Journal of Computer Science and Cybernetics, V.37, N.4 (2021), 365–402
DOI 10.15625/1813-9663/37/4/16151

DEEP LEARNING FOR SEMANTIC MATCHING: A SURVEY

HAN LI, YASH GOVIND, SIDHARTH MUDGAL, THEODOROS REKATSINAS, ANHAI DOAN*

Department of Computer Science, University of Wisconsin, Madison, Wisconsin, USA

Abstract. Semantic matching finds certain types of semantic relationships among schema/data constructs. Examples include entity matching, entity linking, coreference resolution, schema/ontology matching, semantic text similarity, textual entailment, question answering, tagging, etc. Semantic matching has received much attention in the database, AI, KDD, Web, and Semantic Web communities. Recently, many works have also applied deep learning (DL) to semantic matching. In this paper we survey this fast growing topic. We define the semantic matching problem, categorize its variations into a taxonomy, and describe important applications. We describe DL solutions for important variations of semantic matching. Finally, we discuss future R&D directions.

Keywords. Deep learning, semantic matching.

1. INTRODUCTION

Semantic matching is the problem of finding certain types of semantic relationships among schema/data constructs. Examples of such constructs include table attributes, relational tuples, textual mentions, sentences, and ontological concepts. Well-known problem classes of semantic matching include entity matching, entity linking, coreference resolution, schema/ontology matching, semantic text similarity, textual entailment, question answering, and tagging (see Section 2).

Semantic matching plays a fundamental role in data cleaning, data integration, knowledge base construction, natural language understanding, question answering, among many others. As a result, over the past few decades, semantic matching has received much attention in the database, AI, Web, KDD, and Semantic Web communities. Going forward, it will receive even more attention, as data science applications proliferate. This is because such applications often must integrate data from disparate sources before analysis can be carried out to extract insights, and such data integration often requires semantic matching. Further, such applications often benefit from using knowledge bases and domain ontologies, and building such knowledge bases and ontologies often requires semantic matching.

In the past few years, deep learning (DL) has become a major direction in machine learning [1, 2, 3, 4]. DL yields state-of-the-art results for tasks over data with some hidden

Dedicated to Professor Phan Dinh Dieu on the occasion of his 85th birth anniversary.

*Corresponding author.
E-mail addresses: hanli@cs.wisc.edu (H. Li); yashg@cs.wisc.edu (Y. Govind); sidharth@cs.wisc.edu (S. Mudgal); rekatsinas@cs.wisc.edu (T. Rekatsinas); anhai@cs.wisc.edu (A.H. Doan).

structure, e.g., text, image, and speech. On such data, using labeled examples, DL can automatically construct important features, thereby obviating the need for manual feature engineering. This has transformed fields such as image and speech processing, autonomous driving, robotics, natural language processing (NLP), and many others [1, 2]. Recently, DL has also gained the attention of database researchers [5, 4].

As described, deep learning appears well suited for semantic matching, and indeed numerous works have applied deep learning to semantic matching, with promising results. In this paper we survey these works. As far as we can tell, no such survey exists today. Many surveys on various aspects of semantic matching exist (as we discuss in Section 2). But they either do not cover deep learning; or they do, but only for certain problem classes of semantic matching (e.g., those often encountered in natural language processing) and only to a limited depth.

In contrast, in this paper *we aim to provide a comprehensive survey of deep learning approaches to the broad range of semantic matching problems.* This survey can be beneficial in three important ways. First, for seasoned researchers working in semantic matching, the survey can help them deepen their understanding of DL-based solutions, obtain a global view of the semantic matching landscape, and understand how DL solutions to one semantic matching problem can be transferred to another. Second, the survey can help new researchers quickly gain up-to-date knowledge about this important area. Finally, the survey can be used to teach graduate students about DL-based approaches to semantic matching.

The rest of this survey is organized as follows. Section 2 briefly surveys the landscape of semantic matching. It defines the different problem types of semantic matching, categorizes them into a taxonomy, and describes important applications. Section 3 then surveys DL solutions for the important semantic matching problem types, such as entity linking, textual entailment, entity matching, etc. Section 4 discusses future R&D directions, and Section 5 concludes.

2. THE SEMANTIC MATCHING PROBLEM LANDSCAPE

In this section we define the different problem types of semantic matching, categorize them into a taxonomy, then describe important applications.

2.1. Defining semantic matching

As discussed in the introduction, semantic matching is the problem of finding certain types of "semantic relationships" among "schema/data constructs". We now elaborate on this definition. We first discuss databases, ontologies/knowledge bases, and text documents, then use them to discuss schema/data constructs and semantic relationships.

Databases. A database typically consists of a schema and data, e.g., the schema of a relational database defines a set of tables, and the data consists of a set of tuples for each of these tables. Other database types include XML, JSON, key-value stores, etc. In this survey we focus on relational databases, as most semantic matching works have focused on this type of database.

Ontologies/Knowledge bases. An ontology captures the most important aspects of a real-world domain, e.g., e-commerce, real estate, manufacturing, etc. Figure 1 shows a toy

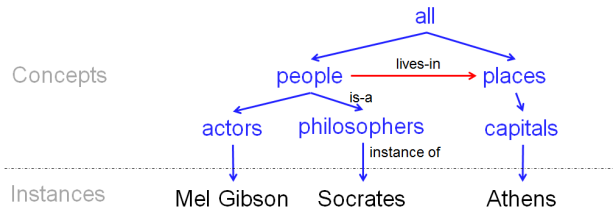

Figure 1: An example of an ontology

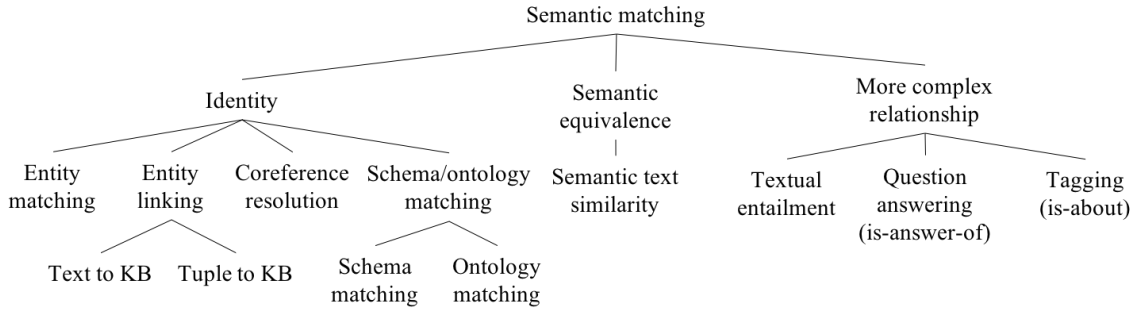

Figure 2: A taxonomy of the most popular semantic matching problems that have been considered by existing work

ontology. An ontology typically consists of a set of *concepts* being organized into a *taxonomy*, such that a child concept is a type of a parent concept (e.g., "actors" is a type of "people", see Figure 1). Each concept is associated with a set of *data instances* (e.g., "Mel Gibson" is an instance of concept "actors"). There are *relationships* among the concepts, e.g., lives-in is a relationship between "people" and "places", and the two data instances "Socrates" and "Athens" participate in this lives-in relationship.

The ontology in Figure 1 is tiny. In practice, ontologies can contain anywhere from hundreds to tens of millions of concepts, and from thousands to millions of relationships [6]. Ontologies are often called *knowledge bases* or *knowledge graphs*. A *data catalog*, or *master data index*, can often be viewed as an ontology which is simplified in certain ways. A *controlled vocabulary* can also be viewed as a simplified ontology with just a set of concepts.

Text documents. A text document contains natural text in a language (e.g., English). Text documents include paragraphs, sentences, phrases (e.g., "a black cat"), and pronouns (e.g., "I", "she").

An important notion that we will refer to often is "text mention" (or "mention" for short), which is a string in a document that is a name referring to a real-world entity. For example, given the two sentences "Harry Potter went to Hogwarts. Initially he disliked it.", "Harry Potter" is a mention of a person and "Hogwarts" is a mention of an organization (note that here "he" is not a mention, but a pronoun).

Another important notion is "text snippet", which in general can refer to a mention, a phrase, a sentence, or several consecutive sentences. However, most existing works use "text snippet" to refer to a single sentence. So in this survey, unless noted otherwise, when we say "text snippet" we mean a sentence.

Schema/Data constructs. We can now define the notion of schema/data construct. A

schema construct refers to a component of a database schema or an ontology schema. Examples of the former include an attribute or a set of attributes in a relational schema, a table schema, etc. Examples of the latter include a concept or a set of concepts in an ontology.

A data construct refers to a data component in a database (e.g., a relational tuple), in an ontology (e.g., an instance of a concept, such as "Socrates", "Athens"), or in a text document (e.g., mentions, pronouns, text snippets).

Semantic relationships. We want to find relationships between schema/data constructs *that correspond to relationships in the real world*. Examples include *identity relationships*, e.g., the two tuples (David Smith, UWM) and (D. Smith, UW-Madison) refer to the same real-world person, *semantic equivalence relationship*, e.g., the two sentences "He is wise" and "He is a sage man" are semantically equivalent, and *entailment*, e.g., the sentence "The cat is chasing a mouse" entails, i.e., implies, the sentence "The cat is moving".

We call these relationships *semantic relationships*. Finding them is the goal of semantic matching. Thus, semantic matching is different from *syntactic matching*, where the goal is to find "matches" that satisfy some syntactic constraints, e.g., finding all strings in a document that satisfy a given regular expression, or finding all person mentions that are within five words of an organization mention.

As described, semantic relationships can cover a wide spectrum, ranging from simple ones such as identity relationships to significantly more complex ones. However, most existing works have considered only a small number of relatively simple semantic relationships, as we discuss next.

2.2. A Taxonomy of semantic matching

We organize the most popular semantic matching problems (as considered by existing works) into the taxonomy in Figure 2. The first subtree of the taxonomy (titled "Identity") describes works that find identity relationships, i.e., find schema/data constructs that refer to the same real-world entity. Specific problems include:

- *Entity matching*, e.g., match tuple (David Smith, UWM) with tuple (D. Smith, UW-Madison).
- *Entity linking*, e.g., link mention "Madison" in a text document to the data instance "City of Madison" in an ontology.
- *Coreference resolution*, e.g., given the two sentences "Madison is the capital of Wisconsin. It has four lakes.", determine that "It" refers to "Madision", not to "Wisconsin".
- *Schema/ontology matching*, e.g., match the attribute "pname" in one table with attribute "person_name" in another table, and match concept "Faculty" in one ontology with concept "Academic_Staff" in another ontology.

The second subtree of the taxonomy (titled "Semantic equivalence") describes works that find semantic equivalence between two data/schema constructs. Most works in this subtree have focused on *semantic text similarity*, i.e., determining if two text snippets are semantically equivalent, such as "He is wise" and "He is a sage man".

The third and final subtree of the taxonomy (titled "More complex relationships") covers existing works that find more complicated semantic relationships. Specific problems include:

- *Textual entailment*, e.g. determining that the sentence "The cat is chasing a mouse" implies the sentence "The cat is moving".

- *Question answering*, e.g., determining that "50" is the answer to the question "How many states are in the US?". This is an example of the relationship "answer-of".
- *Tagging*, e.g., tagging the tweet "The president visited the UK" with "US politics", "UK", "foreign affairs". This is an example of the relationship "is-about-topic".

In the next three subsections, we describe these three subtrees of the semantic matching taxonomy in detail.

2.3. Finding semantic identity relationships

Entity matching. This problem finds *matching* data instances, i.e., those that refer to the same real-world entity, such as the two tuples (Joe Wilson, New York) and (J. Wilson, NY). See [7, 8, 9, 10, 11, 12, 13] for recent books, surveys, and tutorials on entity matching.

A common entity matching problem variation considers two tables A and B and finds all tuple pairs $(a \in A,\ b \in B)$ that match. We call such pairs *matches*. Figure 3 shows an example of three matches between two tables. This problem is also known as *entity resolution, record linkage, entity alignment, reference reconciliation*, and more. The problem of finding all matching tuples within a single table is also called *deduplication*. Entity matching plays a fundamental role in many data management applications, such as data cleaning and integration [7, 8].

Entity linking. This problem is also known as *named entity linking (NEL)* and *named entity disambiguation (NED)*. See [14, 15] for surveys and book chapters on entity linking. The goal of entity linking is to link (i.e., match) entity mentions in text documents to entities (i.e., data instances) in knowledge bases (KBs) [15], such as YAGO [16] and DBpedia [17].

For example, given the text document and the small KB in Figure 4, we need to link the mention "Madison" in the document to the correct entity (i.e., data instance) in the KB, which in this case is "Madison, WI", not "Madison, IL" (if a mention has no corresponding entity in a KB, we can assign to it a special label "NIL", indicating that the mention is unlinkable).

Note that entity linking is different from *entity extraction, a.k.a. named entity recognition (NER)* [18]. Given a tweet such as "Obama gave an immigration speech while on vacation in Hawaii", *entity extraction* determines that string "Obama" is a person name, and that "Hawaii" is a location. *Entity linking* goes one step further, determining that string "Obama" actually refers to the entity (i.e., data instance, also called a node) "President Barack Obama" in a KB, and that string "Hawaii" refers to the entity "State of Hawaii" in the same KB.

Entity linking is widely used in information extraction [19, 20], information retrieval [21, 22], and content analysis [23, 24], among many other applications. For example, named entities extracted by information extraction systems are often ambiguous, and can benefit from disambiguation by linking them to a KB.

Coreference resolution. Given a set of strings that are mentions, pronouns, and noun phrases in a text document, this problem partitions the strings into groups such that all strings in the same group refer to the same real-world entity [14]. For example, given the text document in Figure 5, the three strings in blue (with subscript 1) refer to Harry Potter and form a single group, while the three strings in red (with subscript 2) refer to the Hogwarts magic school and form another group. Variations of this problem include anaphora

Table A

Name	City	Age
Dave Smith	Altanta	18
Daniel Smith	LA	18
Joe Welson	New York	25
Charles Williams	Chicago	45
Charlie Williams	Atlanta	28

Table B

Name	City	Age
David Smith	Atlanta	18
Joe Wilson	NY	25
Daniel W. Smith	LA	30
Charles Williams	Chicago	45

Figure 3: An example of entity matching between two tables, with the matches indicated by red arrows.

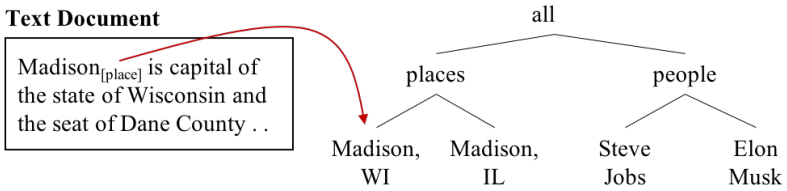

Figure 4: An example of entity linking from a text document to a knowledge base

Hagrid officially invites **[Harry]$_1$** to attend **[Hogwarts]$_2$**, a famous **[magic school]$_2$** in Scotland. **[There]$_2$** the **[young wizard]$_1$** meets **[his]$_1$** closest two friends: Ron and Hermione.

Figure 5: An example of coreference resolution

and cataphora [25]. See [26, 14, 27] for surveys, tutorials, and book chapters on coreference resolution.

Coreference resolution is related to entity matching, but differs in three ways. First, it operates on short strings (e.g., mentions, pronouns). In contrast, entity matching operates on tuples with multiple attributes (e.g., name, age, salary, location). Second, the strings in coreference resolution come from (typically short) text spans that appear in the same document, and thus share similar contexts. Finally, entity matching only determines if any two given tuples match, whereas coreference resolution goes beyond matching, to assign the input strings into groups.

Coreference resolution is an important task in *natural language processing* (NLP), with applications in question answering [28, 29, 30], machine translation [31, 32], and text summarization [33, 34]. For example, in [33] a fuzzy noun phrase coreference system has been used as a core part for a text summarizer.

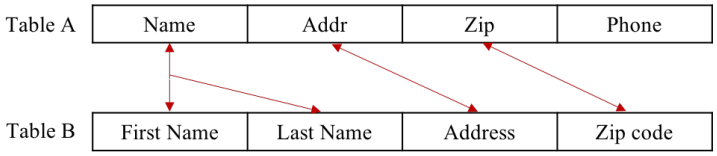

Figure 6: An example of schema matching

Schema and ontology matching. Schema matching finds semantic correspondences, called

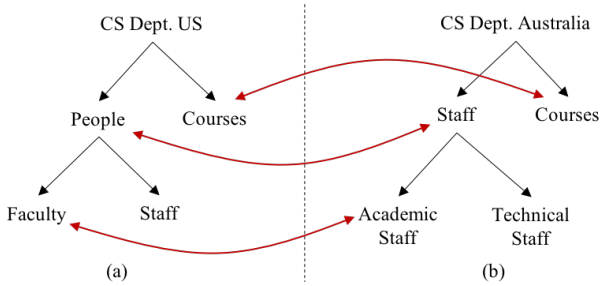

Figure 7: An example of ontology matching

matches, between the attributes of database schemas [35]. Examples of 1-1 matches are "Addr = Address" and "Zip = Zip code". More complex matches include "Name = concat(First Name, Last Name)" (see Figure 6).

Ontology matching is a similar problem of finding semantic matches between the concepts of two ontologies, such as "People = Staff", "Courses = Courses", and "Faculty = Academic Staff" (see Figure 7) [36]. Note that here complex matches are also possible, e.g., "Faculty = union(Research Staff, Teaching Staff)".

See [35, 37] for surveys on schema matching, [38] for a survey on ontology matching, and [39] for a survey on schema/ontology matching.

Both schema matching and ontology matching are fundamental tasks in data integration [8]. Ontology matching plays a critical role in the vision of the Semantic Web [36], which publishes data that conform to ontologies, thus raising a need to establish matches among these ontologies. Building knowledge bases often requires merging data from multiple similar ontologies, which in turn requires ontology matching.

2.4. Finding semantic equivalence relationships

Semantic text similarity. Most works that find semantic equivalence between two schema/data constructs have focused on semantic text similarity, i.e., decide whether two given text snippets semantically convey the same meaning [40, 41]. For example, the two sentences "He is smart" and "That is a wise man" are semantically similar despite sharing no common word other than "is", whereas the two sentences "A cat is chasing a mouse" and "A cat is chasing a dog" are not that similar despite sharing all but one words.

As another example, the site Quora (quora.com) lists many popular questions and answers on a wide range of topics. Deciding if two questions are semantically the same (and thus should be merged) is an important need for Quora.

Semantic text similarity is a core research topics in NLP, with many real-word applications. Many challenges and evaluations (e.g., the SemEval task series on semantic textual similarity, see alt.qcri.org/semeval2017) have been held to help advance this topic. See [41, 42] for a survey and a tutorial, and see [40] for a pilot task on semantic text similarity.

2.5. Finding more complex semantic Relationships

Textual entailment. Given two text snippets (typically two sentences), this problem determines if their meanings are semantically independent, contradictory, or in an entailment relationship, where one snippet (called the premise) can induce the meaning of the other

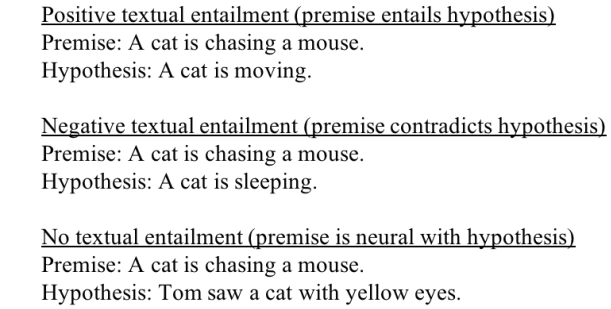

Figure 8: Examples of textual entailment

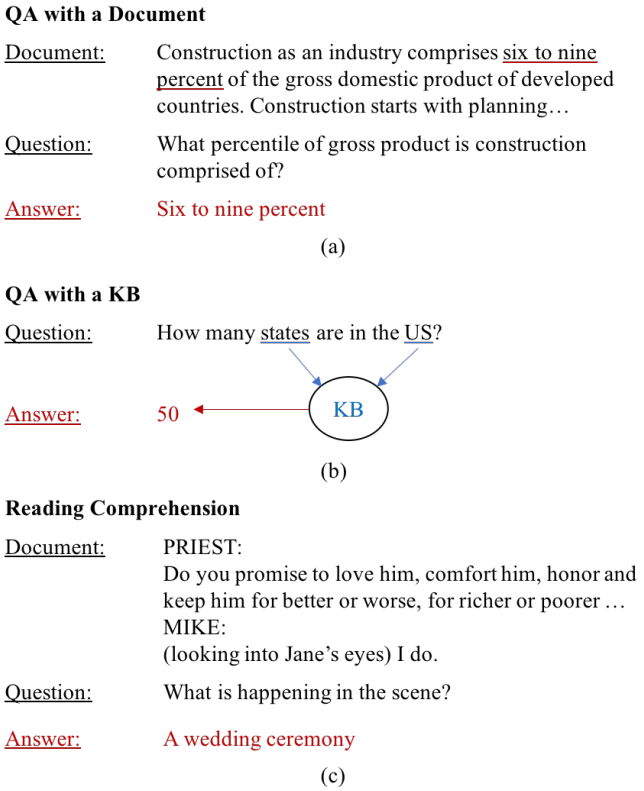

Figure 9: Examples of question answering

(called the hypothesis) [43, 44]. Figure 8 shows three such examples. In the first example, "A cat is chasing a mouse" implies "A cat is moving". In the second example, "A cat is chasing a mouse" contradicts "A cat is sleeping", and so on.

Textual entailment is also known as *recognizing textual entailment (RTE)* and *natural language inference (NLI)*. See [43] for a survey, [44] for a book chapter, and [45] for a tutorial. Similar to semantic text similarity, many competitions have been held for textual entailment, e.g., the PASCAL RTE challenge series (aclweb.org/aclwiki/Textual_Entailment_Resource_Pool).

Textual entailment has been used in machine translation evaluation [46] and question answering [47, 48], among others. For example, [47] shows that textual entailment information can be used to either filter or rank answers returned by a question answering system to improve the accuracy.

Question answering (QA). This is the problem of answering natural language questions automatically. Existing works have considered three main problem settings:

- *QA with a document.* Given a text document and a question, select a text span from the document that best answers the question [49, 50, 51, 52, 53]. See Figure 9.a for an example.
- *QA with a knowledge base.* Given a KB and a question, find the answer from the KB [54, 55, 56, 57]. See Figure 9.b for an example.
- *Machine reading comprehension.* Recent work has also examined machine reading comprehension [58, 59], which answers a question given a document (e.g., a description on an object, a narrative story, or even a dialog between two persons). This can be viewed as a more advanced form of QA with a document, as the answer may not be a span in the document. The system has to reason about the events and entities in the document, and relationships among them, in order to find the correct answer. See Figure 9.c for an example.

In this survey we focus on the first two problem settings, which have been addressed by the majority of QA works. To solve them, existing work typically performs two steps: generating candidate answers, and then matching each candidate to the question to pick the best answer.

The second step is also known as *answer selection* [60, 61, 62, 63, 64, 65], and can be formulated as follows: Given a question and a pool of answer candidates, select the best answer from the pool. We view this as a semantic matching problem (which discovers that an answer is in the relationship "is-answer-of" with a question) and will focus on this answer selection problem in this survey.

Question answering is a long-standing task in NLP. Various workshops and tracks have been held (e.g., TREC QA track), and public datasets (e.g., SQuAD [51], WikiQA [66], InsuranceQA [67], etc.) have been released. QA has many important real-world applications, e.g., playing a key role in Amazon Alexa, Google Assistant, Apple Siri, etc. [68].

See [69] for a general QA survey, which classifies QA from different views. See [51] for a short review on the text span setting of QA and a description of the well-known SQuAD dataset. See [56, 57] for surveys on QA with KBs. Additional QA materials can be found in [70, 51].

Semantic tagging. Given a schema/data construct and a set of controlled phrases, this problem tags the construct with a small set of phrases that best describes the construct, i.e., the topics that it is about [6, 71]. The construct can be a text document, a text snippet, a tweet, a table, etc. The controlled phrases can be manually created, or automatically obtained from an ontology, e.g., taken to be the list of the names of all concepts in an ontology.

For example, given the tweet "Obama gave an immigration speech while on vacation in Hawaii", we may tag it with "politics", "tourism", "vacation", "President Obama", "immigration", and "Hawaii". Several works have automatically recommend such tags (called hashtags) to Twitter users [72, 73]. See [74] for a survey on social tagging techniques. A related problem is tagging images so that they can be easily found by image search engines [75].

Semantic tagging is important for applications in data discovery and recommendations, e.g., [75, 71]. As a concrete example, consider the Environmental Data Initiative website at

environmentaldatainitiative.org, which is a data lake for the environmental research community. Numerous researchers in this community have submitted more than 43K data packages to this data lake, where each package may consist of several CSV tables. They then manually tag each package with phrases obtained from several environmental ontologies (e.g., LTER, ENVO, ENV-THES), so that users of this data lake can easily find the desired tables for a particular research task.

See [76] for a paper that describes a similar problem regarding a data lake in biomedicine. That paper also describes an automatic non-DL approach to automatically tag each given table in the data lake.

3. DEEP LEARNING SOLUTIONS FOR SEMANTIC MATCHING PROBLEMS

We now describe how deep learning has been applied to the semantic matching problems described in Section 2. In particular, we cover entity linking, textual entailment, semantic text similarity, question answering, then entity matching, in that order.

We choose this order because it allows us to group related DL techniques and to explain them in increasing order of complexity. We do not cover schema/ontology matching and tagging because as far as we can tell, there has been very little if any published major work that applies DL to these problems (clearly an opportunity for future research).

For each semantic matching problem, we summarize the basic ideas underlying DL solutions, then describe 1-2 representative solutions in detail. As we will see, even though the problems look quite different, their DL solutions share many commonalities. In particular:

- Virtually all solutions use word embeddings as the input. They rely on the expressive power of word embeddings and neural networks to learn features and their interactions, thereby removing the need for manual labor-intensive feature engineering.
- Most solutions share three key steps: summarizing the two input constructs (e.g., two sentences or two tuples), comparing them to create a comparison vector, then classifying this vector to make a match/no-match prediction.
- The solutions heavily use attention mechanisms. This may be because semantic matching compares two constructs, and attention to get cross-construct information is helpful.
- Finally, the solutions commonly combine multiple techniques to build hybrid DL models.

3.1. Entity linking

Entity linking links mentions in a text document into a KB, e.g., the mention "Madison" in "Madison is the capital of Wisconsin" to the entity (i.e., data instance) "Madison, WI" in DBPedia [17] (rather than to "Madison, CA" or "Madison Square, NY"). Many works have applied DL to entity linking [77, 78, 79, 80, 81, 82, 83, 84, 85]. They often use the following ideas:

- Since the text mentions (to be linked) are short, traditional non-DL approaches need to create multiple task- or language-specific "high-quality" features, to maximize linking accuracy. This is very expensive. DL approaches use word embeddings to avoid such expensive feature engineering.

- These DL works exploit multiple types of context information, such as words surrounding the mention, document context, and the structure of the knowledge base.
- They may perform collective entity linking to capture topical coherence among the mentions. For example, if a mention "Python" in a document links to the entity A = "Python (the programming language)" in a KB, then a mention "Pig" in the same document is more likely to link to the entity B = "Pig (the Big Data language)", which is in the vicinity of entity A, instead of linking to the entity "Pig (the animal)".
- They may use attention mechanisms to capture the dependencies in the context.

Many works (e.g., [77, 78, 79, 80, 82, 85]) employ a DL approach that consists of the following three steps:

- *Context summarization.* Given a text mention, convert its context (e.g., mention text, words surrounding the mention, whole document, etc.) into a summarization, which is an embedding vector. Given a candidate entity in the KB, convert its context (e.g., entity name, entity type, entity description, etc.) into another summarization, which is another embedding vector. Different networks can and have been used in this step, including CNNs, RNNs, tensor networks, etc.
- *Comparison vector generation.* Generate a comparison vector by merging or comparing the vectors (e.g., embeddings) of the mention and the entity candidate.
- *Classification.* Apply a classifier (e.g., using a feed-forward NN) to the comparison vector, to determine if the mention links to the entity.

In the above approach, we only compare a mention with a corresponding entity candidate to determine the similarity. This is known as a local approach where each mention is handled independently. The work [77, 82] extended this to perform collective linking, i.e., trying to link multiple mentions all at the same time, to exploit the topical coherence among the mentions (e.g., linking "Python" and "Pig" to the KB entities falling under the same topic, as discussed earlier).

We now describe CNN-EL [78] and NTN-EL [85], two recent well-known works that are representative of the above three-step DL approach.

CNN-EL. The work [78] proposed a CNN-based model called CNN-EL to capture the semantic correspondence between a mention and a target KB entity. CNN-EL employs two ideas. First, semantics at different granularities are useful for determining the topics of entities for linking. Second, CNNs can convert the information at each granularity into a meaningful topic vector.

Specifically, let M be a text mention, where m is the text of the mention (e.g., "Madison"), c is the context, i.e., the words surrounding the mention (e.g., five words before and five words after), and d is the entire document that contains the mention. Let E be a target entity in the KB, where t is E's title and a is E's description (e.g., the Wikipedia article associated with E). Then the CNN-EL is a log-linear model that learns the conditional distribution over the entity given the mention

$$P(E \mid M) \propto \exp(w^{\top} f_C(M, E; \theta)). \tag{1}$$

Here $f_C(M, E; \theta)$ is the feature vector that contains the six feature values from CNNs with parameter θ. It is generated as follows (see Fig. 10).

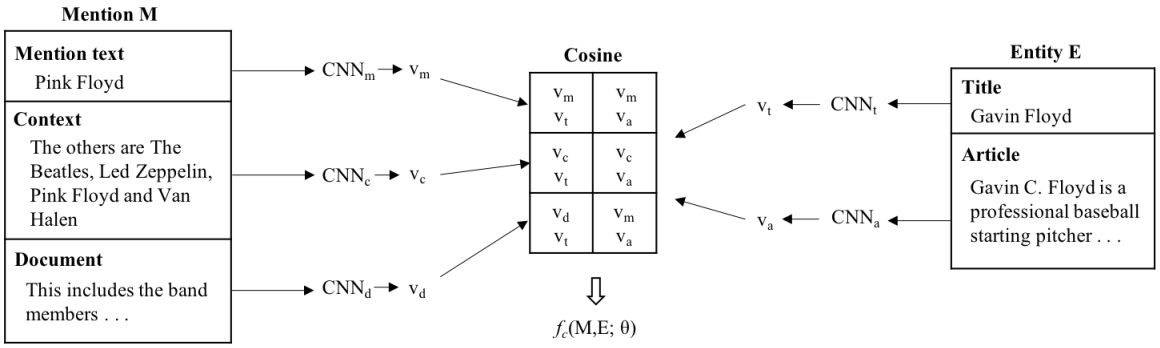

Figure 10: The CNN-EL model architecture

First, for the input text at each granularity, a separate CNN will be used to produce a summarization vector. As shown in Fig. 10, we have CNN_m for the mention text, CNN_c for the mention context, etc. In total, five different CNNs are used.

To generate the summarization, we proceed as follows. Consider summarizing the mention text t as an example. Suppose t is a sequence of n words. Then the first step is to convert them into the word embeddings $w_1, w_2, \ldots, w_n$ using word2vec, where each embedding $w_j \in \mathbb{R}^{d_w}$. Next, a convolution filter $F_m \in \mathbb{R}^{k \times l d_w}$ takes each consecutive l words and produce a k dimensional summary $s_j = F_m w_{j:j+l}$. Then each summary s_j will be sent through ReLU activation. Finally, all results are combined using a sum-pooling operation to get the summary vector v_m. That is,

$$v_m = CNN_m(w_{1:n}) = \sum_{j=1}^{n-l} \max\{0, F_m w_{j:j+l}\}. \tag{2}$$

Note that $w_{i:j}$ is the concatenation of embeddings of words i to j, and max is an element-wise operator.

Once each input text has been summarized into a vector (see vectors v_m, v_c, etc. in Figure 10), we compare all vector pairs across the mention and the target entity using the Cosine similarity measure. This produces six values as shown in the middle of Fig. 10. These six values form the feature vector $f_C(M, E; \theta)$.

Besides the feature values from CNNs, the authors show that the model can achieve better accuracy when integrating some other signals (features), which are created using a prior non-DL work [86].

NTN-EL. The work [85] proposes NTN-EL, which stands for "Neural Tensor Network for Entity Linking". This model is similar to CNN-EL in that context information on both mention and entity is utilized to obtain better summarization vector.

NTN-EL differs from CNN-EL in that rather than generating a summarization vector for text at each granularity with a separate CNN, it uses the Neural Tensor Network (NTN) [87] to combine the text m and the context c of a mention to generate a single summarization vector, and similarly it generates a single summarization vector on the entity side. Then the cosine similarity of the two vectors will be used as the score indicating whether the mention links to the entity or not. We briefly now describe the model.

The input of the model consists of (1) the mention part $M = (m, c)$ with the mention text m and the immediate context c (e.g., five words before and five words after the mention),

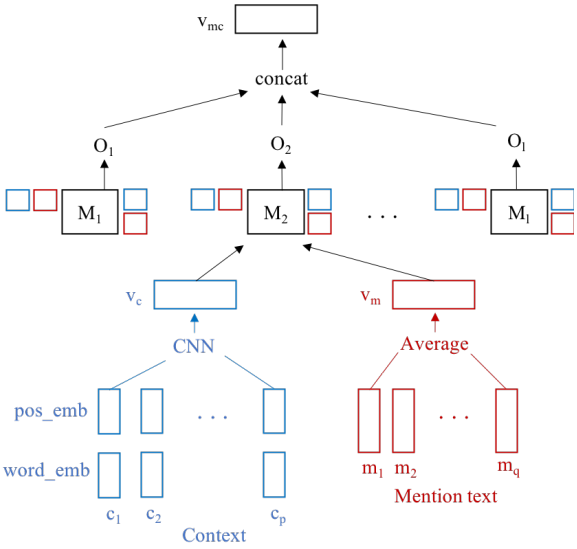

Figure 11: Summarizing the text and context of a mention into a vector v_{mc} in the NTN-EL model.

and (2) the entity part $E = (t, k)$ with the entity title t and the entity class k (e.g., a word or a phrase provided in the infobox of a knowledge base). To predict whether M is linked to E, the model proceeds as follows.

First, a summarization vector v_{mc} is generated condensing the information of M. This is done in three steps (see Figure 11).

1. We first summarize the context c to a vector v_c using a CNN. Assuming that a closer context word might be more informative for disambiguating a mention, the authors also incorporate a position embedding for each context word. So the final embedding of a word (as the input to the CNN) is the concatenation of the original embedding from word2vec and the positional embedding (see the blue boxes in the lower-left corner of Figure 11).
2. Next, a summarization vector v_m is generated for m, the mention text. This text is usually very short, so v_m is simply the average of all word embeddings in m (see the red boxes in the lower right corner of the figure).
3. Finally, we combine v_c and v_m into a final summarization vector v_{mc} using an NTN. An NTN is a list of bilinear layers (i.e., matrices) $T = [M_1, M_2, ..., M_l]$ that takes as input two vectors and outputs a composite vector. Specifically, suppose $[v_c; v_m] \in \mathbb{R}^n$, which is the concatenation of v_c and v_m, then each $M_i \in \mathbb{R}^{n \times n}$ maps $[v_c; v_m]$ into a scalar o_i using a bilinear operation

$$o_i = [v_c; v_m]^\top M_i [v_c; v_m]. \tag{3}$$

   This is illustrated in Figure 11. Note how each M_i is matrix, which participates in a matrix operation with the blue matrix v_c and the red matrix v_m. The vector v_{mc} is then the concatenation of all o_i's to a vector of size l: $v_{mc} = [[v_c; v_m]^\top M_i [v_c; v_m]]_{1:l}$ (see Fig. 11).

Once v_{mc} has been computed, we compute v_e, which summarizes the entity title and entity class, in a similar fashion. Finally, we compute the cosine similarity between v_{mc} and v_e as the score indicating whether M should link to E.

Discussion. So far we have discussed how the DL models are constructed in the works CNN-EL and NTN-EL. Once this is done, the models are trained, i.e., their parameters computed, using labeled training data. Applying the models to new data is a bit more involved. Suppose in the new data we have 1,000 mentions and 100,000 KB entities. When trying to link a mention, we cannot possibly consider all 100,000 entities, as this will become very expensive.

As a result, typically a *blocking step* is carried out to prune the entities, so that for each mention M, we consider only a relatively small set C of entities (e.g., we may consider only entities whose names share at least one word with the mention text). We then apply the above DL models to compute a similarity score between mention M and each entity in the set C, then select the entity with the highest similarity score, or we pair M with each entity in the set C, then apply a classifier (which is typically the last step in applying the DL models) to predict link/no-link.

Most DL works for semantic matching so far do not discuss the blocking step (with the exception of some works in entity matching). As a result, in this survey we also do not discuss this step in detail.

Additional work. We now briefly discuss additional DL work for entity linking. The work [80] proposes a joint encoding model for the entities that utilizes more resources besides the ones used in CNN-EL and NTN-EL, such as fine-grained entity type information. The work [79] extends attention mechanisms to consider not only the input text span but also the surrounding context windows. The works [83, 84] examine cross-lingual entity linking using deep learning. Another line of work incorporates global information to make topical coherent links (i.e., performing collective linking) [77, 82]. Most of the above approaches assume the mentions in the documents are given in advance. The work [81] proposes an end-to-end entity linking solution that jointly discovers and links entities in a text document to a KB based on LSTMs and attention.

3.2. Textual entailment

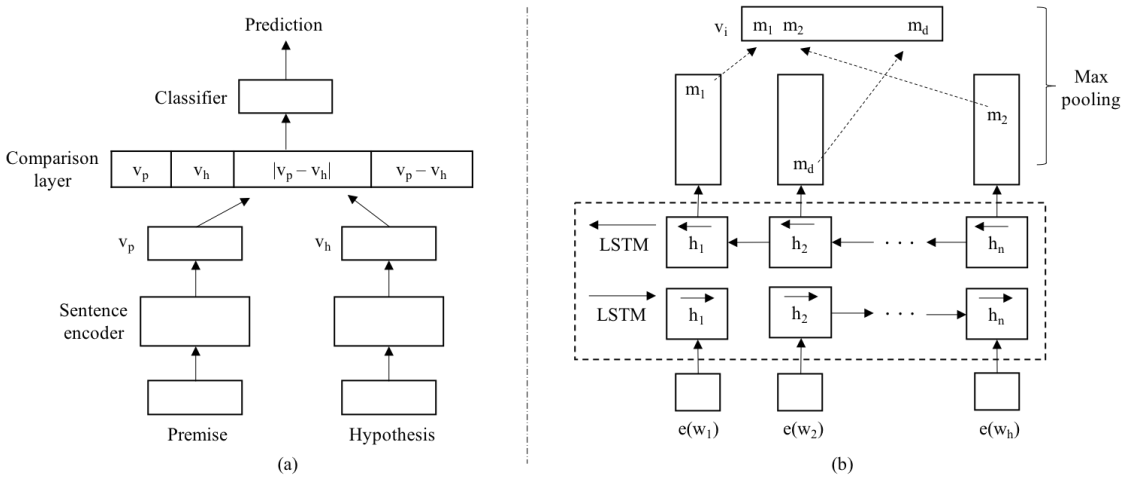

Figure 12: (a) The BiLSTM-MaxPool architecture, which consists of the sentence encoder, the comparison layer, and the classifier; (b) The architecture of the sentence encoder.

Given two text snippets, which are often two sentences, textual entailment determines if one snippet (called the premise) entails (i.e., implies) the other (called the hypothesis), or contradicts it, or is neutral with it [43, 44]. For example, as shown in Fig. 8, "a cat is chasing a mouse" entails "a cat is moving", contradicts "a cat is sleeping", and is neutral with "Tom saw a cat with yellow eyes".

Many works have applied DL to textual entailment [88, 89, 90, 49, 50, 91, 92, 93, 94, 95, 65]. Most works use a DL approach of three steps: sentence encoding (i.e., summarization), sentence comparison, and classification.

1. *Sentence encoding (i.e., summarization).* Convert each sentence into a summarization vector using word embeddings and NNs.
2. *Sentence comparison.* Combine the two summarization vectors into a comparison vector, by concatenating the two summarization vectors, or performing element-wise multiplication or element-wise absolute difference computation, etc.
3. *Classification.* Predict whether the two sentences are in an entailment, or contradictory, or neutral relationship, typically by applying a feed-forward NN to the comparison vector.

In the above approach, the first two steps are most complex, and many solutions for them have been developed. These solutions fall into four groups:

1. The first group [49, 95] encodes sentences using common DL building blocks such as RNNs, CNNs, pooling, mixture, etc.
2. The second group [88, 90] takes into account the syntactic information of each sentence (e.g., the sentence structure) and uses Recursive Neural Networks such as Tree-LSTMs to encode sentences.
3. The third group [50, 93, 94] relies on the attention mechanism for the encoding, by finding cross-sentence alignment or performing self-attention for each sentence, to focus on the important part of the sentences.
4. The last group [89, 92, 65] combines methods used by the above three groups, e.g., combining LSTM and attention.

It is worth noting that many work mentioned above can also be used and have actually been evaluated (in the papers) for other types of semantic matching problems, such as semantic text similarity [49, 50, 94], question answering [50], etc., with possibly slight modifications (e.g., on the output layer).

We now describe two well-known works on textual entailment. The first work falls into the first group described above. It builds a bi-directional LSTM followed by max-pooling. We will refer to it as BiLSTM-MaxPool [49]. The second work falls into the third group, and uses a decomposable attention model, DAM [93].

BiLSTM-MaxPool. The original goal of the work [49] is to generate universal sentence embeddings. However the proposed model is based on the textual entailment setting, and trained on textual entailment datasets (but the paper notes that this model can also be applied to other semantic matching problems). After exploring various NN combinations, the paper shows that BiLSTM with max-pooling achieves the best accuracy on textual entailment (and also on semantic text similarity and paraphrase detection). We now describe the BiLSTM-MaxPool model in the textual entailment setting.

The BiLSTM-MaxPool model takes as input the premise and hypothesis sentences, and conducts a three-class classification to determine whether they are in entailment, contradiction, or neutral relationship. It consists of three modules: The sentence encoder, the comparison layer, and the classifier.

Fig. 12.a shows the working of these modules. The first module, the sentence encoder, converts the two sentences into two summarization vectors $\vec{v}_p$ and $\vec{v}_h$ (we will discuss this in detail soon). The comparison layer then generates a comparison vector that combines the information of the two: $\vec{c} = [\vec{v}_p, \vec{v}_h, |\vec{v}_p - \vec{v}_h|, \vec{v}_p * \vec{v}_h]$. Thus, this vector consists of the concatenation of $\vec{v}_p$, $\vec{v}_p$, the element-wise absolute difference, and the element-wise product of the two. The last module, the classifier, applies a feed-forward NN to vector $\vec{c}$ to predict one of the three classes: entailment, contradiction, or neutral.

We now describe the first module, the sentence encoder, in more detail. This module takes as input a sentence, which is a sequence of words, and outputs an embedding vector. It proceeds in three steps (see Fig. 12.b).

1. Suppose a sentence s has n words $w_1, w_2, \ldots, w_n$, in the first step a word-embedding lookup table is built to convert each word w_i into an embedding $\vec{e}(w_i)$. This lookup table can be built using word2vec, GloVe or fastText.
2. Then a BiLSTM is used to generate a hidden state $\vec{h}_i$ for each word w_i. Specifically, we have a forward LSTM and a backward LSTM to generate hidden states in the opposite directions
$$\begin{aligned} \overrightarrow{h_i} &= \overrightarrow{LSTM}(\overrightarrow{h_{i-1}}, \vec{e}(w_i)) \\ \overleftarrow{h_i} &= \overleftarrow{LSTM}(\overleftarrow{h_{i+1}}, \vec{e}(w_i)) \end{aligned} \tag{4}$$
The final hidden state $\vec{h_i} = [\overrightarrow{h_i}, \overleftarrow{h_i}]$ is the concatenation of the two for each word w_i.
3. Given the hidden states $\vec{h_1}, \vec{h_2}, \ldots, \vec{h_n}$, the third step is to apply a max-pooling operation to get the sentence embedding $\vec{v}_s$. In particular, assuming each $\vec{h_i} \in \mathbb{R}^d$, then $\vec{v}_s \in \mathbb{R}^d$ with value at the j-th index $\vec{v}_s[j] = \max\{\vec{h}_1[j], \ldots, \vec{h}_n[j]\}$, for $0 \le j \le d-1$. Note that only one shared encoder will be created for both the premise and hypothesis, therefore the encoder is a Siamese network [96].

DAM. The work [93] proposes DAM, a decomposable attention model for textual entailment that relies only on the attention mechanism. Attention allows the model to find parts that align across the premise and the hypothesis, on the assumption that this will make the classification more accurate. The model does not use the sequential information of the sentences, like LSTMs.

Similar to BiLSTM-MaxPool, DAM takes as input the premise and the hypothesis, and outputs a prediction indicating the relationship. DAM consists of four modules: Embedding lookup, attending, comparing, and aggregating.

First, the embedding lookup table converts each word in the input into an embedding vector. Let the premise and the hypothesis be $\vec{a} = [\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_m]$ and $\vec{b} = [\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n]$, respectively, where each word $\vec{a}_i$ and $\vec{b}_j$ are in the embedding format.

Second, the attending module conducts decomposable attention. It proceeds in two steps:

1. For each word pair $(\vec{a}_i, \vec{b}_j)$, we calculate an attention score
$$e_{ij} = F'(\vec{a}_i, \vec{b}_j) = F(\vec{a}_i)^\top F(\vec{b}_j). \tag{5}$$

This score reflects the correlation between the two words. A higher score means a higher chance that the words are related to the same concept. Here we can see the reason why this model is called "decomposable": the attention function F' (which takes two inputs) has been decomposed into the dot product of a shared function F (which only takes one input).

2. Then based on the attention scores, for each word $\vec{a}_i$, we compute a weighted average over the words in $\vec{b}$ (we compute a similar weighted average for each word $\vec{b}_j$)

$$\begin{aligned} \boldsymbol{\beta}_i &= \sum_{j=1}^{n} \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \vec{b}_j \quad 1 \le i \le m, \\ \boldsymbol{\alpha}_j &= \sum_{i=1}^{m} \frac{\exp(e_{ij})}{\sum_{k=1}^{m} \exp(e_{kj})} \vec{a}_i \quad 1 \le j \le n. \end{aligned} \tag{6}$$

Here each $\boldsymbol{\beta}_i$ is softly aligned to $\vec{a}_i$, representing how the meaning of $\vec{a}_i$ is distributed in the words of $\vec{b}$ (a similar process is carried out for $\boldsymbol{\alpha}_j$).

In the third step, the comparing module takes as input a word-alignment pair (i.e., each $(\vec{a}_i, \boldsymbol{\beta}_i)$ or $(\vec{b}_j, \boldsymbol{\alpha}_j)$) and outputs a similarity vector, using a feed-forward NN. Specifically,

$$\begin{aligned} \vec{c}_{a,i} &= G(\vec{a}_i, \boldsymbol{\beta}_i) \quad 1 \le i \le m \\ \vec{c}_{b,j} &= G(\vec{b}_j, \boldsymbol{\alpha}_j) \quad 1 \le j \le n \end{aligned} \tag{7}$$

Here G represents a feed-forward NN. The reason the comparing module is useful is that the attention step only finds conceptually related alignment, but it does not measure the semantic similarity, e.g. a pair $(\vec{a}_i, \boldsymbol{\beta}_i)$ represents similar meaning or contrary. For instance, it finds out that "red" and "blue" are both colors and thus are conceptually related, but it cannot tell that they are different colors, and thus are not similar.

Finally, the aggregating module aggregates the similarity vectors for each input sentence, and predicts a label using a feed-forward NN as a classifier. Specifically,

$$\begin{aligned} \vec{c}_a &= \sum_{i=1}^{m} \vec{c}_{a,i}, \quad \vec{c}_b = \sum_{j=1}^{n} \vec{c}_{b,j} \\ \hat{y} &= H(\vec{c}_a, \vec{c}_b) \end{aligned} \tag{8}$$

where H is the feed-forward NN, and $\hat{y}$ is the prediction.

It is worth noting that even though DAM was originally invented for textual entailment, it can be applied to other tasks, e.g., entity matching [97].

Additional work. Many works have applied DL to textual entailment. A line of work builds on BiLSTMs combined with different pooling and attention operations. The work [92] uses BiLSTM and inner-attention within the hypothesis. [95] extends BiLSTM-MaxPool model and proposes hierarchical BiLSTM max-pooling architecture. [89] uses BiLSTM with generalized pooling. Incorporating the sentence's syntactic information, [88, 90] proposes Tree-LSTM based methods. Similar to DAM, another line of work [50, 91, 94] suggests using only attention mechanisms with simple feed-forward NNs.

3.3. Semantic text similarity

In this problem we determine if two text snippets, which are typically two sentences or sentence fragments, semantically convey the same meaning [40, 41]. For example, "he is smart" and "he is a wise man" are semantically similar, whereas "a cat is chasing a mouse" and "a cat is chasing a dog" are not.

Many works have applied DL to this problem [49, 50, 98, 99, 100, 101, 102, 94, 65]. As semantic text similarity is closely related to textual entailment, many methods for textual entailment can also be applied to semantic text similarity [49, 50, 93, 94, 65]. In fact, many papers conduct empirical evaluation on both textual entailment and semantic text similarity [49, 50, 94, 65] (this includes the paper on the BiLSTM-MaxPool model described in Subection 3.2). There are work specifically focusing on semantic text similarity [98, 100, 101, 102]. Even these works share a similar model architecture to textual entailment: sentence encoding, sentence comparison, and similarity prediction (sometimes the comparison layer and the prediction layer are combined [100, 101]).

We now briefly describe a well-known work in semantic text similarity: MaLSTM [100], which is simple yet performs well on many datasets [100].

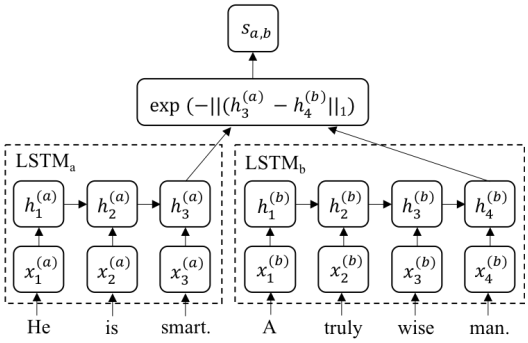

Figure 13: An illustration of how the MaLSTM model works, given two sentence fragments as the input.

MaLSTM. MaLSTM [100] represents Manhattan LSTM. As the name suggests, the model is based on LSTMs and Manhattan distance. Given a pair of sentences, the model predicts a scalar score indicating the level of similarity between the two sentences. To do so, it performs sentence encoding and then sentence comparison (see Figure 13).

Given a sentence pair $\vec{a} = [\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_m]$ and $\vec{b} = [\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n]$, where each word is in the format of embedding, a Siamese LSTM is used to encode each of the $\vec{a}$ and $\vec{b}$ into a fixed-size vector, by selecting the last hidden state $\vec{h}_{a,m}$ and $\vec{h}_{b,n}$ respectively.

Then the following similarity function is used to produce a similarity score $s_{a,b}$

$$s_{a,b} = \exp(- \parallel \vec{h}_{a,m} - \vec{h}_{b,n} \parallel_1) \in [0, 1]. \tag{9}$$

Here $\| \cdot \|_1$ is the Manhattan metric (i.e., the l_1-norm), which is the element-wise sum of the input vector. As pointed out in the paper, the Manhattan metric slightly outperforms other reasonable alternatives such as cosine similarity.

Additional work. The work [101] proposes a similar Siamese network that uses a character-level BiLSTM. It also uses a contrastive loss function in training so that matching sentences would be nearby in the vector space. [102] builds upon [101] and proposes training the network by jointly minimizing a logistic loss apart from the contrastive loss. [98, 99] argue that

sentence encoding like LSTM (as used in the aforementioned work) is too coarse to capture fine-grained word-level information. The work [98] proposes instead to use CNN to extract sentence features at multiple levels of granularity, and to compare sentence representations at several granularities using multiple similarity metrics. The work [99] proposes a pair-wise word interaction model after the LSTM layer used in [100, 101], to encourage direct comparisons between word contexts across the sentences.

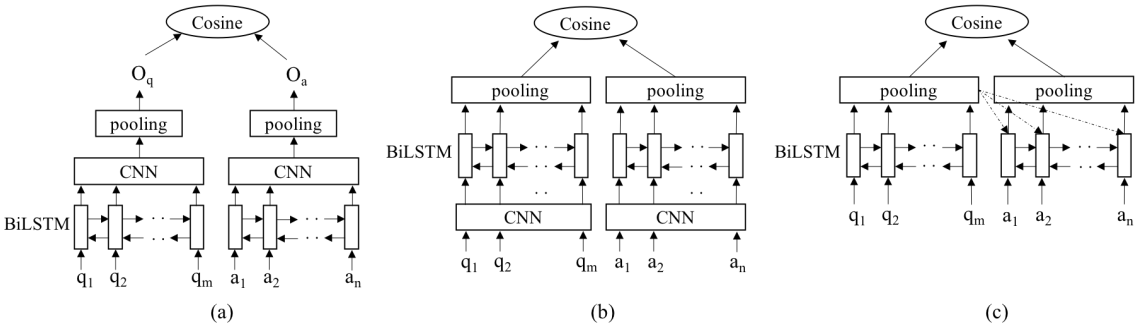

Figure 14: Three QA models: (a) Convolutional-pooling LSTM, (b) Convolution-based LSTM, and (c) Attention-based LSTM.

3.4. Question answering

Question answering (QA) automatically answers questions in natural languages. As discussed in Section 2, several QA settings exist. However, they often share a common task, which is answer selection: given a question and a pool of answer candidates, select the best answer from the pool. In this section we focus on this task.

Many works have applied DL to answer section (see [103] for a recent survey). Most of them (e.g., [60, 61, 62, 63, 64, 65]) use a DL approach similar to those for textual entailment and semantic text similarity. Specifically, given a question and a candidate answer, this approach encodes them as vectors, compares the vectors to obtain a comparison vector, then uses this vector to predict how likely the candidate answer is the correct answer to the question.

There are two main approaches for the encoding step. The first approach encodes the question and the candidate answer independently [61, 63]. There is no explicit interaction between the question and the answer during encoding. The second approach uses attention mechanisms to capture cross-sentence information (here we view both the question and the candidate answer as two sentences). The majority of work in answer selection fall into the second approach (e.g., [60, 62, 63, 64, 65]). Note that some of these works have also been evaluated on textual entailment [65] and semantic text similarity tasks [62, 65].

We now describe [63], which specifically focuses on answer selection. It proposes two model variants that belong to the first approach (i.e., encoding the question and the answer independently). These variants use both CNN and LSTM. It also proposes a model variant that belongs to the second approach. This variant uses attention-based LSTM.

All three models return a score indicating how likely an input candidate answer is the correct answer to an input question. All three use an embedding lookup table to convert each word in the sentences (i.e., the input question and the input answer) into an embedding vector. Let $\vec{q} = [\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_m]$ and $\vec{a} = [\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n]$ be the question and the candidate

answer respectively, where each word in already the embedding format. We now describe the three models in detail.

ConvPool-LSTM. The first model, convolutional-pooling LSTM, applies a convolution-based pooling operation after a BiLSTM layer. Figure 14.a shows the architecture. While LSTM can capture long-range dependencies, it has difficulties capturing local n-gram coherence. So the ConvPool-LSTM model adds a CNN layer after the LSTM layer to keep the local information as well. The resulting model consists of three parts.

The first part is a BiLSTM that takes as input a sentence and output a sequence of hidden states. This is exactly the same as Equation 4. Note that the BiLSTM used here is Siamese, meaning the question(s) and the answer(s) share the same BiLSTM (see the bottom part of Figure 14.a).

The second part is a convolution layer followed by a max-pooling operation. Specifically, suppose $\vec{h}_q = [\vec{h}_{q,1}, \vec{h}_{q,2}, \ldots, \vec{h}_{q,m}]$ are the hidden states for the question, with each $\vec{h}_{q,i} \in \mathbb{R}^h$. Then a list of c filters with kernel size of k (i.e., every k consecutive words in the sentence will be considered for a convolution operation for each filter) are used for the convolution. Let $W_f \in \mathbb{R}^{c \times kh}$ be the convolution parameters. Suppose we rewrite $\vec{h}_q$ in the matrix format to be $H_q \in \mathbb{R}^{kh \times L}$, with L being the output dimension after convolutions (i.e., the number of convolutions). Then we can define the convolution function

$$C = CNN(\vec{h}_q) = Tanh(W_f H_q). \tag{10}$$

Then we apply a max-pooling over C to obtain the final output vector $\vec{o}_q$ with the i-th dimension being $\vec{o}_q[i] = \max\{C_{i,1}, C_{i,2}, \ldots, C_{i,L}\}$. Similarly, for the answer $\vec{a}$ we can obtain the output $\vec{o}_a$. This CNN layer is also Siamese.

Finally, we compute the cosine similarity score of $\vec{o}_q$ and $\vec{o}_a$, and return it as the score that captures how likely it is that the candidate answer is the correct answer to the question.

Conv-LSTM. The second variant, convolution-based LSTM, applies a BiLSTM after the outputs of a convolution layer. Figure 14.b shows the architecture. Compared to ConvPool-LSTM, we can see that this model tries to capture the local n-gram information first, and then extract the long-range dependencies.

Similar to ConvPool-LSTM, Conv-LSTM also consists of three parts. The first part is a CNN layer which is identical to the one used in ConvPool-LSTM except that the input are the word embeddings in a sentence. The second part is a BiLSTM followed by a max-pooling operation.

Suppose $X_q \in \mathbb{R}^{c \times L}$ is the output of the CNN layer, we form a sequence of vectors $\vec{x}_q = [\vec{x}_{q,1}, \vec{x}_{q,2}, \ldots, \vec{x}_{q,L}]$ where each $\vec{x}_{q,i}$ is the i-th column in matrix X. Then $\vec{x}_q$ will be used as the input to the BiLSTM to get a list of hidden states $\vec{h}_q = [\vec{h}_{q,1}, \vec{h}_{q,2}, \ldots, \vec{h}_{q,L}]$. The output $\vec{o}_q$ is generated by max-pooling each dimension of the hidden states: $\vec{o}_q[i] = \max\{\vec{h}_{q,1}[i], \vec{h}_{q,2}[i], \ldots, \vec{h}_{q,L}[i]\}$. We obtain the representation $\vec{o}_a$ for the candidate answer in a similar fashion. Finally, we compute the cosine similarity score between $\vec{o}_q$ and $\vec{o}_a$. Note that both CNN and BiLSTM are Siamese in Conv-LSTM.

Att-LSTM. One drawback of both ConvPool-LSTM and Conv-LSTM is that they generate the summarization vectors for the question and the answer separately, not taking into account the cross-sentence information. So the summarization may not extract crucial content for the prediction. In particular, even the correct answer can be very long and contains lots of words that are not related to the question. In such cases the resulted summarization can

be "distracted" by non-useful information. To address this problem, the same work [63] proposes a third model, attention-based LSTM, which will attend to the question for the answer summarization generation.

This model proceeds in three steps. The first step is to generate the summarization vector $\vec{o}_q$ for the question, using a BiLSTM followed by a max-pooling operation. This step is very similar to the second step in the Conv-LSTM model.

In the second step, the answer summarization vector $\vec{o}_a$ is generating by attending to $\vec{o}_q$. We first apply the same BiLSTM used for the question to generate a list of hidden states $\vec{h}_a = [\vec{h}_{a,1}, \vec{h}_{a,2}, \ldots, \vec{h}_{a,n}]$. Next, for each state $\vec{h}_{a,i}$ we attend to $\vec{o}_q$ to get a weight score indicating how important the hidden state is. Specifically, we compare each $\vec{h}_{a,i}$ to $\vec{o}_q$ to get a comparison vector $\vec{m}_i$

$$\vec{m}_i = W_{am}\vec{h}_{a,i} + W_{qm}\vec{o}_q. \tag{11}$$

Then we get the weight score $s_{a,i}$ for $\vec{h}_{a,i}$

$$s_{a,i} \propto \exp\left(\vec{w}_{sm}^{\top} Tanh(\vec{m}_i)\right). \tag{12}$$

Here W_{am}, W_{qm}, and $\vec{w}_{sm}$ are the attention parameters. Then we get the final weighted hidden states $\widetilde{\vec{h}}_a = [\widetilde{\vec{h}}_{a,1}, \widetilde{\vec{h}}_{a,2}, \ldots, \widetilde{\vec{h}}_{a,n}]$, where each $\widetilde{\vec{h}}_{a,i} = s_{a,i}\vec{h}_{a,i}$. The summarization vector $\vec{o}_a$ is obtained by max-pooling on each dimension of all weighted hidden states.

Once we have $\vec{o}_q$ and $\vec{o}_a$, in the last step we calculate the cosine similarity to obtain the prediction score, similar to the way the previous two models compute this score.

Additional work. The attention in the Att-LSTM model [63] mentioned above only applies on one side, where the question embedding is used for encoding the answer. [60] extends it by creating a global view of the candidate answer and applying another attention for the answer encoding based on that global view. [64] uses a tensor network [87] to model the interactions between the two sentences, and then uses k-max pooling to filter out the k strongest interactions for final prediction. [62] uses both cross-sentence attention to discover fine-grained alignment and intra-sentence attention to emphasize important words from the perspective of semantic composition for sentence encoding. [65] uses a bilateral multi-perspective matching (BiMPM). First, each word in the question (or the answer candidate) is matched to all words in the answer candidate (or the question) from four different perspectives (with four different networks) and form a matching vector. Then the matching vectors for each sentence (i.e., question or answer) are aggregated into a single vector as the sentence embedding. Finally, the two sentence embeddings are used to make a prediction through a feed-forward NN.

3.5. Entity matching

Entity matching finds data records that refer to the same real-word entity, such as (Joe Wilson, New York) and (J. Wilson, NY) [7, 104, 105, 97]. We focus on the common setting where given two tables A and B with the same schema $S = \{a_1, a_2, \ldots, a_n\}$ of n attributes, we need to find matching pairs across A and B (if A and B do not share the same schema, we can apply schema matching first).

Typically entity matching proceeds in two steps blocking and matching. It is often very expensive to compare all pairs in $A \times B$, so the blocking step uses some cheap heuristics to remove obviously non-matching tuple pairs to get a set of candidates (usually much less than

enumerating all pairs). Then the matching step determines if each pair in the candidate set is a match. To our knowledge, as of November 2018 there have been only two published work on entity matching using deep learning: DeepER [106] and DeepMatcher [97]. We now describe both.

DeepER. DeepER covers both blocking and matching. It converts each tuple into an embedding vector, then performs blocking and matching using these vectors. We now describe these steps.

1) *Converting tuples into embedding vectors:* To convert a tuple t into a vector, first, a word-embedding lookup table is used to convert each textual word in t into an embedding vector. Then these vectors are combined into a single vector, using either simple averaging or LSTM-based composition.

- Suppose tuple t has value $t[a_i]$ for attribute a_i. To generate the tuple embedding vector $\vec{v}(t)$ for t using simple averaging, we first calculate an attribute embedding vector $\vec{v}(t[a_i])$ for each a_i. Assuming there are $|a_i|$ words in attribute a_i, we do this by converting each word $w_{i,j}$ in a_i into an embedding vector $\vec{e}(w_{i,j})$ using a lookup table, then calculate the average vector $\vec{v}(t[a_i]) = (\sum_{j=1}^{|a_i|} \vec{e}(w_{i,j}))/|a_i|$. Then the final tuple embedding $\vec{v}(t)$ is the concatenation of all attribute embeddings: $\vec{v}(t) = [\vec{v}(t[a_1]), \ldots, \vec{v}(t[a_n])]$.
- The simple averaging method cannot capture the sequence information of words. To address this, the paper proposes using LSTM to obtain the tuple embedding. Specifically, it first concatenates all attributes into a single string. Assuming the concatenated string has m words $w_1, w_2, \ldots, w_m$ in that order, then it converts each word into an embedding vector, and loops through the embeddings to get the hidden representations

$$h_i = LSTM(\vec{e}(w_i), h_{i-1}). \tag{13}$$

  The last hidden output h_m will be the tuple embedding vector $\vec{v}(t)$.

2) *Blocking:* This step performs blocking using the tuple embeddings. As each tuple is now represented by a high-dimensional vector, the paper uses locality sensitive hashing (LSH) [107] to perform blocking. Given a set of tuples as input, LSH will produce a set of hash blocks, such that if two tuple embeddings are very similar (by cosine similarity), then with high probability they will be put into the same block. This is carried out in three steps.

First, we generate a hash table $g(\vec{v}(t)) = [h_1(\vec{v}(t)), h_2(\vec{v}(t)), \ldots, h_K(\vec{v}(t))]$ with K hash functions each of which takes as input a tuple embedding $\vec{v}(t)$, and outputs a binary vector with K dimension. Each binary vector represents a hash block, and different tuples with the same binary vector will be placed into the same block. Each hash function $h_i(\vec{v}(t))$ is a random hyperplane $\vec{w}_i$ (a unit vector through the origin drawn randomly) in the tuple embedding space. Given the tuple embedding t, each $h_i(\vec{v}(t))$ outputs 1 or -1 as follows

$$h_i(\vec{v}(t)) = \begin{cases} +1 & \text{if } \vec{w}_i \cdot \vec{v}(t) \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{14}$$

Second, we repeat the first step multiple times to obtain a set of L different hash tables $g_1(\vec{v}(t)), \ldots, g_L(\vec{v}(t))$. Since for each hash table we use K hash functions, which reduces the chance that similar tuple pairs fall into the same hash block, we use multiple hash tables to mitigate this problem.

Finally, given a set of tuples (e.g., the union of the two tables to be matched), we pass each tuple through all L hash tables to obtain a list of blocks. Then the candidate set for

matching consists of all tuple pairs that appear together in at least one block (there are pruning strategies to further reduce the candidate set size, see [106]).

3) *Matching:* To classify each candidate tuple pair as a match or no-match, we first convert each pair of tuple embeddings into a comparison vector $sim(\vec{v}(t_1), \vec{v}(t_2))$. Then a feed-forward NN is used to map the comparison vector into a scalar score indicating a match or not.

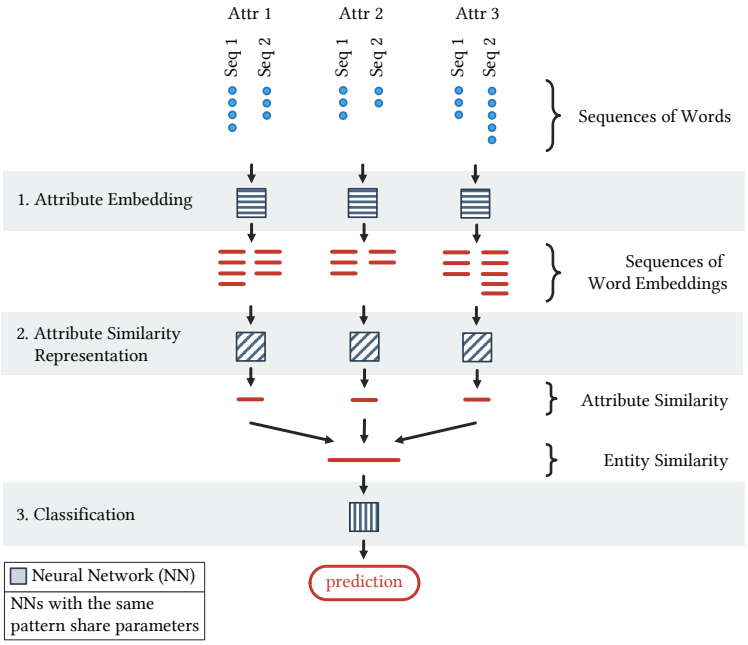

Figure 15: The DeepMatcher architecture

Table 1: Options available for the three modules in the DeepMatcher architecture

Architecture module		**Options**	
Attribute embedding		*Granularity:* (1) Word-based (2) Character-based	*Training:* (3) Pre-trained (4) Learned
Attribute similarity representation	(1) Attribute summarization	(1) Heuristic-based (2) RNN-based (3) Attention-based (4) Hybrid	
	(2) Attribute comparison	(1) Fixed distance (cosine, Euclidean) (2) Learnable distance (concatenation, element-wise absolute difference, element-wise multiplication)	
Classifier		NN (multi-layer perceptron)	

DeepMatcher. Unlike DeepER which covers both blocking and matching, DeepMatcher [97] focuses on the matching step. Specifically, the DeepMatcher authors propose a template architecture for entity matching, as shown in Figure 15. This template consists of three modules: attribute embedding, attribute similarity representation, and classification. The authors provide a set of design choices for each module, as summarized in Table 1. From the different combinations of design choices, they select four DL solutions as representative points

in the design space. These solutions correspond to DL models of varying representational power and complexity.

All four solutions are similar in two ways. First, they use fastText [108] to implement the attribute embedding module, i.e., each textual word will be turned into an embedding vector using fastText. Second, they use a two-layer feed-forward NN (more specifically, fully-connected ReLU HighwayNet[1]; see [109] for details.) followed by a softmax layer to implement the classifier module.

However, the four solutions use different choices for the attribute summarization process, and so are named *SIF*, *RNN*, *Attention*, and *Hybrid*, respectively, for their choice of the attribute summarization part. We now describe the four solutions, focusing on this part.

1) *SIF:* This model is the simplest. For each attribute in a tuple, it uses an aggregate function, specifically a weighted average over the word embeddings to get the attribute summarization. This step is similar to the simple averaging method in DeepER, except that here it calculates a weighted average based on smooth inverse frequency (SIF). Concretely, a weight $f(w)$ is generated for each word w with $f(w) = a/(a + p(w))$ where a is a hyperparameter and $p(w)$ is the normalized unigram frequency of w in the input corpus. Due to its simplicity, the model's performance for matching relies mostly on the expressive power of the attribute embedding and the classifier used.

2) *RNN:* This model uses a bidirectional GRU to summarize an attribute. This is similar to the LSTM summarizer used in DeepER except that here it uses GRUs as the RNN units rather than LSTMs. Compared to SIF, this model takes into account the sequence information and may obtain better representation power. It introduces more parameters (in the GRUs) and therefore is a more complex model.

3) *Attention:* This is an attention model based on the Decomposable Attention Model (DAM) described in Subsection 3.2. For each attribute, it basically executes all steps before the classification step (see Subsection 3.2), to obtain the attribute summarization. Compared to SIF, this model considers cross-tuple information, which makes the model focus on the most important words for predictions. As the attention layer introduces more parameters, this is also a more complex model than SIF.

4) *Hybrid:* This model combines attention and RNN to consider both sequence information within an attribute value and cross-tuple alignment information. The model can be decomposed into three steps: soft alignment, comparison, and aggregation. Figure 16 shows the workflow for the attribute summarization. We now briefly describe each step. We assume a tuple pair (t_1, t_2) and a table schema $S = \{a_1, a_2, \ldots, a_n\}$. For each attribute a_l we will only describe performing summarization on $t_1[a_l]$ with $t_2[a_l]$ as the attended context (we can simply swap t_1 and t_2 for the summarization on t_2).

The first step is soft alignment, which is essentially the decomposable attention step in DAM combined with RNNs. Specifically, suppose $t_1[a_l] = [\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_m]$, and $t_2[a_l] = [\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_k]$ with all words already in the embedding format. We first compute the attention score e_{ij} for each word pair $(\vec{a}_i, \vec{b}_j)$ following Equation 5. Next, a bidirectional GRU, GRU_1,

[1]HighwayNets are used since they speed up convergence and produced better empirical results than traditional fully connected networks across entity matching tasks, especially in the case of small datasets.

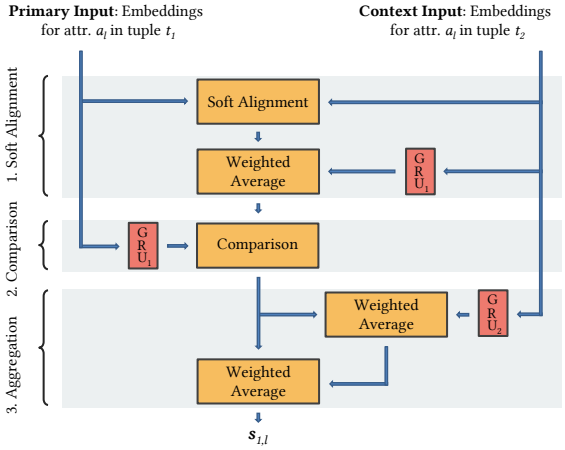

Figure 16: The Hybrid model workflow for attribute summarization

will be applied on $t_2[a_l]$ to get the hidden states: $\vec{h}_2 = GRU_1(t_2[a_l]) = [\vec{h}_{2,1}, \vec{h}_{2,2}, \ldots, \vec{h}_{2,k}]$. Then for each word $\vec{a}_i$, a weighted average vector $\boldsymbol{\beta_i}$ over $\vec{h}_2$ is calculated as the soft alignment for $\vec{a}_i$

$$\boldsymbol{\beta}_i = \sum_{j=1}^{k} \frac{\exp(e_{ij})}{\sum_{p=1}^{k} \exp(e_{ip})} \vec{h}_{2,j}, \quad 1 \leq i \leq m. \tag{15}$$

We can see that Equation 15 is similar to Equation 6 in the original DAM. The difference is that here the weighted average is over the RNN hidden states. This can be viewed as capturing sequence information in attention, which can build phrase-level alignment rather than word-level alignment.

The second step is the comparison step. First, we apply the same GRU_1 (used in step one) to $t_1[a_l]$ to get the hidden states $\vec{h}_1 = GRU_1(t_1[a_l]) = [\vec{h}_{1,1}, \vec{h}_{1,2}, \ldots, \vec{h}_{2,m}]$. Then for each $\vec{h}_{1,i}$, we compare it with the corresponding alignment $\boldsymbol{\beta}_i$ using a feed-forward NN, to get a comparison vector $\vec{c}_{1,i}$.

The last step is to aggregate the comparison vectors from the previous step, to get the attribute summarization vector $\vec{s}_{1,l}$ for $t_1[a_l]$. In this step, the second attention is conducted. Specifically, we first apply a new bidirectional GRU, GRU_2, to $t_2[a_l]$ and take the last hidden output $\vec{h}'$ as the summarization of $t_2[a_l]$. Then, for each comparison vector $\vec{c}_{1,i}$, we concatenate $\vec{c}_{1,i}$ and $\vec{h}'$, and pass it to a feed-forward NN to get a weight score $w_{1,i}$. Essentially, this score represents the importance of $\vec{c}_{1,l}$ by attending to the summary of $t_2[a_l]$. And finally we get the attribute summarization vector for $t_1[a_l]$ as $\vec{s}_{1,l} = \sum_{i=1}^{m} w_{1,i}\vec{c}_{1,i}$.

3.6. Coreference resolution

Coreference resolution takes as input a document with text mentions already identified. It partitions the mentions into groups such that all mentions within a group (and only those mentions) refer to the same real-world entity [14]. As such, it is somewhat different from the works that we have discussed so far, which focus on matching two given data items, rather than partitioning a set of data items.

Many works have applied DL to coreference resolution [110, 111, 112, 113, 114, 115, 116]. As the mentions are typically very short, a major challenge is to generate informative features

and model the correlations among them. By using DL, these works try to take advantage of the expressive power of the embeddings and NNs to learn the features as well as the fine-grained interactions among them, without manual effort. The majority of existing DL works for coreference resolution fall into three groups.

First, the mention-pair models [110, 115] receive a pair of mentions as input and decide if they are coreferent or not. Each entity will be represented as a chain of coreferent pairs of mentions. The models usually treat each mention pair independently, and may produce incoherent coreferences.

Second, the entity-mention model [111, 114] goes one step further than the mention-pair models, to consider the entity-level information. Specifically, the models learn to build the entity clusters directly, where each cluster consists of all coreferent mentions.

The final group is end-to-end models [113, 116]. The previous two groups of models assume the set of mentions in a document are given in advance. This last group of models do not make that assumption. They perform coreference resolution from scratch: first identity all potential mentions and then resolve the coreferences.

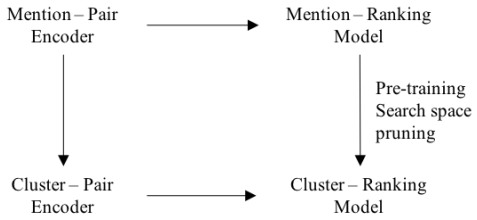

Figure 17: Cluster-CR model architecture

We now describe Cluster-CR, a well-known entity-mention model [111]. Cluster-CR directly builds clusters of mentions, but it also employs a mention-pair model submodule. Understanding Cluster-CR should give a good view into how DL has generally been applied to coreference resolution.

Cluster-CR. A major challenge in coreference resolution is to utilize entity-level information (rather than only comparing mention pairs). For example, given two mention clusters $\{Bill\,Clinton\}$ and $\{Clinton, she\}$, we can determine that they do not refer to the same person. However, given only the mention pair $\{Bill\,Clinton\}$ and $\{Clinton\}$, we cannot determine whether they are coreferent.

The work [111] develops *Cluster-CR*, an NN-based coreference resolution method which operates in an agglomerative clustering fashion. This method builds distributed representations of pairs of coreference clusters, and can take advantage of the entity-level information without many hand-crafted features.

We first briefly introduce the whole workflow, and then zoom into the details of *Cluster-CR*. Suppose a set of mentions in a document have been identified. The model resolves the coreferences by merging mentions into the same cluster if they refer to the same entity. Concretely, the model builds up the mention coreference clusters incrementally, i.e., it will start with each mention as a single cluster, then iteratively merge a pair of clusters in each step. The neural network will learn the cluster-pair representations, which will be used to determine if combining the two clusters is desirable.

Cluster-CR consists of four major modules: Mention-Pair Encoder, Cluster-Pair Encoder,

Mention-Ranking Model, and Cluster-Ranking Model. Figure 17 show the architecture of *Cluster-CR*. Since the Mention-Ranking Model works as a pre-training and search-space-pruning step for the Cluster-Ranking Model, we will not discuss it further, for ease of exposition.

The Mention-Pair Encoder takes as input a mention m, a candidate antecedent coreference a, and context features, to output an embedding vector $\vec{r}_m(a, m)$ using a feed-forward neural network. It consists of three hidden layers with ReLU activation where each layer has the form

$$\vec{h}_i(a, m) = \max\{0, W_i \vec{h}_{i-1}(a, m) + \vec{b}_i\}. \tag{16}$$

The input $\vec{h}_0$ consists of various features: The embeddings of the candidate antecedent and mentions, syntactic features such as the type of the mention words (e.g., noun, pronoun, etc.), context features such as the document genre and the distance between the two mentions. The embedding $\vec{r}_m(a, m)$ is the output $\vec{h}_3$ of the last layer.

The Cluster-Pair Encoder takes as input a pair of mention clusters $c_i = \{m_1^i, m_2^i, \ldots, m_{|c_i|}^i\}$ and $c_j = \{m_1^j, m_2^j, \ldots, m_{|c_j|}^j\}$, and outputs a vector $\vec{r}_c(c_i, c_j)$. It proceeds in three steps:

1) For each pair $(m_i, m_j) \in c_i \times c_j$, vector $\vec{r}_m(m_i, m_j)$ is generated by MPEnc;

2) A matrix $R_m(c_i, c_j)$ is created by using each $\vec{r}_m(m_i, m_j)$ as a column

$$R_m(c_i, c_j) = [\vec{r}_m(m_1^i, m_1^j), \vec{r}_m(m_1^i, m_2^j), \ldots, \vec{r}_m(m_{|c_i|}^i, m_{|c_j|}^j)];$$

3) We apply a pooling operation over each row to get $\vec{r}_c(c_i, c_j)$. Suppose each vector $\vec{r}_m(m_i, m_j) \in \mathbb{R}^d$, then the output vector $\vec{r}_c(c_i, c_j) \in \mathbb{R}^{2d}$ is obtained by concatenating the results of max-pooling and average-pooling on $R_m(c_i, c_j)$. The k-th dimension in $\vec{r}_c(c_i, c_j)$ is

$$\vec{r}_c(c_i, c_j)_k = \begin{cases} \max\{R_m(c_i, c_j)_{k,\cdot}\}, & 0 \le k < d, \\ \text{avg}\{R_m(c_i, c_j)_{k-d,\cdot}\}, & d \le k < 2d. \end{cases} \tag{17}$$

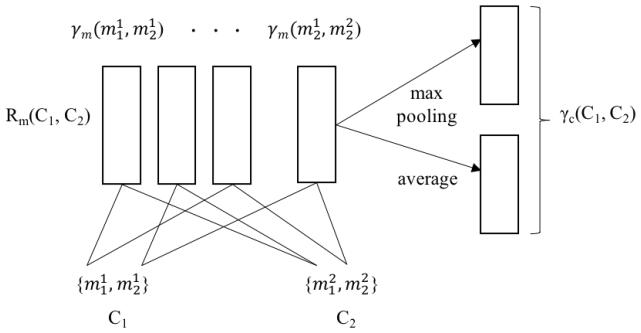

Figure 18: An example of the Cluster-Pair Encoder's workflow

Figure 18 shows an example of the encoder's workflow given two clusters where each cluster has two mentions.

The Cluster-Ranking Model takes as input the representation vector $\vec{r}_c(c_i, c_j)$ of a pair of mention clusters c_i, c_j. It outputs a scalar score $s_c(c_i, c_j)$ that captures the confidence score on whether the two clusters are coreferent or not. This is done by a fully connected neural network

$$s_c(c_i, c_j) = W_c \vec{r}_c(c_i, c_j) + b_c. \tag{18}$$

Note that it is possible that a mention m has no antecedent coreference. Therefore a separate network is also defined to determine if m should link to an "NA" (meaning no antecedent)

$$s_{\text{NA}}(m) = W_{\text{NA}}\vec{r}_m(\text{NA}, m) + b_{\text{NA}}. \tag{19}$$

So far we have discussed the *Cluster-CR* model structure. We now discuss the cluster ranking policy. Cluster ranking can be viewed as a sequential decision procedure. Given a set of mentions in a document with each as a single cluster, we loop through all mentions. In each iteration step, we take one of the following two actions: Either merging the current mention's cluster with an antecedent, or skipping it.

In particular, we define a state $s = (C, m)$ where C contains all clusters (after applying some merging) seen so far, and m is the current mention. In the start state s_0, each cluster in C is just a single mention. In the state $s_i = (C_i, m_i)$, let C_m be the cluster containing m_i, and $A(m_i)$ be candidate antecedent set for m_i. We perform one of the following actions: MERGE(c_m, c) where c is a cluster containing a mention in $A(m_i)$, or PASS(m) leaving c_m unchanged. The probability we should merge or pass is determined by the scores we get from the Cluster-Ranking Model

$$\begin{aligned} P(\text{MERGE}(c_m, c) \mid s_i) &\propto e^{s_c(c_m, c)}, \\ P(\text{PASS}(m) \mid s_i) &\propto e^{s_{\text{NA}}(m)}. \end{aligned} \tag{20}$$

We take the action with the highest probability, and move to the next state $s_{i+1} = (C_{i+1}, m_{i+1})$. The training of the model is based on the learning-to-search algorithm [117].

Additional work. The work [115] uses a mention-ranking model for anaphoricity detection and antecedent ranking. The model is built on neural networks to learn representations that do not need to specify interactions of input features. A follow-up work [114] incorporates global representations of entity clusters using RNN, which is similar to Cluster-CR.

All aforementioned methods assume a set of mentions have been identified in advance. [113] proposes an end-to-end coreference resolution system, which consists of two major steps. First, the model computes embedding representations of text spans (up to a maximum width) that are potential mentions using LSTMs and attention, and generates a potential score for each text span. Only spans with high scores will be considered as mention candidates. Then in the second step, antecedent scores of a pair of spans are calculated by feed-forward networks, and the final coreference score is the sum of the antecedent score and the mention scores of two spans. A follow-up work [116] improves [113] with bi-affine attention, joint mention detection, and mention clustering. Another type of work focuses on cross-lingual coreference resolution [112].

4. DISCUSSION AND FUTURE DIRECTIONS

We discuss several open challenges and opportunities in the intersection of semantic matching and machine learning. The open problems we cover aim to attract the attention of the database community to how machine learning can help semantic matching be more effective. By no means is this an exhaustive list.

Multi-modal data integration. Traditionally semantic matching has focused on textual data. However, there is an abundance of image, sensory, and audio data that is rarely integrated with textual data into a common queryable knowledge repository. This is to a certain extent due to the inherently different methods required to process each aforementioned data

mode. Nonetheless, state-of-the-art deep learning methods can potentially provide the necessary tools and formalisms required for multi-modal data integration. Recent results in multi-modal information extraction [118] and multi-modal deep learning [119] certainly provide positive evidence.

Fast and cheap training data for data integration. Machine learning models for semantic matching can require large amounts of training data when applied over domains with reach domain-specific semantics [97]. Obtaining large number of examples can be resource-intensive in many practical scenarios. Recent approaches in the literature have focused on active learning methods to solicit human supervision more effectively [104]. A pro-mising direction is to understand how these methods relate to weak supervision methods recently introduced in the machine learning community [120]. Overall, there is an immediate need for new algorithmic frameworks and systems for collecting large amounts of training data for DI more effectively.

What makes this challenge unique to semantic matching is that existing weak supervision approaches have focused primarily on textual data for tasks such as that of information extraction [121], or visual data for tasks such as image classification [122]. As such, a fundamental challenge for deep learning-based semantic matching solutions is to devise new weak supervision methods tailored to structured data as well as methods that are more robust to the class imbalance (between positive and negative examples) in semantic matching.

	Product Description Text Span
R1	Kingston 133x high-speed 4GB compact flash card **ts4gcf133**, 21.5 MB per sec data transfer rate, dual-channel support, multi-platform compatibility.

	Product Description Text Span
R2	Kingston **ts4gcf133** 4GB compactflash memory card (133x).

Figure 19: An example of matching product descriptions

Semantic-aware word embeddings. Deep learning models have limited capability of capturing domain-specific semantics. A promising research direction is to explore mechanisms for introducing domain-specific knowledge to deep learning models. We envision this to be possible either via new weak-supervision methods [118] or by integrating domain-knowledge in the architecture of deep learning models [123]. We envision the design of new domain-specific representation learning models, such as domain-specific word embeddings for semantic matching. For example, consider matching the two records shown in Figure 19. The product serial code "ts4gcf133" shown in the figure a description of product features included in other parts of the description, i.e., its capacity of "4GB" and speed of "133x". As a result the embedding for this token should capture this semantic information with respect to the structure of the product serial key.

Deep learning for schema alignment. Schema alignment matches types and attributes. Although automatic schema mapping seems an overkill when we align data between two data sources with typical sizes of schemas, it is important when we consider millions of sources from the web. For example, Pimplikar et al. [124] study how to apply graphical model to align webtables with knowledge bases, by aligning entities and schemas at the same time.

Universal schema [125, 126] has revolutionized schema alignment. It is motivated by Ope-

nIE knowledge extraction: unlike traditional information extraction that extracts knowledge according to a predefined ontology (i.e., schema), OpenIE extracts (subject, predicate, object) triples, where the predicate can be any word or phrase from texts. Reasoning over the predicates and mapping them to existing ontology predicates is important to broaden applications for OpenIE results. Overall, deep learning solutions to schema alignment have focused mainly on knowledge base construction and little work has been done for structured data sources. Exploring the use of deep learning for schema matching and alignment further is a promising direction.

Human-in-the-loop semantic matching. Machine learning models, or any other automatic approaches, can hardly obtain a 100% accuracy on semantic matching, which is a very complex task. It is thus important to involve human in the loop, conducting labeling, verifications, and auditing. A future direction is for a system to automatically identify when, where, and how to get human involved, by applying active learning, transfer learning, and reinforcement learning.

Efficient model serving for semantic matching. Model serving for semantic matching entails several resource intensive operations such as data normalization and blocking before entity resolution or data fusion is performed. Existing methods execute each step in isolation without taking into account the computation performed in subsequent steps along the semantic matching pipeline. Open questions here include abstractions that will enable RDBMS-style plan generation and optimization to serve semantic matching models efficiently by avoiding redundant computation or by reusing computation across different steps.

Micro-services for semantic matching. Historically, semantic matching solutions have focused on isolated problems, such as entity resolution or schema mapping, rather than end-to-end solutions. However, the recent results in semantic matching suggest that machine learning can provide a common formal footing for all different problems along the data integration stack. A systematic study is required to identify the common abstractions across different ML-based solutions in semantic matching (e.g., data labeling, data partitioning, training, testing, etc.). These abstractions can in turn lead to a framework of elastic micro-services for semantic matching [127].

5. CONCLUSIONS

Semantic matching encompasses a broad range of problems, including entity matching, entity linking, textual entailment, question answering, schema/ontology matching, and more. Over the past few years, numerous works have applied deep learning to semantic matching. Yet, this is just the beginning. In the foreseeable future it is likely that there will be even more interest in this topic.

To address this growing interest, in this paper we have provided a comprehensive survey of deep learning works for semantic matching. We have discussed semantic matching problems, DL building blocks, and how these blocks have been used to develop DL solutions. It is our hope that this survey will be beneficial to a broad range of readers, including seasoned researchers, new researchers entering this area, graduate students, and practitioners seeking to implement DL solutions for semantic matching.

REFERENCES

[1] I. Goodfellow *et al.*, *Deep Learning.* MIT Press, 2016.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[4] W. Wang *et al.*, "Database meets deep learning: Challenges and opportunities," *ACM SIGMOD Record*, vol. 45, no. 2, pp. 17–22, 2016.

[5] J. Dittrich, "Deep learning (m)eats databases." VLDB Keynote, 2017.

[6] A. Gattani, D. S. Lamba, N. Garera, M. Tiwari, X. Chai, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan, "Entity extraction, linking, classification, and tagging for social media: a wikipedia-based approach," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1126–1137, 2013.

[7] P. Christen, *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, ser. Data-Centric Systems and Applications. Springer, 2012.

[8] A. Doan, A. Y. Halevy, and Z. G. Ives, *Principles of Data Integration.* Morgan Kaufmann, 2012.

[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 1, pp. 1–16, 2007.

[10] T. P. George Papadakis, "Web-scale, schema-agnostic, end-to-end entity resolution (tutorial)," in *Proceedings of The International Conference on World Wide Web*, 2018.

[11] L. Getoor and A. Machanavajjhala, "Entity resolution: theory, practice & open challenges," *Proceedings of The VLDB Endowment*, vol. 5, no. 12, pp. 2018–2019, 2012.

[12] H. Köpcke and E. Rahm, "Frameworks for entity matching: A comparison," *Data & Knowledge Engineering*, vol. 69, no. 2, pp. 197–210, 2010.

[13] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas, "Comparative analysis of approximate blocking techniques for entity resolution," *PVLDB*, vol. 9, no. 9, pp. 684–695, 2016.

[14] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2nd Edition).* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.

[15] W. Shen, J. Wang, and J. Han, "Entity linking with a knowledge base: Issues, techniques, and solutions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 2, pp. 443–460, 2015.

[16] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of The 16th International Conference on World Wide Web.* ACM, 2007, pp. 697–706.

[17] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The Semantic Web.* Springer, 2007, pp. 722–735.

[18] R. Sharnagat, "Named entity recognition: A literature survey," *Center For Indian Language Technology*, 2014.

[19] T. Lin, O. Etzioni *et al.*, "Entity linking at web scale," in *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*. Association for Computational Linguistics, 2012, pp. 84–88.

[20] N. Nakashole, G. Weikum, and F. Suchanek, "Patty: a taxonomy of relational patterns with semantic types," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 2012, pp. 1135–1145.

[21] T. Cheng, X. Yan, and K. C.-C. Chang, "Entityrank: searching entities directly and holistically," in *Proceedings of The 33rd International Conference on Very Large Data Bases*. VLDB Endowment, 2007, pp. 387–398.

[22] G. Demartini, T. Iofciu, and A. P. De Vries, "Overview of the inex 2009 entity ranking track," in *International Workshop of the Initiative for the Evaluation of XML Retrieval*. Springer, 2009, pp. 254–264.

[23] M. Michelson and S. A. Macskassy, "Discovering users' topics of interest on twitter: a first look," in *Proceedings of The Fourth Workshop on Analytics for Noisy Unstructured Text Data*. ACM, 2010, pp. 73–80.

[24] O. Phelan, K. McCarthy, and B. Smyth, "Using twitter to recommend real-time topical news," in *Proceedings of The Third ACM Conference on Recommender Systems*. ACM, 2009, pp. 385–388.

[25] R. Sukthanker, S. Poria, E. Cambria, and R. Thirunavukarasu, "Anaphora and coreference resolution: A review," *arXiv preprint arXiv:1805.11824*, 2018.

[26] P. Elango, "Coreference resolution: A survey," *University of Wisconsin, Madison, WI*, 2005.

[27] V. Ng, "Coreference resolution: Successes and challenges," http://www.hlt.utdallas.edu/%7Evince/ijcai-2016/coreference/slides.pdf, 2016.

[28] T. S. Morton, "Using coreference for question answering," in *Proceedings of the Workshop on Coreference and its Applications*. Association for Computational Linguistics, 1999, pp. 85–89.

[29] R. Stuckardt, "Coreference-based summarization and question answering: a case for high precision anaphor resolution," in *International Symposium on Reference Resolution*, 2003.

[30] J. L. Vicedo and A. Ferrández, "Importance of pronominal anaphora resolution in question answering systems," in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2000, pp. 555–562.

[31] M. Novák, "Utilization of anaphora in machine translation," *Proceedings of Contributed Papers, Week of Doctoral Students*, pp. 155–160, 2011.

[32] L. M. Werlen and A. Popescu-Belis, "Using coreference links to improve spanish-to-english machine translation," in *Proceedings of the 2nd Workshop on Coreference Resolution Beyond OntoNotes (CORBON 2017)*, 2017, pp. 30–40.

[33] S. Bergler, R. Witte, M. Khalife, Z. Li, and F. Rudzicz, "Using knowledge-poor coreference resolution for text summarization," in *Proceedings of DUC*, vol. 3, 2003.

[34] J. Steinberger, M. Poesio, M. A. Kabadjov, and K. Ježek, "Two uses of anaphora resolution in summarization," *Information Processing & Management*, vol. 43, no. 6, pp. 1663–1680, 2007.

[35] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 695–701, 2011.

[36] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Ontology matching: A machine learning approach," in *Handbook on Ontologies*. Springer, 2004, pp. 385–403.

[37] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[38] N. F. Noy, "Semantic integration: a survey of ontology-based approaches," *ACM Sigmod Record*, vol. 33, no. 4, pp. 65–70, 2004.

[39] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," in *Journal on Data Semantics IV*. Springer, 2005, pp. 146–171.

[40] E. Agirre, M. Diab, D. Cer, and A. Gonzalez-Agirre, "Semeval-2012 task 6: A pilot on semantic textual similarity," in *Proceedings of the First Joint Conference on Lexical and Computational Semantics*. Association for Computational Linguistics, 2012, pp. 385–393.

[41] W. H. Gomaa and A. A. Fahmy, "A survey of text similarity approaches," *International Journal of Computer Applications*, vol. 68, no. 13, pp. 13–18, 2013.

[42] D. Jurgens and M. T. Pilehvar, "Semantic similarity frontiers: From concepts to documents," in *Conference on Empirical Methods in Natural Language Processing EMNLP*, 2015, p. 269.

[43] I. Androutsopoulos and P. Malakasiotis, "A survey of paraphrasing and textual entailment methods," *Journal of Artificial Intelligence Research*, vol. 38, pp. 135–187, 2010.

[44] M. Sammons, V. Vydiswaran, and D. Roth, "Recognizing textual entailment," *Multilingual Natural Language Applications: From Theory to Practice*, pp. 209–258, 2012.

[45] I. Dagan, D. Roth, and F. M. Zanzotto, "Textual entailment," http://u.cs.biu.ac.il/%7Edagan/TE-Tutorial-ACL07.ppt, 2017.

[46] S. Padó, M. Galley, D. Jurafsky, and C. Manning, "Robust machine translation evaluation with entailment features," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*. Association for Computational Linguistics, 2009, pp. 297–305.

[47] S. Harabagiu and A. Hickl, "Methods for using textual entailment in open-domain question answering," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2006, pp. 905–912.

[48] M. Wang and C. D. Manning, "Probabilistic tree-edit models with structured latent variables for textual entailment and question answering," in *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 1164–1172.

[49] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 670–680.

[50] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[51] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 2383–2392.

[52] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou, "Gated self-matching networks for reading comprehension and question answering," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2017, pp. 189–198.

[53] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, "Qanet: Combining local convolution with global self-attention for reading comprehension," *arXiv preprint arXiv:1804.09541*, 2018.

[54] W. Cui, Y. Xiao, H. Wang, Y. Song, S.-w. Hwang, and W. Wang, "Kbqa: learning question answering over qa corpora and knowledge bases," *Proceedings of the VLDB Endowment*, vol. 10, no. 5, pp. 565–576, 2017.

[55] R. Das, M. Zaheer, S. Reddy, and A. McCallum, "Question answering on knowledge bases and text using universal schema and memory networks," *arXiv preprint arXiv:1704.08384*, 2017.

[56] D. Diefenbach, V. Lopez, K. Singh, and P. Maret, "Core techniques of question answering systems over knowledge bases: a survey," *Knowledge and Information Systems*, vol. 55, no. 3, pp. 529–569, 2018.

[57] K. Liu, J. Zhao, S. He, and Y. Zhang, "Question answering over knowledge bases," *IEEE Intelligent Systems*, vol. 30, no. 5, pp. 26–35, 2015.

[58] E. Choi, H. He, M. Iyyer, M. Yatskar, W.-t. Yih, Y. Choi, P. Liang, and L. Zettlemoyer, "Quac: Question answering in context," *arXiv preprint arXiv:1808.07036*, 2018.

[59] T. Kočiskỳ, J. Schwarz, P. Blunsom, C. Dyer, K. M. Hermann, G. Melis, and E. Grefenstette, "The narrativeqa reading comprehension challenge," *Transactions of the Association of Computational Linguistics*, vol. 6, pp. 317–328, 2018.

[60] Y. Bachrach, A. Zukov-Gregoric, S. Coope, E. Tovell, B. Maksak, J. Rodriguez, and C. McMurtie, "An attention mechanism for answer selection using a combined global and local view," *arXiv preprint arXiv:1707.01378*, 2017.

[61] A. Severyn and A. Moschitti, "Learning to rank short text pairs with convolutional deep neural networks," in *Proceedings of The 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015, pp. 373–382.

[62] G. Shen, Y. Yang, and Z.-H. Deng, "Inter-weighted alignment network for sentence pair modeling," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1179–1189.

[63] M. Tan, C. Dos Santos, B. Xiang, and B. Zhou, "Improved representation learning for question answer matching," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016, pp. 464–473.

[64] S. Wan, Y. Lan, J. Guo, J. Xu, L. Pang, and X. Cheng, "A deep architecture for semantic matching with multiple positional sentence representations." in *AAAI*, vol. 16, 2016, pp. 2835–2841.

[65] Z. Wang, W. Hamza, and R. Florian, "Bilateral multi-perspective matching for natural language sentences," *arXiv preprint arXiv:1702.03814*, 2017.

[66] Y. Yang, W.-t. Yih, and C. Meek, "Wikiqa: A challenge dataset for open-domain question answering," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 2013–2018.

[67] M. Feng, B. Xiang, M. R. Glass, L. Wang, and B. Zhou, "Applying deep learning to answer selection: A study and an open task," *arXiv preprint arXiv:1508.01585*, 2015.

[68] R. Sarikaya, P. A. Crook, A. Marin, M. Jeong, J.-P. Robichaud, A. Celikyilmaz, Y.-B. Kim, A. Rochette, O. Z. Khan, X. Liu *et al.*, "An overview of end-to-end language understanding and dialog management for personal digital assistants," in *Spoken Language Technology Workshop (SLT), 2016 IEEE*. IEEE, 2016, pp. 391–397.

[69] A. Mishra and S. K. Jain, "A survey on question answering systems with classification," *Journal of King Saud University-Computer and Information Sciences*, vol. 28, no. 3, pp. 345–361, 2016.

[70] K. Arivuchel Van and K. Lakahmi, "Reading comprehension system-a review," *Indian J. Sci. Res*, vol. 14, no. 1, pp. 83–90, 2017.

[71] Z. Liu, X. Chen, and M. Sun, "A simple word trigger method for social tag suggestion," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 1577–1588.

[72] Z. Ding, Q. Zhang, and X. Huang, "Automatic hashtag recommendation for microblogs using topic-specific translation model," *Proceedings of COLING 2012: Posters*, pp. 265–274, 2012.

[73] S. Sedhai and A. Sun, "Hashtag recommendation for hyperlinked tweets," in *Proceedings of The 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 2014, pp. 831–834.

[74] M. Gupta, R. Li, Z. Yin, and J. Han, "Survey on social tagging techniques," *ACM Sigkdd Explorations Newsletter*, vol. 12, no. 1, pp. 58–72, 2010.

[75] C. W. Leong, R. Mihalcea, and S. Hassan, "Text mining for automatic image tagging," in *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. Association for Computational Linguistics, 2010, pp. 647–655.

[76] M. Bernstein, A. Doan, and C. N. Dewey, "MetaSRA: normalized human sample-specific metadata for the sequence read archive," *Bioinformatics*, vol. 33, no. 18, pp. 2914–2923, 2017.

[77] Y. Cao, L. Hou, J. Li, and Z. Liu, "Neural collective entity linking," in *Proceedings of the 27th International Conference on Computational Linguistics*, 2018, pp. 675–686.

[78] M. Francis-Landau, G. Durrett, and D. Klein, "Capturing semantic similarity for entity linking with convolutional neural networks," in *Proceedings of NAACL-HLT*, 2016, pp. 1256–1261.

[79] O.-E. Ganea and T. Hofmann, "Deep joint entity disambiguation with local neural attention," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2619–2629.

[80] N. Gupta, S. Singh, and D. Roth, "Entity linking via joint encoding of types, descriptions, and context," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2681–2690.

[81] N. Kolitsas, O.-E. Ganea, and T. Hofmann, "End-to-end neural entity linking," in *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2018, pp. 519–529. [Online]. Available: http://aclweb.org/anthology/K18-1050

[82] T. H. Nguyen, N. Fauceglia, M. R. Muro, O. Hassanzadeh, A. M. Gliozzo, and M. Sadoghi, "Joint learning of local and global features for entity linking via neural networks," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 2310–2320.

[83] J. Raiman and O. Raiman, "Deeptype: Multilingual entity linking by neural type system evolution," *arXiv preprint arXiv:1802.01021*, 2018.

[84] A. Sil, G. Kundu, R. Florian, and W. Hamza, "Neural cross-lingual entity linking," *arXiv preprint arXiv:1712.01813*, 2017.

[85] Y. Sun, L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang, "Modeling mention, context and entity with neural networks for entity disambiguation," in *IJCAI*, 2015, pp. 1333–1339.

[86] G. Durrett and D. Klein, "A joint model for entity analysis: Coreference, typing, and linking," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 477–490, 2014.

[87] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with neural tensor networks for knowledge base completion," in *Advances in Neural Information Processing Systems*, 2013, pp. 926–934.

[88] S. R. Bowman, J. Gauthier, A. Rastogi, R. Gupta, C. D. Manning, and C. Potts, "A fast unified model for parsing and sentence understanding," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016, pp. 1466–1477.

[89] Q. Chen, Z.-H. Ling, and X. Zhu, "Enhancing sentence embedding with generalized pooling," in *Proceedings of the 27th International Conference on Computational Linguistics*, 2018, pp. 1815–1826.

[90] J. Choi, K. M. Yoo, and S.-g. Lee, "Learning to compose task-specific tree structures," in *Proceedings of the 2018 Association for the Advancement of Artificial Intelligence (AAAI). and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, 2018.

[91] J. Im and S. Cho, "Distance-based self-attention network for natural language inference," *arXiv preprint arXiv:1712.02047*, 2017.

[92] Y. Liu, C. Sun, L. Lin, and X. Wang, "Learning natural language inference using bidirectional lstm model and inner-attention," *arXiv preprint arXiv:1605.09090*, 2016.

[93] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," *arXiv preprint arXiv:1606.01933*, 2016.

[94] T. Shen, T. Zhou, G. Long, J. Jiang, S. Pan, and C. Zhang, "Disan: Directional self-attention network for rnn/cnn-free language understanding," *arXiv preprint arXiv:1709.04696*, 2017.

[95] A. Talman, A. Yli-Jyrä, and J. Tiedemann, "Natural language inference with hierarchical bilstm max pooling architecture," *arXiv preprint arXiv:1808.08762*, 2018.

[96] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML Deep Learning Workshop*, vol. 2, 2015.

[97] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, "Deep learning for entity matching: A design space exploration," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 19–34.

[98] H. He, K. Gimpel, and J. Lin, "Multi-perspective sentence similarity modeling with convolutional neural networks," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1576–1586.

[99] H. He and J. Lin, "Pairwise word interaction modeling with deep neural networks for semantic similarity measurement," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 937–948.

[100] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity." in *AAAI*, vol. 16, 2016, pp. 2786–2792.

[101] P. Neculoiu, M. Versteegh, and M. Rotaru, "Learning text similarity with siamese recurrent networks," in *Proceedings of the 1st Workshop on Representation Learning for NLP*, 2016, pp. 148–157.

[102] M. Nicosia and A. Moschitti, "Accurate sentence matching with hybrid siamese networks," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 2017, pp. 2235–2238.

[103] T. M. Lai, T. Bui, and S. Li, "A review on deep learning techniques applied to answer selection," in *Proceedings of the 27th International Conference on Computational Linguistics*, 2018, pp. 2132–2144.

[104] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu, "Corleone: hands-off crowdsourcing for entity matching," in *Proceedings of The 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 2014, pp. 601–612.

[105] P. Konda, S. Das, P. Suganthan GC, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton *et al.*, "Magellan: Toward building entity matching management systems," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 1197–1208, 2016.

[106] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1454–1467, 2018.

[107] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *arXiv preprint arXiv:1408.2927*, 2014.

[108] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[109] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[110] K. Clark and C. D. Manning, "Deep reinforcement learning for mention-ranking coreference models," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 2256–2262.

[111] ——, "Improving coreference resolution by learning entity-level distributed representations," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016, pp. 643–653.

[112] G. Kundu, A. Sil, R. Florian, and W. Hamza, "Neural cross-lingual coreference resolution and its application to entity linking," *arXiv preprint arXiv:1806.10201*, 2018.

[113] K. Lee, L. He, M. Lewis, and L. Zettlemoyer, "End-to-end neural coreference resolution," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 188–197.

[114] S. Wiseman, A. M. Rush, and S. M. Shieber, "Learning global features for coreference resolution," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 994–1004.

[115] S. J. Wiseman, A. M. Rush, S. M. Shieber, and J. Weston, "Learning anaphoricity and antecedent ranking features for coreference resolution." Association for Computational Linguistics, 2015.

[116] R. Zhang, C. N. d. Santos, M. Yasunaga, B. Xiang, and D. Radev, "Neural coreference resolution with deep biaffine attention by joint mention detection and mention clustering," *arXiv preprint arXiv:1805.04893*, 2018.

[117] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine Learning*, vol. 75, no. 3, pp. 297–325, 2009.

[118] S. Wu, L. Hsiao, X. Cheng, B. Hancock, T. Rekatsinas, P. Levis, and C. Ré, "Fonduer: Knowledge base construction from richly formatted data," in *SIGMOD*, ser. SIGMOD '18, 2018.

[119] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. USA: Omnipress, 2011, pp. 689–696. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104482.3104569

[120] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré, "Data programming: Creating large training sets, quickly," in *Advances in Neural Information Processing Systems*, 2016, pp. 3567–3575.

[121] A. Ratner, S. H. Bach, H. Ehrenberg *et al.*, "Snorkel: Rapid training data creation with weak supervision." VLDB, 2017.

[122] V. Mnih and G. E. Hinton, "Learning to label aerial images from noisy data." ICML, 2012.

[123] W. W. Cohen, "Tensorlog: A differentiable deductive database," *CoRR*, vol. abs/1605.06523, 2016.

[124] R. Pimplikar and S. Sarawagi, "Answering table queries on the web using column keywords," *PVLDB*, vol. 5, no. 10, pp. 908–919, 2012.

[125] S. Riedel, L. Yao, B. M. Marlin, and A. McCallum, "Relation extraction with matrix factorization and universal schemas," in *HLT-NAACL*, 2013.

[126] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon, "Representing text for joint embedding of text and knowledge bases," in *EMNLP*, 2015.

[127] Y. Govind, E. Paulson, M. Ashok, P. S. GC, A. Hitawala, A. Doan, Y. Park, P. L. Peissig, E. LaRose, and J. C. Badger, "Cloudmatcher: A cloud/crowd service for entity matching."

Received on July 20, 2021
Accepted on September 03, 2021