and positive improvements separately. When the difference(Δ) is larger than 0.25, we assume significant change against AP, the left most bar group are the tables that hurt most and the right most bar group indicates tables that benefit the most.

According to Fig. 5.3, out of all the combination methods in Table 5.4, the number of tables have negative growth($\Delta < -0.05$) are much less than that have positive growth($\Delta > 0.05$), which indicates the fact that the combination of Table2VecE* and three other baselines outperforms corresponding individual methods. For BL2 and BL3, more than 800 tables have increased their AP performance. Recall that those two

Table 5.5: Column population performance.

Method	#Seed column labels ($ L $)					
	1		2		3	
	MAP	MRR	MAP	MRR	MAP	MRR
Baseline	0.2507	0.3753	0.2845	0.4037	0.2852	0.3552
Baseline + Table2VecH	0.2551^o	0.3796^o	0.3322[‡]	0.4400^o	0.4000[‡]	0.5080[‡]

methods themselves does not contribute at all to the combination performance, which means Table2VecE* contributes significantly in the ranking of relevant elements. We further analyze the ΔAP over Table2VecE*, the number of tables in different groups are the same when the combined method contains BL2, and BL3. More in detail, 439 tables has AP improved against Table2VecE*, and 155 tables achieve significant improve. When the combined method contains BL1, there are 524 tables benefits from the combination against Table2VecE*, and 236 tables have obtained $\Delta > 0.25$.

Figure 5.4 shows the result of reciprocal rank improvement over individual tables. We note that for all methods, a large amount of tables their Reciprocal Rank remain no change ($\Delta < 0.05$). Also the number of tables in the group of $0.15 < \Delta < 0.25$ are less than 100. Specifically according to the right most bar group, 325 tables have achieved significantly improvement against BL1, and more than 500 tables have largely improved their reciprocal rank against BL2 and BL3 ($\Delta > 0.25$). E2* and E3* have the same behavior, this is consistent with the result in Table 5.4 and they have 169 tables with Reciprocal Rank improved. When it comes to E1*, the number of tables improved becomes larger (207) and there are 175 table in the right most bar group ($\Delta > 0.25$). We further notice that Fig. 5.3 and 5.4 have similar behavior in terms of distribution of the number of tables.

5.3 Column Population

We report the column population results and analysis in Sect. 5.3.1 and Sect.5.3.2 respectively.

5.3.1 Experimental Results

In Sect. 4.3.1 we have introduced our candidate selection method for column population, i.e., (i) using table caption, (ii) using column heading labels, (iii) using table entities. For both performance and comparison reasons, we employ the same candidates from [7] in our baseline and combined method. The candidates are chosen by the combination of all three methods listed above.

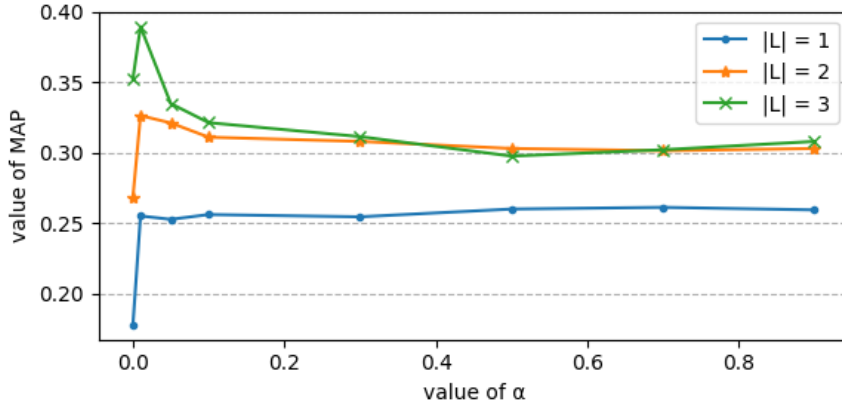


Figure 5.5: Effect of varying the interpolation parameters for combination method in column population.

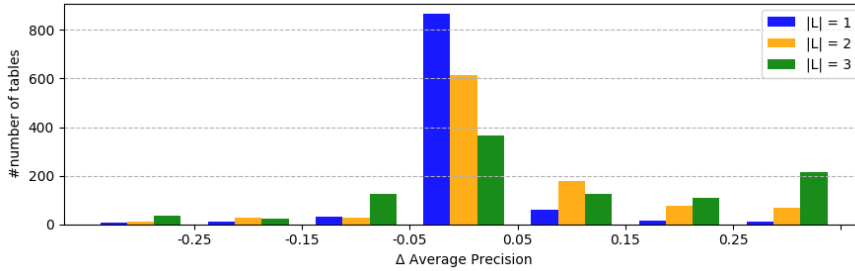


Figure 5.6: Column population performance of individual tables in terms of Average Precision.

Table 5.5 shows column population performance. Out of all input levels, we find that the combined method involving Table2VecH has achieved performance improvement against baseline in terms of both MAP and MRR. More specific, the combined method significantly outperforms the baseline method ($p < 0.01$) in terms of MAP when seed number exceeds 1, and when $|L| = 3$ it achieves substantial and significant improvements ($p < 0.01$) both in terms of MAP and MRR. Moreover, while the baseline performance does not improve with more seed column labels, the combined method can effectively utilize larger input sizes and keeps improving the performance.

According to Fig. 5.5, the interpolation parameter (cf. Eq. (4.3.2)) that yielded the best performance for the combined method is $\alpha = 0.01$, which indicates Table2VecH similarity is assigned much more importance than the baseline. Note that while $|L| = 1$, the Mean Average Precision is not the highest at the point $\alpha = 0.01$. But with more inputs given, it becomes clear that the best performance is achieved at $\alpha = 0.01$, hence we reported the result at this point in Table 5.5. We further notice for both methods, performance improves along with more seed column labels, because more information is given for determining the related labels. This phenomenon is consistent with that in our row population task.

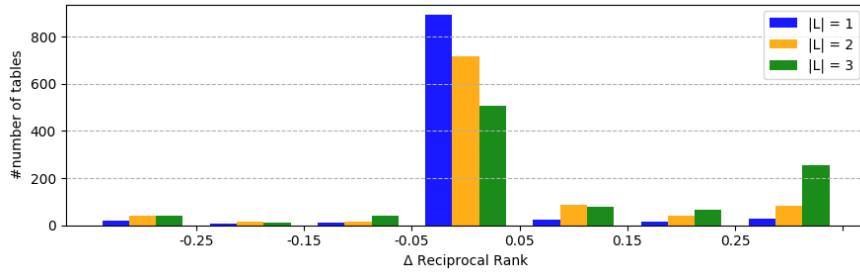


Figure 5.7: Column population performance of individual tables in terms of Reciprocal Rank.

5.3.2 Analysis

We continue our analysis of individual tables performance in column population. Figure 5.6 shows the result of Average Precision difference over individual tables in terms of three different input levels, #1, #2, #3. We notice that the number of tables that have no significant change ($\Delta < 0.05$) are the largest among all bar groups. We also notice that the number of tables that have positive Average Precision improvement increases along with the more inputs. More in detail, the number of tables with positive improvement are 85, 323, 446 for inputs at #1, #2, #3 respectively. This is because of more information given by more inputs, and eventually we achieve better rankings of some individual tables. We can see from Fig. 5.6, the number of tables remain no change decreases dramatically from 866 to 367 along with more inputs.

Figure 5.7 shows the result of reciprocal rank difference over individual tables, similarly a large amount of tables their Reciprocal Rank remains no change ($\Delta < 0.05$). We further notice that Fig. 5.6 and 5.7 have the same behavior.

5.4 Table Retrieval

In Sect. 5.4.1 we present the evaluation results of table retrieval tasks, and in Sect. 5.4.2 the further analysis is performed in detail.

5.4.1 Experimental Results

Table 5.6 reports the table retrieval results together with the significance testing results against the baseline. For all methods, their performance improve along with bigger cut-off point and at NDCG@20, they achieve the highest performance. We notice that the performance of all the methods improve over the baseline in terms of NDCG@5 but these improvements are not significant yet. With the cut-off point getting bigger, we

Table 5.6: Table retrieval evaluation results. Statistical significance is tested against the baseline.

Method	NDCG@5	NDCG@10	NDCG@15	NDCG@20
Baseline	0.5527	0.5456	0.5738	0.6031
Baseline + Word2Vec	0.5954 [°]	0.6006 [†]	0.6315 [‡]	0.6588 [†]
Baseline + Graph2Vec	0.5844 [°]	0.5764 [°]	0.6128 [°]	0.6340 [°]
Baseline + Table2VecW	0.5974 [°]	0.6096 [‡]	0.6312 [‡]	0.6505 [†]
Baseline + Table2VecE	0.5602 [°]	0.5569 [°]	0.5760 [°]	0.6161 [°]

start to achieve statistically improvement over some methods. We note that Table2VecW and Word2Vec have very comparable performance to each other and they outperform all other methods and significantly improve over baseline method ($p < 0.01$). For Graph2Vec and Table2VecE, we achieve improvements over the baseline in terms of all NDCG cut-off points, but these are not statistically significant.

The lack of difference between the two word embeddings indicates that it does not make a difference for the table retrieval task whether word embeddings are trained specifically on tables or not. These results also show that word embeddings are more beneficial for table retrieval than entity and graph embeddings.

5.4.2 Analysis

We further conduct analysis of all four different semantic representations listed in Table 5.6 against the baseline method in terms of individual queries, and compute the query-level differences on the two query subsets between the baseline and our embeddings Table2VecE and Table2VecW.

In Fig. 5.8, we present the results for both our table embedding methods and baseline against two query subsets, QuerySet 1 and QuerySet 2, in terms of NDCG@20. We note that both our table embeddings methods outperform the baseline over two query subsets. Moreover, out of three methods, the performance on QuerySet 1 exceeds that on QuerySet 2 which contains more specific queries. Table2VecW has the best performance among those three, which is consistent with our discover before in Table 5.6.

Figure 5.9 shows the performance of different semantic methods against baseline over individual queries in terms of NDCG@20. We note that out of four methods, almost half of the queries their performance remains no change ($\Delta < 0.05$). We notice that Word2Vec and Graph2Vec have similar distribution patterns. The former one has less queries that were impaired and more that were improved, while the queries that were significantly helped were less. Word2Vec outperforms Graph2Vec in terms of the overall improvement. We further note Table2VecE has least queries that were significantly

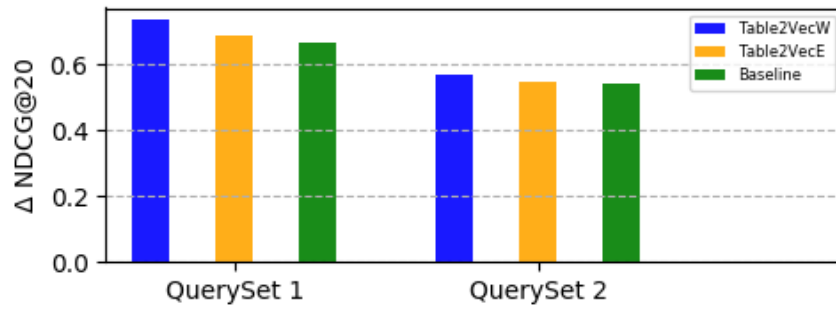


Figure 5.8: Table retrieval results against query subsets in terms of NDCG@20.

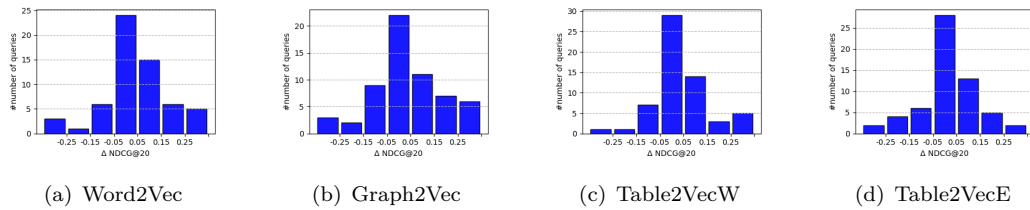


Figure 5.9: Performance of different methods against baseline over individual queries in terms of NDCG@20.

helped among all the methods, which is consistent with its overall performance in Table 5.6. As for Table2VecW, it has less queries that were significantly hurt (left most bar) compared with other methods, and only 9 queries (the left three bars) were hurt in total. Figure 5.10 illustrates the differences between baseline and Table2VecW over individual queries in two query subset respectively against NDCG@20. And we present the retrieval results of the queries we discuss in the following parts in Table 5.7. The left (or right) most bar represents the query with most significant improvement (or impairment) over baseline.

For Fig. 5.10(a), the left most bar corresponds to the query, *stocks*, which has three relevant tables in the corpus. For the baseline method, it has retrieved none of those tables, while Table2VecW managed to return the all the relevant ones in the top 4 with the highly relevant one at the rank of 1. The improvement is up to 0.936 in this case. We also checked out the rank of *stocks* by Table2VecE, which remain no change compared with baseline method. The right most bar represents the result of query *used cellphones*, which has only one relevant tables. The baseline method returned the relevant table in the 9th place, while Table2VecW did not retrieve it at all. And the improvement in this case is -0.301. For table2VecE, it managed to retrieve the table at the 18th place.

For Fig. 5.10(b), the query that the left most bar represents is *food type*. The baseline method managed to return 6 highly relevant tables in the top 18 result, while our method has found 7 highly relevant and 1 relevant results in the top 19. In this case

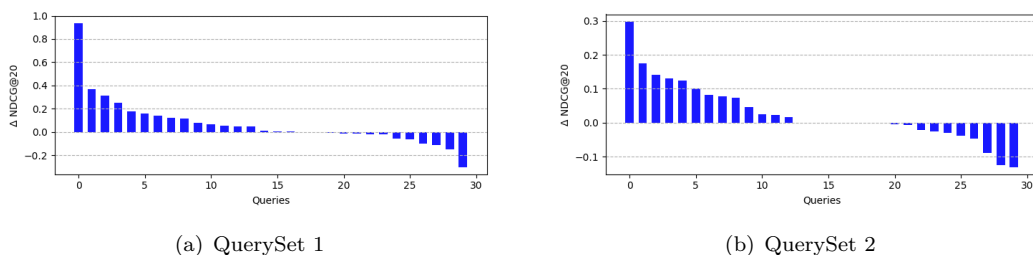


Figure 5.10: Query-level differences on the two query subsets between the baseline and Table2VecW.

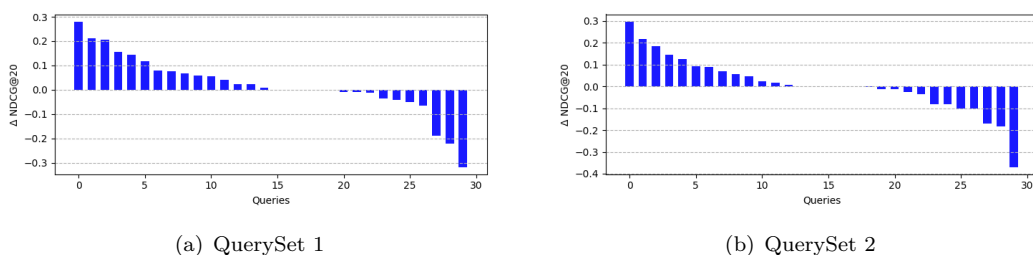


Figure 5.11: Query-level differences on the two query subsets between the baseline and Table2VecE.

the improvement over baseline is 0.2978. Table2VecE also has achieved improvement against this query. The right most bar is the result of query *hormones effects*. The main difference of baseline and Table2VecW results is that the baseline method managed to return 1 more highly relevant tables, *Bisphenol A/Low-dose exposure in animals*, at the rank of 14. Here the impairment is 0.1302. Compared with baseline, Table2VecE managed to get a better rank of the results, hence an improvement of 0.0569 against the performance.

Figure 5.11 illustrates the differences between baseline and Table2VecE over individual queries in two query subset respectively against NDCG@20. In the Fig. 5.11(a), *composition of the sun* is the query corresponds to the left most bar, and it has 1 highly relevant and 1 relevant tables in the corpus. Table2VecE beats baseline method in this scenario because it managed to return the highly relevant one in the first place instead of the second which gives us the improvement of 0.2805. Table2VecW also achieved good results over this query. The right most bar represents the result of query *nutrition values*. The baseline method managed to return up to 10 highly relevant and 1 relevant results in the top 16, while Table2VecE found only 6 highly relevant ones. This undermined the performance up to 0.3176. Against this query, Table2VecW outperforms Table2VecE but not the baseline.

For Fig. 5.11(b), the query *eu countries year joined* improved the most against baseline method and two tables are marked as relevant to it in the table corpus. Table2VecE

Table 5.7: Examples of retrieval results for table embeddings. Note Rel. denotes the relevance level, B. represents our baseline method, W. and E. here refer to Table2VecW and Table2VecE respectively. (+) and (-) refer to the query performance is improved or impaired separately.

Query	Rel.	B.	W.	E.
(+)stocks:				
<i>Stocks for the Long Run/Key Data Findings: annual real returns</i>	2	-	1	-
<i>Hang Seng Index/Selection criteria for the HSI constituent stocks</i>	1	-	3	-
<i>TOPIX/TOPIX New Index Series</i>	1	-	4	-
(-)used cellphones:				
<i>List of companies of Taiwan/D</i>	1	9	-	18
(+)food type:				
<i>List of Philippine dishes/Miscellaneous and street food</i>	2	3	1	1
<i>List of Spanish dishes/Others</i>	2	9	7	9
<i>List of Philippine dishes/Pickles and side dishes</i>	2	15	9	10
<i>List of Spanish dishes/Breads and pastries</i>	2	16	3	5
<i>List of Philippine dishes/Breads and pastries</i>	2	17	5	6
<i>List of Malaysian dishes/Dishes</i>	2	18	10	18
<i>List of Malaysian dishes/Noodle dishes</i>	2	-	17	19
<i>List of rice dishes/Rice dishes</i>	1	-	19	-
(-)hormones effects:				
<i>List of human hormones/Steroid</i>	2	1	1	1
<i>Bioidentical hormone replacement therapy/Lack of evidence for claims</i>	1	2	2	4
<i>Anterior pituitary/Major hormones secreted</i>	2	3	3	2
<i>Reference ranges for blood tests/Thyroid hormones</i>	1	4	7	6
<i>Hypothalamus/Endocrine hormones</i>	2	7	14	3
<i>Bisphenol A/Low-dose exposure in animals</i>	2	14	-	13
Query				
(+)composition of the sun:				
<i>Atmosphere of Jupiter/Elemental abundances relative to hydrogen in Jupiter and Sun</i>	2	2	1	1
<i>White dwarf/Composition and structure</i>	1	3	3	4
(-)nutrition values:				
<i>Goat/Basic composition of various milks (mean values per 100g)</i>	2	1	1	1
<i>Crunchie/Nutrition information</i>	2	2	3	3
<i>Parenteral nutrition/Total parenteral nutrition</i>	2	3	4	4
<i>Space Raiders/Nutrition information</i>	2	5	5	5
<i>Sprite Zero/Nutrition</i>	2	6	-	6
<i>Jelly Tots/Nutrition information</i>	2	7	6	11
<i>Pepita/Nutrition</i>	1	8	-	8
<i>Solanum quitoense/Nutrition</i>	2	9	-	9
<i>Indomie Mi goreng/Nutrition Information</i>	2	12	12	13
<i>V8 (beverage)/11.5 fluid ounce (340 mL) can of V8 100% Vegetable Juice (United States)</i>	2	15	-	-
<i>Oak (flavoured milk)/Nutrition</i>	2	16	-	16
(+)eu countries year joined:				
<i>Mandatory renewable energy target/Selected EU countries</i>	1	-	15	-
<i>National identity cards in the European Economic Area/Overview of national identity cards</i>	1	-	18	-
(-)cereals nutritional value:				
<i>Sesame/Sesame seed kernels, toasted</i>	2	1	2	1

outperforms the baseline against this single query because it retrieved both relevant result in the 15th and 18th place while both the baseline method and Table2VecW found no match at all. And this gives us an improvement of 0.2976 in terms of NDCG@20. *cereals nutritional value* is the query that hurt most in this case with the impairment of 0.3691 in the performance. Both methods have successfully retrieved the only highly relevant table, the difference is that the baseline gives a higher rank against Table2vecE.

Chapter 6

Conclusion and Future Work

In this chapter, we present the summary and outlook of our research. Section 6.1 gives a brief conclusion regarding our work and followed by explanation about the three research questions given in Chap. 1. In Sect. 6.2, we report the future directions of our research.

6.1 Conclusion

Tables contain information in a structured form which are significantly useful to many table-related applications. How to leverage tabular data is a problem on its own count. In this thesis, our goal was to investigate the performance of different table-related tasks while introducing neural embeddings derived particularly for these tasks. Specifically, We have introduced Table2Vec, a neural language model for training four different kinds of embeddings on various table elements. These embeddings are derived based on Wikipedia Tables corpus which contains only high-quality relational tables, and have subsequently been utilized in various table-related tasks, such as table population and table retrieval.

For table population, we have concentrated on tables with an entity focus and experimented on two different types of tasks regarding rows and columns respectively. In more detail, Table2VecE* considers entities from the left most column of the table, and have been leveraged for row population task. For column population, we have trained an embedding called Table2VecH based on table column labels extracted from column headings. We have employed cosine similarity to calculate entity and label pairwise similarity for row and column population respectively. these calculations were based on the semantic vector representations we have obtained. For evaluation, we have chosen candidates through a KB and introduced a process that simulates a user through his work of populating a table with additional rows and columns. We have shown that our

methods significantly and substantially outperform all baselines. Especially, when the number of seed labels becomes larger, Table2VecH achieves 40% relative improvement over the baseline. For both methods, combining with an effective baseline has led to further improvements.

Table retrieval is a task of returning a ranked list of tables in response to a keyword query. We have investigated a novel semantic retrieval framework using neural word and entity embeddings, where queries and tables are represented as semantic vectors. We have proposed Table2VecW where embeddings are based on all words appearing in a table, and Table2VecE which extracts all table entities to train the model. We have experimented on a combination of multiple vector similarity measures for matching those semantic representations. For evaluation, we have employed the metrics and test collection by [4]. The results have shown significantly improvement of retrieval effectiveness against a strong baseline.

Recall that we have proposed our research questions in Chap. 1. After the evaluation and analysis of our results, we are ready to answer them.

RQ1 Can Table2Vec improve table population performance against the state-of-the-art baselines?

Neural embeddings have shown interesting applications to many existing table-related domains such as table classification [16, 59], table retrieval [7] but table population. We have proposed a novel approach utilizing table embeddings to help a user populate a seed table, and evaluate the performance against baselines using a knowledge base. According to the results and analysis in Sect. 5.2 and Sect. 5.3, we conclude that methods that have incorporated our Table2Vec neural embeddings significantly outperform the state of the art.

RQ2 Would different training datasets affect the embeddings thus the retrieval result?

We consider this research question in terms of table retrieval task. There are some pre-trained embeddings available for us to use directly in our tasks, such as word embeddings by [56] based on Google News Data. In this thesis, we have derived our own neural word embeddings from Wikipedia Tables corpus. In the experiment process, we employ both models in the table retrieval task, and the results in Sect. 5.4 have shown that it does not make a difference for the table retrieval task whether word embeddings are trained specifically on tables or not.

RQ3 Which of the semantic representations performs better in table retrieval?

In this thesis, we have investigated different types of semantic representation in the task of table retrieval, see Table 5.6. According to our observation, Table2VecE and

Graph2Vec have achieved improvements over the baseline, but these are not statistically significant. The method employing Table2VecW outperforms the start-of-the-art baseline by over 10%. This is on par with using pre-trained Word2Vec embeddings using Google News data. To summary, word embeddings yields out the best performance among all the semantic models.

6.2 Future Work

In this thesis, we have investigated the performance of various table-related tasks while incorporating neural word and entity embeddings. We have achieved considerable improvements over the state-of-the-art baselines. We have derived these embeddings particularly on Wikipedia Tables corpus which contains only high-quality relational tables. In the future, we wish to extend our experiments to other corpus of data, as well as other types of tables, e.g., entity table and matrix table. We also plan to employ these Table2Vec embeddings for additional table-related tasks, and further extend our approach to embed tables themselves, instead of specific table elements.

Appendix A

Resources

Resources developed within this thesis are available at the following GitHub repository:

<https://github.com/ninalx/table2vec-lideng>

- `table2vec-lideng/gt/`: This directory contains groundtruth files for three table tasks.
- `table2vec-lideng/runfile/`: The directory consists of run files for table retrieval and column population. Run files for row population are too large to be added on GitHub.
- `table2vec-lideng/python/`: This directory has Python programs, for indexing tables, generating groundtruth files, generating run files, etc.
- `table2vec-lideng/data/`: We put other important data under this path, for example, test table IDs and search queries.

Bibliography

- [1] Oriol Vinyals and Quoc Le. A neural conversational model. *CoRR*, abs/1506.05869, 2015.
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.
- [3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. of NIPS '12*, page 1097 – 1105, 2012.
- [4] Shuo Zhang and Krisztian Balog. Ad hoc table retrieval using semantic similarity. In *Proc. of WWW '18*, pages 1553 – 1562, 2018.
- [5] Michael J. Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proc. of VLDB Endow*, 2(1):1090 – 1101, 2009.
- [6] Rakesh Pimplikar and Sunita Sarawagi. Answering table queries on the web using column keywords. *Proc. of VLDB Endow*, 5(10):908 – 919, 2012.
- [7] Shuo Zhang and Krisztian Balog. Entitables: Smart assistance for entity-focused tables. In *Proc. of SIGIR '17*, pages 255–264, 2017.
- [8] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proc. of SIGMOD '12*, pages 817 – 828, 2012.
- [9] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: Entity augmentation and attribute discovery by holistic matching with web tables. In *Proc. of SIGMOD '12*, pages 97 – 108, 2012.
- [10] Ahmad Ahmadov, Maik Thiele, Julian Eberius, Wolfgang Lehner, and Robert Wrembel. Towards a hybrid imputation approach using web tables. In *Proc. of BDC '15*, pages 21 – 30, 2015.
- [11] Yoones A. Sekhavat, Francesco di Paolo, Denilson Barbosa, and Paolo Merialdo. Knowledge base augmentation using tabular data. In *Proc. of LDOW '14*, 2014.

- [12] Xiaolu Zhang, Yueguo Chen, Jinchuan Chen, Xiaoyong Du, and Lei Zou. Mapping entity-attribute web tables to web-scale knowledge bases. In *Proc. of DASFAA '13*, pages 108 – 122, 2013.
- [13] Bhagavatula, Chandra Sekhar, Noraset Thanapon, and Downey Doug. Tabel: Entity linking in web tables. In *Proc. of ISWC '15*, pages 425 – 441, 2015.
- [14] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: Exploring the power of tables on the web. *Proc. of VLDB Endow*, 1(1): 538 – 549, 2008.
- [15] Majid Ghasemi-Gol and Pedro A. Szekely. Tabvec: Table vectors for classification of web tables. *CoRR*, abs/1802.06290, 2018.
- [16] Anna Lisa Gentile, Petar Ristoski, Steffen Eckel, Dominique Ritzke, and Heiko Paulheim. Entity matching on web tables: a table embeddings approach for blocking. In *Proc. EDBT '17*, pages 510 – 513, 2017.
- [17] Ellen M Voorhees and Donna K Harman. Trec: Experiment and evaluation in information retrieval. *MIT press Cambridge*, 1, 2005.
- [18] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Proc. of NIPS '15*, page 1693 – 1701, 2015.
- [19] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016.
- [20] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333 – 389, 2009.
- [21] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. In *Proc. of SIGIR '04*, 22:179 – 214, 2004.
- [22] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3:225 – 331, 2009.
- [23] Dwaipayan Roy, Debjyoti Paul, Mandar Mitra, and Utpal Garain. Using word embeddings for automatic query expansion. *CoRR*, abs/1606.07608, 2016.
- [24] Fernando Diaz, Bhaskar Mitra, and Nick Craswell. Query expansion with locally-trained word embeddings. In *Proc. of ACL*, page 367–377, 2016.

- [25] Saar Kuzi, Anna Shtok, and Oren Kurland. Query expansion using word embeddings. *In Proc. of CIKM '16*, pages 1929 – 1932, 2016.
- [26] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. *In Proc. of WWW '17*, page 1291 – 1299, 2017.
- [27] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. *In Proc. of CIKM '13*, page 2333 – 2338, 2013.
- [28] Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. A dual embedding space model for document ranking. *CoRR*, abs/1602.01137, 2016.
- [29] Richard C. Wang and William W. Cohen. Language-independent set expansion of named entities using the web. *In Proc. of ICDM '07*, pages 342 – 350, 2007.
- [30] Richard C. Wang and William W. Cohen. Iterative set expansion of named entities using the web. *In Proc. of ICDM '08*, page 1091 – 1096, 2008.
- [31] Richard C. Wang and William W. Cohen. Character-level analysis of semi-structured documents for set expansion. *In Proc. EMNLP '09*, pages 1503 – 1512, 2009.
- [32] Yeye He and Dong Xin. Seisa: set expansion by iterative similarity aggregation. *In Proc. of WWW '11*, pages 427 – 436, 2011.
- [33] Xiaoxin Yin, Wenzhao Tan, and Chao Liu. Facto: A fact lookup engine based on web tables. *In Proc. of WWW '11*, 2011.
- [34] Steffen Metzger, Ralf Schenkel, and Marcin Sydow. Qbees: query-by-example entity search in semantic knowledge graphs based on maximal aspects, diversity-awareness and relaxation. *Journal of Intelligent Information Systems*, 49:333 – 366, 2017.
- [35] Steffen Metzger, Ralf Schenkel, and Marcin Sydow. Qbees: query by entity examples. *In Proc. of CIKM '13*, pages 1829 – 1832, 2013.
- [36] Steffen Metzger, Ralf Schenkel, and Marcin Sydow. Aspect-based similar entity search in semantic knowledge graphs with diversity-awareness and relaxation. *In Proc. of WI-IAT '14*, pages 60 – 69, 2014.
- [37] Marc Bron, Krisztian Balog, and Maarten de Rijke. Example based entity search in the web of data. *In Proc. of ECIR '13*, pages 392 – 403, 2013.
- [38] Oliver Lehmberg, Dominique Ritzke, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. The mannheim search join engine. *Web Semant*, page 159 – 166, 2015.

- [39] Michael J. Cafarella, Alon Halevy, and Jayant Madhavan. Structured data on the web. *Commun. ACM*, 54:72 – 79, 2011.
- [40] Jayant Madhavan, Loredana Afanasiev, Lyublena Antova, and Alon Y. Halevy. Harnessing the deep web: Present and future. *CoRR*, abs/0909.1785, 2009.
- [41] Sunita Sarawagi and Soumen Chakrabarti. Open-domain quantity queries on web tables: Annotation, response, and consensus models. *In Proc. of KDD '14*, pages 711 – 720, 2014.
- [42] Meihui Zhang and Kaushik Chakrabarti. Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables. *In Proc. of SIGMOD '13*, pages 145 – 156, 2013.
- [43] Xin Dong, Evgeniy Gabrilovich, Jeremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmman, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. *In Proc. of KDD '14*, pages 601 – 610, 2014.
- [44] Eric Crestan and Patrick Pantel. Web-scale table census and classification. *In Proc. of WSDM '11*, pages 545 – 554, 2011.
- [45] Emir Munoz, Aidan Hogan, and Alessandra Mileo. Using linked data to mine rdf from wikipedia’s tables. *In Proc. of WSDM '14*, pages 533 – 542, 2014.
- [46] Stefan Zwicklbauer, Christoph Einsiedler, Michael Granitzer, and Christin Seifert. Towards disambiguating web tables. *In Proc. of ISWC-PD '13*, pages 205 – 208, 2013.
- [47] Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: Learning to query tables in natural language. *In Proc. of IJCAI '16*, page 2308 – 2314, 2016.
- [48] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendesf, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Soren Auer, and Christian Bizer. Dbpedia – a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167 – 195, 2015.
- [49] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. *In SIGMOD Conference*, pages 1247 – 1250, 2008.
- [50] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. *In Proc. of WWW '07*, pages 697 – 706, 2007.

-
- [51] Jingjing Wang, Haixun Wang, Zhongyuan Wang, and Kenny Q. Zhu. Understanding tables on the web. In *Proc. of ER '12*, pages 141 – 155, 2012.
- [52] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. Methods for exploring and mining tables on wikipedia. In *Proc. of IDEA '13*, pages 18 – 26, 2013.
- [53] AnHai Doan and Alon Y. Halevy. Semantic-integration research in the database community. *AI Magazine*, 26(1):83 – 94, 2005.
- [54] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334 – 350, 2001.
- [55] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proc. of VLDB Endow*, 4(9):528 – 538, 2011.
- [56] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of NIPS '13*, pages 3111 – 3119, 2013.
- [57] Petar Ristoski and Heiko Paulheim. RDF2vec: RDF graph embeddings for data mining. In *Proc. of ISWC '16*, pages 498 – 514, 2016.
- [58] Kyosuke Nishida, Kugatsu Sadamitsu, Ryuichiro Higashinaka, and Yoshihiro Matsuo. Understanding the semantic structures of tables with a hybrid deep neural network architecture. In *Proc. of AAAI '17*, page 168 – 174, 2017.
- [59] Majid Ghasemi-Gol and Pedro Szekely. Tabvec: Table vectors for classification of web tables. *CoRR*, abs/1802.06290, 2018.
- [60] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *Proc. of KDD '14*, pages 701 – 710, 2014.
- [61] Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proc. of KDD '15*, pages 1365 – 1374, 2015.
- [62] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledge-base. *Communications of the ACM*, 57:78 – 85, 2014.
- [63] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Distributed representations. *Parallel distributed processing: explorations in the microstructure of cognition*, pages 77 – 109, 1986.

-
- [64] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernocký. Empirical evaluation and combination of advanced language modeling techniques. In *Interspeech*, pages 605 – 608, 2011.
- [65] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137 – 1155, 2003.
- [66] Holger Schwenk. Continuous space language models. *Computer Speech and Language*, 21:492 – 518, 2007.
- [67] David Milne and Ian H. Witten. An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *Proc. of AAAI '08*, 2008.
- [68] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.