UNIVERSITÀ
DI TRENTO

Department of Computer Science and Information Engineering

International ICT Doctoral School
PhD Dissertation

Leveraging Structure for Effective Question Answering

Daniele BONADIMAN

Advisor

Alessandro Moschitti
University of Trento

April 2020

Abstract

In this thesis, we focus on Answer Sentence Selection (A2S) that is the core task of retrieval based question answering. A2S consists of selecting the sentences that answer user queries from a collection of documents retrieved by a search engine. Over more than two decades, several solutions based on machine learning have been proposed to solve this task, starting from simple approaches based on manual feature engineering to more complex Structural Tree Kernels models, and recently Neural Network architectures. In particular, the latter requires little human effort as they can automatically extract relevant features from plain text. The development of neural architectures brought improvements in many areas of A2S, reaching unprecedented results. They substantially increase accuracy on almost all benchmark datasets for A2S. However, this has come with the cost of a huge increase in the number of parameters and computational costs of the models. A large number of parameters has led to two drawbacks. The model requires a massive amount of data to train effectively, and huge computational power to maintain an acceptable transaction per second in a production environment. Current state-of-the-art techniques for A2S use huge Transformer architectures, having up to 340 million parameters, pre-trained on a massive amount of data, e.g., BERT. The latter and related models in the same family, such as RoBERTa, are general architectures, i.e., they can be applied to many tasks of NLP without any architectural change.

In contrast to the trend above, we focus on specialized architectures for A2S that can effectively encode the local structure of the question and answer candidate and global information, i.e., the structure of the task and the context in which the answer candidate appears.

In particular, we propose solutions to effectively encode both the local and the global structure of A2S in efficient neural network models. (i) We encode syntactic information in a fast CNN architecture exploiting the capabilities of Structural Tree Kernel to encode the syntactic structure. (ii) We propose an efficient model that can use semantic relational information between question and answer candidates by pretraining word representations on a relational knowledge base. (iii) This efficient approach is further extended to encode each answer candidate's contextual information, encoding all answer candidates in the original context. Lastly, (iv) we propose a solution to encode task-specific structure that is available, for example, available on the community Question Answering task.

The final model, which encodes different aspects of the task, achieves state-of-the-art performance on A2S compared with other efficient architectures. The proposed model is more efficient than attention based architectures and outperforms BERT by two orders of magnitude in terms of transaction per second during training and testing, i.e., it processes 700 questions per second compared to 6 questions per second for BERT when training on a single GPU.

Contents

Chapter 1

Introduction

Ever since the advent of the Internet, finding ways to store and access information has always been a problem. In the last two decades, search engines, like Google, Bing, or Yahoo, played a crucial role in helping people seek information among the considerable number of documents on the Internet. A search engine accepts a query, i.e., a set of keywords, and returns as output a list of related documents. Although search engines are the de facto standard for accessing the information on the web, users still need to put much effort to digest such information. Firstly, users need to adapt their style of asking questions from long descriptive questions (easy to understand for other people) to short keyword-based questions. The second source of effort for the users comes from the fact that search engines output a list of documents as a result, and therefore users still need to search through these documents for the required information. Search engines improved in understanding questions; hence, allowing users to search for information using natural language rather than keyword-based queries. Although this development eased the user experience for querying the search engine, the second issue remains open. It is more critical now than ever, as there has been a shift in how people search for information. With the advent of smartphones and personal assistants like Siri, Google Assistant, or Alexa, users need short, direct, and precise answers to their questions without processing documents themselves.

The task of automatically providing answers to questions is known as automatic Question Answering (QA). QA is a broad and multi-faced problem, as both questions and answers may be complex to understand, even for humans. For example, composite questions such as *"Where was the president of the United States born?"* require answering two different questions *"Who is the president of the United States?"* and *"Where was X born?"*. Additionally, another type of complexity in questions may come from the fact that the answer requires expressing opinions or performing different levels of (causal) inference, e.g., *"What is the cause of global warming?"*. This complexity created, over the years, several different solutions for the task of Question Answering. Nested factual questions are better answered by a system that stores knowledge in *structured* databases such as tables or relational knowledge bases. A retrieval-based approach on *unstructured* text is more suited for complex non-factual questions. In retrieval approaches, a search engine retrieves a document containing information about global warming. A second component selects and then extracts the relevant snippet

of text that answers the question. Even though QA on structured knowledge is more precise and controllable than the one on unstructured text, it is expensive to maintain as relational knowledge bases are manually maintained. QA on structured knowledge-bases can answer a limited number of question types, i.e., questions asking facts (factual questions). In contrast, retrieval-based question answering systems answer a broader set of questions such as causal questions (why/how) or descriptive questions such as *What is the DNA?*. For these reasons, in this thesis, we focus on the information retrieval-based Question Answering.

The standard approach to question answering uses a Search Engine (SE) to retrieve relevant documents and an Answer Sentence Selector (A2S) to select the sentence that answers the question. This simple pipeline can be enriched with an optional component that extracts, generates, or reformulates the final answer. This last component is optional and application-specific since the answer can be presented to the user in different forms depending on the application. For example, a virtual assistant wants a short and conversational answer. However, on a smartphone, we may want to highlight the answer sentence in the source document's original context.

A2S is the task of selecting the sentence or a small passage that contains the answer to the question from a list of candidate sentences. This list of sentences can come from different sources, depending on the implementation of the pipeline. For example, it can be the list of sentences, i.e., the answer candidates, in the first document of the search engine [149, 64], a list of passages (if the search engine retrieves sub-parts of the documents) [144] or the document summaries [5]. A2S uses advanced machine learning models to assign a score to each answer candidate according to their relevance to the question [107, 109, 145, 36]. These models are supervised and learn from training examples, i.e., they are complex mathematical functions whose parameters adjusted to approximate a given question answering dataset. Most models of A2S take the question and a single answer candidate as input and produce a score that represents the relevance of the answer candidate to the question. An A2S component sorts (re-ranks) the original list of answer candidates according to the machine learning models' score. Over the years, several machine learning approaches have been proposed for the task. Among others, notable examples are: Severyn and Moschitti 2013 [107] that proposed to use SVM with Tree Kernels over syntactic structures achieving unprecedented results on the task without much need to exploit task-specific features of the datasets; Yu et al. [161] and Severyn and Moschitti 2015 [109] was the first technique that successfully applied deep learning to A2S. From there, Deep Learning is the de facto standard for the task. It received further developments, particularly with the inclusion of attention mechanisms [145] to better capture the semantic relatedness between the words in the question and the answer. Additionally, recent approaches for training neural language models, e.g., ELMo [94], GPT [97], BERT [29], RoBERTa [74], XLNet [151], have led to major improvements in several NLP areas. These methods create a sentence representation that captures the relations between words and their compounds by pre-training big neural networks model on huge datasets. Interestingly, these pre-trained models can be easily applied to different tasks by fine-tuning them on the target training data.

All the models above (especially the transformer-based architectures) require many layers

and a considerable number of parameters (up to 340 million for BERT Large) to capture all the nuances of language needed to solve different tasks. This generality is achieved by sacrificing efficiency, thus posing critical challenges for having such models in production: models have to be scaled horizontally on multiple GPUs to obtain acceptable latencies, even at test time, increasing the operational cost in real-world scenarios. This also has a huge environmental issue, as pointed out by Strubell, Ganesh, and McCallum [120], and they require many resources for pre-training, e.g., both data and compute power (TPUs). These resources may not be available for low resource languages or domain-specific applications.

In this thesis, we investigate and propose solutions to design accurate A2S models, still preserving high efficiency.

1.1 Question Answering

A Question Answering system is a computer system capable of answering user questions using natural language. Question Answering is a challenging research problem since it requires a deep understanding of the human language to understand the user question and produce a meaningful answer. The way information is stored additional challenges to the task. For example, information can be stored in a *structured* way, using knowledge bases, or in *unstructured* textual documents. Structured Knowledge bases are precise sources of information, and they are an effective way to store domain-specific knowledge, such as the organizational chart of a company. However, they are expensive and hard to maintain in open-domain scenarios when users can ask questions about any topic. Inside a knowledge-base, all human knowledge needs to be mapped in a logical form. Unstructured textual documents, on the other end, are ubiquitous. Question Answering systems capable of finding and extracting information from open texts require little maintenance and can be applied across domains.

1.1.1 Structured data

In structured databases, information consists of data-points and their relations. One of the oldest and most common types of data structures used to store and access information is tabular. For example, a simple database for a bank may contain a table of the clients, another table for the accounts, and a third table that contains the transactions among different accounts. In this setting, when users search for information, the user queries the database using an ad-hoc language, e.g., SQL, and retrieves a new tabular representation of the searched data. Tabular databases are the building block of many information systems. However, they are not suited for storing the world's general knowledge for building an open domain question answering system.

Another standard method to store knowledge is using relational knowledge graphs. Knowledge graphs represent data, e.g., facts, as entities together with their relations. For example, it is possible to represent the fact that "Metallica" played the "Black Album" in the form of a triplet <metallica, *played*, black_album>. In the same way, <black_album, contains, enter_sandman> represents the fact that the "Black Album" contains the song "Enter Sandman."

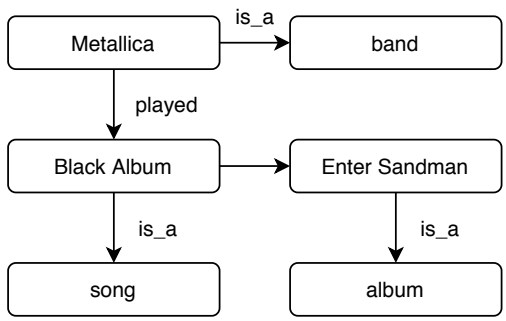

Figure 1.1: Example of a Knowledge base

The collection of all facts contained in a knowledge base forms a graph with entities as nodes and relations as edges, as depicted in Figure 1.1. Similarly to tabular databases, it is possible to retrieve information from the knowledge graph by querying it. For example, it is possible to retrieve the information of who played "Enter Sandman" with the query <?, *played*, enter_sandman>. In this example, the engine queries the knowledge base, performs the required inference steps, and retrieves the entity "Metallica."

Over the years, different initiatives attempted to store general information of the world in knowledge bases. Notable examples are:

- YAGO [121] and DBpedia [3] are open-source knowledge graphs automatically extracted from Wikipedia.
- Conceptnet [72] that is a project that aims at encoding common knowledge as a graph. For example, it is common human knowledge that a rock that is thrown eventually falls.
- Freebase [12] was a collaborative community effort to create and maintain a general Knowledge Graph about the world.

Moreover, private companies, such as Google, have their knowledge graphs to perform Question Answering and enrich their users' search experience. Knowledge graphs are powerful but are typically manually built and maintained.

1.1.1.1 Knowledge Base Question Answering

Question answering on Knowledge bases (KBQA) is a challenging problem since it requires to perfectly understand the user question and convert it to a query for the knowledge base. For simple cases, this is achieved using template-based matching [134]. Most of the questions have a standard form that can be easy to detect, e.g., *" Who played Enter Sandman? "* can use the template *" Who played X? "* to map the question to the query "<?, *played*, X>". The same template is valid for the question *" Who played Yellow Submarine? "*. Template-matching is effective for simple questions; however, it is not scalable for harder questions such as composite questions, e.g., *"Where was the president of the United States born?"*. Com-

plex questions require automatic machine learning approaches such as semantic parsing [9]. A semantic parser learns from example to map questions to their Abstract Meaning Representation (AMR), i.e., a structured representation of the sentence's meaning. The AMR for the question can be used to query the database. When queried, the knowledge base returns a fact answering the question, i.e., usually a triplet from the graph. Despite KBQA can precisely answer questions, there are many sources of error in the process, e.g., the parser fails to map the question to the KB-query or the fact answering the question is not present in the database. From that, KBQA may fail to achieve high recall, thus failing to identify a correct answer to the user question. KBQA requires an additional component to convert facts from the structured form, i.e., a triplet, to a natural language answer. This task is known as Natural Language Generation (NLG) and is crucial for any KBQA system [18].

Moreover, KBQA can answer only a limited amount of user questions, i.e., factual questions. Factual questions are the type of questions that asks about world facts and do not include opinions, explanations, or questions asking about procedures, e.g., *"How can I apply for a Visa?"*.

1.1.2 Unstructured data

Most of the knowledge on the Internet is stored as textual documents for humans to read. Documents on the Internet contain different categories of information. For example, encyclopedia articles, such as Wikipedia documents, contain factual information about the world. Online forums can contain opinions about various topics and procedural information, such as the procedure to apply for a visa. This information is not structured; hence it is difficult for a machine to understand and process. Nevertheless, question answering systems capable of leveraging this information can answer a broader set of questions than the KB counterparts, potentially achieving higher recall in a production setting.

1.1.2.1 Information Retrieval

Question Answering on textual documents is an Information Retrieval (IR) task. An information retrieval system is a computer system capable of identifying and retrieving information from various sources. IR systems can operate on different types of information, such as text, audio, images, or maps. However, searches are typically based on textual documents, such as web pages or books. The most iconic example of information retrieval systems is search engines. A search engine accepts user queries and retrieves related documents in the form of rank, i.e., a ranked list of documents. The list of documents from a search is ranked according to each document's relevance to the query. Search engines can perform searches on a different scale, e.g., a library may have a search engine to identify books, in contrast to the Web search engines that operate on a web-scale, i.e., index and perform searches on the entire Internet.

Search engines are an invaluable asset for accessing human knowledge. Therefore, they are a crucial component for open-domain question answering systems.

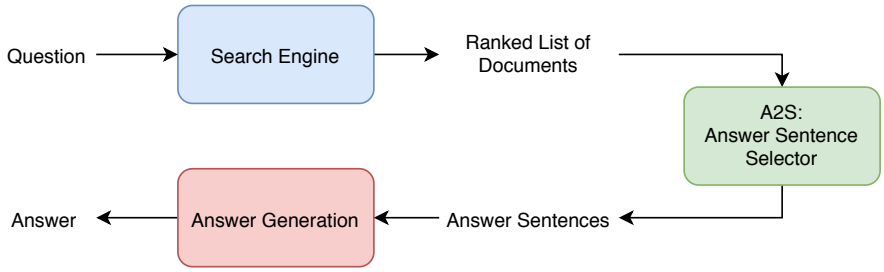

Figure 1.2: General QA architecture

1.1.3 General Architecture for QA

The core components of an information-retrieval based question answering system are the search engine, the answer sentence selector, and the answer generator. This pipeline is depicted in Figure 1.2.0.1. The Search Engine takes the user question as input and retrieves the ranked list of related documents. The Answer Sentence Selector (A2S) selects from the documents the sentence or sentences answering the user question. The final component, the answer generation module, take the answer sentences in output from the A2S and generate a coherent answer for the user.

1.1.3.1 Search Engine

Search Engine (SE) is a general term in the community of Information Retrieval (IR). A SE is typically a computer system that enables search and retrieval of documents, typically at web-scale [104]. A basic SE index a collection of documents, e.g., the entire web, in an index. Typically, the SE preprocesses the documents to extract relevant features from each document for fast retrieval. When the user searches on the collection, the SE computes lexical and semantic similarity features between the user query and each document in the index and returns the most similar documents related to the user query. Search engines that operate at the web-scale include additional features, such as Page Rank [68] or user preference features, such as the history of the user searches. Depending on the application, SE systems for QA do not always operate at web-scale, but they typically index encyclopedias, such as Wikipedia, or online forums.

For the remainder of this work, we refer to SE as a black-box capable of retrieving a ranked list of documents relevant to the user query.

1.1.3.2 A2S: Answer Sentence Selection

The central task of an IR based QA system is the Answer Sentence Selection (A2S). The task consists in identifying the sentence answering the user question inside an ordered list of answer candidates. The model takes in input the sentences, or small passages, extracted from the document(s) retrieved by the SE, and it assigns a score to each sentence according to their relevance to the question [149]. Alternatively, in a pure retrieval setting, the search engine retrieves directly the sentences (or small paragraphs) that may contain the answer. A2S in

this setting aims at re-ranking the passages retrieved by the search engine in such a way that the resulting list has answers ranked higher in the list of candidates [144].

1.1.3.3 Answer Generation

The Answer Generation (AG) component is crucial for an end to end Question Answering system. AG typically takes in input the answer candidate from the A2S (typically the first sentence in the A2S rank) and returns the answer to the user question. AG is application-specific as it depends on the way the answer is presented to the user. For example, when the question answering system is integrated into a search engine's web page, the answer may be presented in context by presenting an excerpt from the original document. In virtual assistants, the answer has to be conversational and coherent with the user question.

Given the variety of applications, AG's task is solved using different techniques, answer extraction for factoid question answering [107], or summarization for more complex applications or sequence generation [92].

1.1.4 Applications

Question Answering (QA) systems greatly improve the capabilities of users to access knowledge. For this reason, the task has a variety of applications. For example, it can improve the experience of using Search Engines by providing a concise and on-point answer to user questions. Additionally, it is a key component of Dialogue Systems as a big part of human-human conversation consists of questions and answers. Question Answering is successfully applied in restricted domains such as the medical domain to help caregivers provide diagnosis and medications to their patients. In these domains, Retrieval Based QA systems are often combined with high precision domain-specific ontologies [44].

The applications of QA are not only industrial, but the task has direct research implications. Given the complexity of the task, it is often used as a proxy task to test machines' capabilities to understand written text, i.e., Machine Reading Comprehension.

Another application of the Question Answering pipeline is for the task of Community Question Answering (cQA). cQA websites are online forums, such as Quora, Yahoo Answers, or Stack Exchange, where users can ask questions to the other members of the website and answer questions of other users. In this setting, a retrieval-based QA pipeline can retrieve already answered questions from the website and their answers.

1.1.4.1 Machine Reading Comprehension

Machine Reading Comprehension (MRC) is a broad and multi-faced problem that aims at studying the capabilities of machines to read and understand written text. In a similar way in which children are tested for reading comprehension, QA can be used as a mean to *test* the capabilities of machines to understand written text. Practically the task involves extracting the exact text span answering the question from a document, where the document is usually provided with the target question. Although MRC could be theoretically solved using a combination

of an A2S and an Answer Extraction component, in recent years, the task has been addressed by an independent research area aiming to determine the capabilities of end-to-end deep learning methods of identifying the correct answer directly from documents. Additionally, vertical research on MRC aims at building models capable of answering complex questions, e.g., questions that require reasoning across different documents, i.e., Multi-Hop Question Answering [150]

Even though MRC techniques are gaining more and more popularity thanks to the availability of specific datasets, A2S is more suited as a general solution for open-domain QA in a production scenario since a combination of a retrieval engine together with a sentence selector model often constitutes an industrial QA system. Several models have been proposed to adapt MRC to an end-to-end retrieval setting, e.g., see Chen et al. [19] and Kratzwald and Feuerriegel [62], the deployment of MRC systems in production is challenged by two key factors: the lack of datasets for training MRC with realistic retrieval data, and the large volume of relevant contents needed to be processed, i.e., MRC cannot efficiently process a large amount of retrieved data. In contrast, A2S research originated from the TREC competitions [144]; thus, it has targeted large databases of unstructured text from the beginning of its development.

1.1.4.2 QA in Dialogue Systems

According to Stolcke et al. [119], almost 10% of human-human conversations on the Switchboard corpora are in the form of questions and answers. From this, it is clear how QA plays a crucial role in Dialogue systems aiming to have a seamless conversation between humans and machines.

Current Question answering systems assume fixed interactions between the user and the machines, i.e., the user asks a question, and the system replies with a concise answer. However, in a dialogue system, fluid interaction between the user and the system is expected. The systems ask for clarification, and the user asks follow-up questions to search for information.

The task of building a QA system for dialogue is known as interactive Question Answering (iQA), and it is an open research problem. Quarteroni and Manandhar [96] formalized the task and proposed a possible pipeline. The pipeline makes use of a retrieval based QA system, an Answer Extraction/Generation component, and a Dialogue Manager managing the interactions between the user and the machine.

1.1.4.3 Community Question Answering

In recent years, community Question Answering (cQA) websites have gained popularity. In these websites, users can ask a question to other community members hoping for a satisfactory answer as a response. However, the answers from the users on the forum are not always optimal, and therefore this can impact the user's experience looking for information. For these reasons, in cQA, an Answer Sentence Selector (A2S) component can be used to select the relevant answer to the user question (Task A). Another problem of cQA websites is related to the number of duplicate questions present on these websites. This problem is

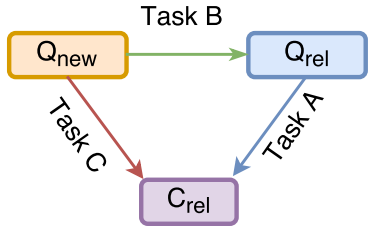

Figure 1.3: The community Question Answering task structure

a consequence of the fact that websites often provide a sub-optimal way to search for answers to a related question on the website. If users cannot easily find the answer to their new question, they will post the new question on the website, creating duplication. A variation of the retrieval-based pipeline can be used to access previously answer questions to ease these problems.

When a user submits a new question (Q_{new}), a Search Engine retrieves related questions (Q_{rel}), and a Question Selector, which is a similar architecture to the A2S, selects the Q_{rel} on the website that is a paraphrase of Q_{new} (Task B). An A2S component is then used to select the relevant user comment (C_{rel}) for Q_{new} among the answers to Q_{rel} (Task C) [89].

The final system for Task C is effectively a retrieval based Question Answering system. The user can ask a question to the system, and the system answers with a user comment to a related question. The advantage of a community Question Answering system is that it can retrieve answers to opinion questions, how-to questions, or more complex questions that users cannot be typically answered by querying a Search Engine.

1.2 A2S: Answer Sentence Selection

In Section 1.1.4, we explain why the task of Answer Sentence Selection (A2S) is a core component of many Question Answering applications. Not only A2S is the core component of the general pipeline for Open-Domain Question Answering, but it is a building block of other QA applications such as iQA and cQA pipelines.

The task of Answer Sentence Selection (A2S) can be formalized as follows: given a question q and a set of answer sentence candidates $C = \{c_1, c_2, ..., c_n\}$ the task is to assign a score s_i for each candidate c_i such that the sentence receiving the highest score is the one that contains the answer. An excerpt from an A2S dataset is presented in Table 1.1; in this example, the sentence that more likely answers the question "*How long was I Love Lucy on the air ?*" is the second. In this scenario, the model needs to assign a score to each sentence so that the second sentence is ranked above the others. The task of A2S can be, in fact, effectively modeled as a re-ranking task.

Although re-ranking is a structured output problem, most state-of-the-art approaches treat the task of A2S as point-wise classification, i.e., classifying sentences that contain the answer

How long was I Love Lucy on the air ?
I Love Lucy is an American television sitcom starring Lucille Ball, Desi Arnaz, Vivian Vance, and William Frawley .
The black-and-white series originally ran from October 15, 1951, to May 6, 1957, on the Columbia Broadcasting System (CBS).
After the series ended in 1957, however, a modified version continued for three more seasons with 13 one-hour specials, running from 1957 to 1960, known first as The Lucille Ball-Desi Arnaz Show and later in reruns as The Lucy-Desi Comedy Hour.
I Love Lucy was the most watched show in the United States in four of its six seasons, and was the first to end its run at the top of the Nielsen ratings (an accomplishment later matched by The Andy Griffith Show and Seinfeld).
I Love Lucy is still syndicated in dozens of languages across the world

Table 1.1: An example of question/answer-candidate from WikiQA. In green the answer to the question.

	WikiQA	SQuAD	NQ-LA
# questions (Q)	633	11873	6230
# sentences (C)	6165	63959	193k
% Q answered	38.39	49.92	55.47
avg. # passages	9.74	5.38	30.95
avg. Q lenght	7.28	10.02	9.38
avg. C lenght	25.36	23.75	98.76
P@1 (random)	14.43	18.34	3.24
MAP (random)	25.15	43.81	12.33
P@1 (RR)	46.09	30.54	46.06
MAP (RR)	64.21	53.53	57.30
P@1 (WO)	32.51	65.48	23.06
MAP (WO)	51.02	77.90	38.08
P@1 (WO+RR)	56.38	73.12	41.01
MAP (WO+RR)	68.25	83.60	53.98

Table 1.2: Statistics of the different datasets (the test-sets are taken into account).

as positive and all the others as negative. This design bias prevents A2S models from capturing the underlying structure of the original rank. However, in this thesis, we argue that building systems capable of capturing such information is crucial for improving the performances of efficient A2S models.

1.2.0.1 A2S Datasets

A2S datasets can be divided into two categories: retrieval based and document-based. The difference between the two categories resides in the source of the answer candidates. In the former, answer candidates are retrieved from a search engine, i.e., TrecQA [144] and, more recently, MSMarco [5]. For the latter, a search engine is often used to retrieve the

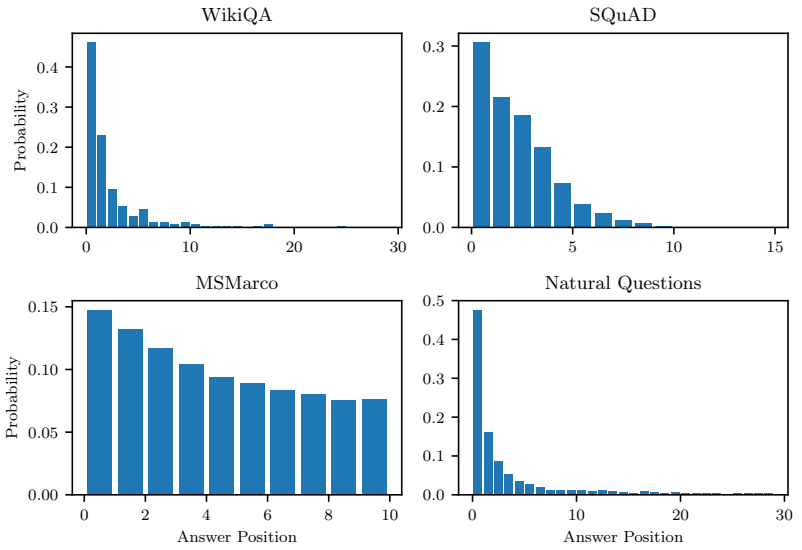

Figure 1.4: The community Question Answering task structure

relevant document, but the task is to select the relevant answer candidate from the document itself. Notable examples of document-based A2S are the WikiQA dataset [149] and the "long-answer" version of Natural Question [64].

Despite the heterogeneous nature of the datasets, both types present strong features for detecting relevant answers in the candidate set: local features of the Question and candidate pairs, with a substantial lexical overlap between the question and the answer candidate and a global structure, e.g., the original order of the sentences in the rank.

In this thesis, we study ways to exploit different structural information types, both local and global, to build accurate neural network-based models that retain high efficiency.

- **Local Structure**: The local structure of question-answers pairs includes not only the syntactic structure of the two sentences but the relational information between the words in the question and the answer.
  - *Syntactic Structure*: Before neural networks, state-of-the-art approaches to the A2S task use the question's syntactic structure and the answer. This thesis wants to study ways to include this type of information into neural network-based approaches.
  - *Relational Structure*: Relational information among the words in the question and the answer candidates are crucial for the task of question answering. One of the strongest features in A2S is the lexical overlap, i.e., whether words appear in both questions and answer candidates. The importance of this feature is highlighted in Table 7.1. We used the number of unique words in both the question and the

candidates as a single feature to rank question-answer pairs. From the table, it is clear that this feature alone significantly outperforms the random baseline in most datasets. For Squad-sent, which is an adaptation of the SQuAD v. 2.0 dataset where the task is to identify the sentence containing the answer, in particular, this feature alone identifies the sentence containing the answer 65.48% of the times. All of the recent models in the literature have tried to model such features. For example, Severyn and Moschitti [111] uses the relational feature that marks words appearing in both the question and the answer, and many state-of-the-art approaches have primarily used the attention mechanism [145, 10, 114].

- **Global Structure**: State-of-the-art approaches to A2S focus on local structure, i.e., the type of information that can be extracted from individual questions and answer candidate pairs. The pairs are decontextualized from their original document or forum thread, thus losing much useful information. In the case of community question answering, we have seen that three different tasks need to be solved to identify an answer to a new question (see Section 1.1.4.3). The three tasks are related structurally; hence a model for the task has to access this structural information.

  – *Task Structure*: In community Question Answering, the task of retrieving an answer to a new question involves solving two related tasks. A2S techniques can be applied to solve all three tasks of cQA: (A) Selecting a relevant answer among the answers posted on a website for a given question, (B) selecting a question relevant to a new question from the one that already received an answer (C) Selecting an answer from the answers to related questions that answer a new, unanswered, question.

    There is an intrinsic relation between the three tasks: intuitively, (C) is a combination of the tasks (B) and (A). This strict relation among tasks is a unique and powerful feature of the Community Question Answering, and it should be encoded into Question Answering models for cQA.

  – *Document Structure*: Another relevant feature for A2S datasets is the global structure present in the original rank. The structure of the document in a document-based dataset provides a critical signal needed for answer sentence selection. Table 1.2 shows that in the case of WikiQA, SQuAD, and Natural Questions, there is a high chance that the answer is contained in the first sentence/paragraph. This is particularly true for WikiQA and Natural Questions. In these datasets, the P@1 when using the reciprocal rank of the original sequence positions as a signal for ranking is ~ 46, i.e., the reciprocal of the original rank $1/rank$. There may be several reasons for this distribution. For example, we believe that there is an intrinsic correlation between the real world distribution of questions and the Wikipedia document structure. We believe that encyclopedic knowledge is usually organized in a way that more general information about a topic is summarized and organized at the beginning of the document.

    In contrast, by construction, the signal is less present in datasets such as SQuAD,

where annotators are asked to write questions after reading the whole paragraph. Thus the answer distribution is less skewed. However, for the same reason, it is essential to note that annotators tend to introduce more lexical overlap bias when writing questions after reading the source of the answers.

Additionally, Table 1.2 shows that the combination of the two features, word-overlap, and reciprocal rank, gives a strong baseline for all the datasets in consideration. This simple rule-based model ranks candidates according to the lexical overlap between question and candidates, and, in the case when two sentences have the same amount of overlapping words, it uses the reciprocal rank as a discriminator.

In this thesis, we claim that, in order to build efficient and accurate models for the task of Question Answering, models should be able to encode all different structural aspects of the task. To this end, we propose five different solutions that encode different aspects of the task.

1.3 Structure of the thesis

The first two chapters of the thesis (Chapter 2 and Chapter 3) provide a detailed overview of current machine learning approaches for the task of A2S. In particular, chapter 2 provides basic concepts of machine learning methods that are used throughout this thesis. In Chapter 3, we analyze the literature of A2S providing describing the building blocks of standard A2S models. Additionally, we describe two relevant architectures from the literature, the Relational CNN, and the Compare-Aggregate architecture. Section 3.5 highlights the major limiting factors of current A2S architectures to achieve high efficiency.

1.3.1 Contributions

This thesis's main contributions are grouped into two parts, depending on the type of structural information we address, i.e., Local Structure and Global Structure.

1.3.1.1 Local Structure

Chapters 4, 5, and 6 focus on **Local Structure**. In particular Chapter 4 and 5 focuses on techniques to combine and inject the syntactic structure in Neural architectures for A2S. Chapter 6 proposes an efficient solution to automatically encode the relational structure in NNs architectures.

Combining Convolutional Neural Networks and Convolution Kernels (Chapter 4)

This chapter analyzes two different machine learning methods for the task of A2S: Convolutional Neural Networks (CNNs) and Convolution Tree Kernels (CTKs). The two methods learn two different types of local information between sentences, Semantic and Syntactic similarities. Additionally, we propose a strategy for combining the two approaches into a unique model capable of achieving competitive results on benchmark datasets for A2S, WikiQA, and TrecQA. Finally, we show how the proposed solution can be applied to community Question Answering (cQA). The proposed approach ranked second at the Community Question Answering competition at Semeval 2016 [89].

Injecting Syntactic and Relational Structure in Neural Networks (Chapter 5)

In this chapter, we build on the solution proposed in the previous chapter. We propose a solution that uses a teacher-student approach to inject the Syntactic information from CTKs into the CNNs network. Although two different models are used at training time, the final model is a simple, efficient Relational CNNs. The resulting model outperforms the two single models trained independently on the final task. We tested this approach on two Community Question Answering datasets, Semeval 2016 [89] and the Quora Question-Question similarity dataset.

Cosinenet: Semantic Relation Learning (Chapter 6)

In this chapter, we propose a novel neural architecture for A2S: The Cosinenet. The model is simple and efficient (comparable to the RelCNN of Severyn and Moschitti [111]). The RelCNN architecture uses a simple Lexical Matching feature to encode the relation between the words in the question and the answer. Our approach uses a fast and efficient technique to detect this feature (the cosine similarity between the words in the two sentences). Additionally, we show that this model obtains comparable results in contrast with attention-based approaches when using retrofitted embeddings [117] while maintaining efficiency.

1.3.1.2 Global Structure

Chapters 7 and 8 provide a set of techniques that are used for encoding global structures. Chapter 7 extends our work in Chapter 6, building a Global architecture for the task of A2S that exploits the original context of the answer candidate (the document) while retaining high efficiency. Additionally, Chapter 8 proposes a technique to jointly model all the three sub-tasks of community Question Answering using a unique architecture.

Encoding the Document Structure for efficient A2S (Chapter 7)

The last contribution of this thesis is a model capable of encoding the document structure or the original rank structure for A2S. This architecture can augment different models for A2S (we tested on the CosineNet and the Compare-Aggregate) with a listwise training approach that captures the original ordering of sentences. The improved Cosinenet Model that encodes the global rank obtains results on the various ranking datasets that were never obtained by any efficient models. The resulting model achieves an average MAP of 75.62 ± 0.8 on the WikiQA dataset by training the entire model in 8.9 seconds on GPU.

Encoding the task structure of community Question Answering (Chapter 8)

In this chapter, we propose a model for cQA that exploits the structure of the various subtasks. To do that, we perform multi-task learning, which consists of solving multiple tasks simultaneously. The resulting model achieves competitive results on Task C (the harder of the three) by solving it directly and not relying on the complete pipeline.

1.3.1.3 List of Publications

This thesis is partially the result of the following peer-reviewed publications (in chronological order):

[130] "Convolutional neural networks vs. convolution kernels: Feature engineering for answer sentence reranking". Kateryna Tymoshenko, **Daniele Bonadiman**, and Alessandro Moschitti. *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies.* 2016.

[7] "Convkn at semeval-2016 task 3: Answer and question selection for question answering on Arabic and English fora." Alberto Barrón-Cedeno, **Daniele Bonadiman**, et al. *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016).* 2016.

[131] "Learning to rank non-factoid answers: Comment selection in web forums.". Kateryna Tymoshenko, **Daniele Bonadiman**, and Alessandro Moschitti. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM, 2016.* 2016.

[13] "Effective shared representations with multi-task learning for community question answering." **Daniele Bonadiman**, Antonio Uva, and Alessandro Moschitti. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers.* 2017.

[132] "Ranking kernels for structures and embeddings: A hybrid preference and classification model." Kateryna Tymoshenko, **Daniele Bonadiman**, and Alessandro Moschitti. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing.* 2017.

[135] "Injecting Relational Structural Representation in Neural Networks for Question Similarity." Antonio Uva, **Daniele Bonadiman**, and Alessandro Moschitti. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers).* 2017.

Part I

Background

Chapter 2

Machine Learning Methods

Machine Learning is the computer science area that studies algorithms and statistical methods capable of learning from data rather than relying on a set of explicit instructions. Nowadays, machine learning methods are widespread in many computer science areas and artificial intelligence. They are used to solve many computer vision problems where machines have to understand an image or video's content. Moreover, they find application in many Natural Language Processing tasks.

Machine Learning methods can be classified in different categories depending on the type of data they use and the task they solve:

- In Supervised Learning, the dataset contains the input and the desired output. The main methodologies in this category are classification and regression. In this setting, the machine learning algorithm extracts regularities, patterns in the input data to solve the final task.

- In Unsupervised Learning, the data is provided without the desired output. Therefore the model extracts general patterns in the input data without knowledge on the desired output. A popular approach in this category is unsupervised clustering, i.e., the task of aggregating input examples according to their similarities.

- In Reinforcement Learning, the models learn by interacting in an environment. For example, an agent can learn to navigate a room using a trial and error approach, hence receiving feedback from its actions, e.g., positive feedback for reaching a goal or negative when hitting a wall.

In this chapter, we describe some of the essential concepts and machine learning techniques that are used in this thesis. In particular, we briefly describe the problem of supervised learning. We provide an introduction to Neural Network methods that is the core methodology used in this dissertation. Lastly, we briefly describe Support Vector Machines and Kernel methods, which is the technique that we use to leverage syntactic information in Chapter 4 and Chapter 5.

2.1 Supervised Learning

In supervised learning, given a labeled training set $D = \{(x_i, y_i)\}_{i=1}^{n}$ the task is to identify a parametrized decision function $h_w \in \mathcal{H}$ from the hypothesis space $\mathcal{H}$, that from an input $x \in X$ assign an output $\hat{y} \in Y$ with w as parameters, i.e., $h_w : X \to Y$.

The supervised learning process is typically divided into two steps: *training* and *inference*.

Training. The training process consists of identifying the set of parameters w for the decision function h_w that better approximate the training dataset D. This consists of identifying the decision function that minimizes the average error in the training set.

Inference. During inference the model uses the best set of parameters identified with the training process, and predicts the output $\hat{y}$ for a given input example x, $\hat{y} = h_w(x)$.

2.2 Basics on Neural Networks

Neural Networks (NNs) are a kind of supervised learning algorithm. NNs are defined as a set of parametrized non-linear transformations that connects the input to the output. The simplest type of Neural Network architecture is the feed-forward network, also known as Multilayer Perceptron (MLP).

2.2.1 Feed forward Network

An MLP is composed of a set of basic mathematical units organized in layers: an input layer, a hidden layer, and the output layer. Starting from the input layer, the output of each layer is the input of the following layer. The output layer represents the final output of the network.

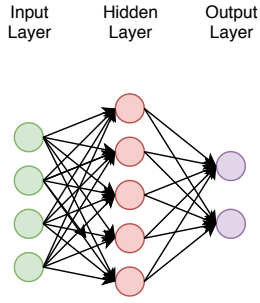

Figure 2.1: An example of Feed Forward Network

A MLP is a mathematical function $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$, where d_{in} is the size of the input and d_{out} is the size of the output. The hidden and output layer values are obtained by a linear transformation of the values from the previous followed by a non linearity. For example, the first hidden layer is computed by multiplying the input vector with a weight matrix, and adding a bias vector to the result. In our case, the full network is specified by the following

Logistic (or sigmoid)	$f(x) = \frac{1}{1+\epsilon^{-x}}$
Hyperbolic Tangent (tanh)	$f(x) = \frac{2}{1+\epsilon^{-2x}} - 1$
Rectified Linear Unit (ReLU)	$f(x) = \max(0, x)$
Softmax	$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$ for $i = 1, ..., K$

Table 2.1: Common activation functions.

equations:

$$\hat{\mathbf{y}} = f(\mathbf{x}) = o(h(\mathbf{x}))$$
$$h(\mathbf{x}) = \sigma(\mathbf{W_h}\mathbf{x} + \mathbf{b_h})$$
$$o(\mathbf{x}) = \mathbf{W_o}\mathbf{x} + \mathbf{b_o}$$

where $x \in \mathbb{R}^{d_{in}}$ is the input vector and $\hat{y} \in \mathbb{R}^{d_{out}}$ is the predicted output vector. d_{in} and d_{out} represent the input. The hidden layer it is a linear transformation of the input vector x with $\mathbf{W_h} \in \mathbb{R}^{d_h \times d_{in}}$ being the weight matrix and $\mathbf{b_h} \in \mathbb{R}^{d_h}$ the bias vector. Similarly, $\mathbf{W_o} \in \mathbb{R}^{d_{out} \times d_h}$ and$\mathbf{b_o} \in \mathbb{R}^{d_{out}}$ are the weight matrix and the bias vector for the output layer. σ is a non-linear activation function. The non-linearity is fundamental building fundamental for building deep neural networks. A combination of multiple linear functions otherwise results in a linear function itself. The output vector $\hat{\mathbf{y}}$ contains the network's predictions, i.e., the logits. The task that is solved defines which activation function and loss function to use.

2.2.1.1 Activation Functions

An activation function is a non-linear function that enables the network to learn complex, non-linear functions. Any non-linear function should be differentiable for allowing the network to train the network using stochastic gradient descent.

Table 2.1 contains some examples of activation functions. The sigmoid function can be used to squash values between 0 and 1 and model the probability of a single outcome in a network with one output unit. In contrast, the tanh function outputs values between -1 and 1. Another standard activation function is the ReLU function [88] keeps the positive part of the argument, and it is a common choice for hidden layer activation functions [39]. ReLU is piecewise linear, i.e., a combination of two linear functions; however, it acts as a non-linear activation function, and it provides competitive results in most application, despite having a is a non-differentiable point in 0.

Additionally, The softmax function is commonly applied to the final output of a neural network to obtain a probability distribution out of a vector of logits.

2.2.2 Training the Network

The output of a neural network like the MLP in Figure 2.1 is a combination of the input and the network parameters, i.e., the bias vector and the weight matrices. Although the parameters

are fixed during inference, they need to be adapted and modified to find a configuration that minimizes the error, i.e., the loss on the training examples.

At the beginning of the training process, the parameters of the network are initialized at random. During training, the network performs a *forward* pass on a given instance with the current parameters setup. The resulting logits are compared with the desired output, i.e., the training instance's gold labels, to determine the error of the model for the given example. The comparison is performed with an error function called loss function. This error is then propagated through the network to adjust its parameters and minimize the expected loss.

2.2.3 Loss Functions

During training, the network compares the output with the ground truth using a loss function. The choice of the loss function is fundamental to obtain the desired results. This choice is defined by the task and by the desired output, e.g., continuous for regression tasks or categorical for classification. Among others, for regression, a typical loss function is the Mean Squared Error (MSE):

$$\text{MSE} = \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}, \tag{2.1}$$

where $\hat{y}$ is the output of the network and y is the gold label. For classification, that is the most used for NLP problems and in this thesis, an appropriate loss is the categorical cross-entropy. This loss is used when we want to classify an input instance into one of possible classes. The cross-entropy measures the divergence between two probability distributions: the probability distribution produced by the network, and the ground truth probability distribution, which is typical endoded using a one-hot representation, i.e. all values set to zero but the one corresponding to the gold label. Therefore we have:

$$\text{CrossEnt} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(\hat{y}_{ij}), \tag{2.2}$$

where $\mathbf{y}$ is the one-hot encoded label vector, and $\hat{\mathbf{y}}$ contains the result of the network. $\hat{\mathbf{y}}$ is typically transformed into a valid probability distribution using the softmax activation function.

2.2.4 Backpropagation

Once the network and the loss function are defined, the model parameters are updated to reduce the loss function. The backward propagation or backpropagation [100] is an algorithm that iteratively adjusts the network's parameters while reducing the loss. Such an algorithm computes the derivative of the loss function with respect to each network parameter, using derivatives' chain rule. Optimization algorithms based on gradient descent can use the gradient of the loss to update the network parameters. Stochastic Gradient Descent (SGD) is a popular algorithm that draws a batch of examples from the training set, computes the loss

of the model each example in the batch, computes the gradients of the loss, and accordingly updates the parameters [15] by a small step, which is defined by the learning rate hyper-parameter, in the opposite direction of the error. Adaptive optimization methods, such as Adam [61], are also popular since they provide a faster convergence rate by adapting their internal hyper-parameters during training.

2.3 Neural Networks for NLP

Neural Networks are used in many areas of Natural Language Processing as they do not require manual feature engineering, they can leverage the capabilities of GPUs to perform fast linear algebra operations, and they can achieve results that are unmatched by other approaches in particular when trained on a massive amount of data.

A basic network for NLP is composed of two core modules: (i) A word representations module, and (ii) A sentence representation module. In this section, we present the main building blocks of Neural Network for NLP as well as recent advancement on how to build state-of-the-art models for language.

2.3.1 Word Representations

Before training a neural model for NLP, we need to induce a vocabulary V from the textual data processed by the network. Each word in the vocabulary is mapped to a d-dimensional vector $\mathbf{w} \in \mathbb{R}^d$. The numerical vectors are typically stored in a matrix of vectors, i.e., embedding matrix $\mathbf{W} \in \mathbb{R}^{d \times |V|}$. This matrix is used to lookup for the vector corresponding to each word in the text instance. Words that are not in this vocabulary are typically mapped to a special token UNK for unknown words, and therefore to a random vector. Alternatively, they can be mapped to different random vectors allocated explicitly for each word.

In most NLP tasks, the input to our neural network is a sentence $\mathbf{s}$, i.e., a sequence of words $(w_i, .., w_{|s|})$ drawn from the vocabulary V. Each word in the sequence is looked up in the word embedding matrix $\mathbf{W}$, and mapped to its distributional vector $\mathbf{w}$. Therefore, each input sentence $\mathbf{s}$ is transformed in an embedding matrix $\mathbf{S} \in \mathbb{R}^{d \times |s|}$, where each column i contains the embedding $\mathbf{w}_i$ associated with the i-th word in the sentence:

$$\mathbf{S} = \begin{bmatrix} | & | & | \\ \mathbf{w}_1 & \dots & \mathbf{w}_{|s|} \\ | & | & | \end{bmatrix}$$

After this, the neural network can apply different types of *sentence representation* to encode the interactions between the various elements of the input sentence.

2.3.1.1 Pre-trained Embeddings

The embedding matrix W can be initialized with random values and trained with back-propagation on the final task. However, the embedding matrix is sparse, and it can be difficult

to obtain good word representations. This behavior is noticeable in particular if the dataset for the target task is small.

Therefore it is common to pre-train the word embedding on a big dataset — a proxy task to exploit big datasets without needing gold labels. Conventional approaches to pre-train embedding vectors are the Skip-Gram model included in the word2vec toolkit [81] and Glove [93]. The skip-gram model operates on massive datasets, such as Wikipedia, and for each word, it predicts the words in its surrounding context, updating the word embedding of the target word accordingly. In contrast, the Glove creates dense representations of words by starting from the word's co-occurrence matrix in a huge dataset.

When used to initialize the embedding matrix W, the approaches above typically provide improvements, in contrast with random initialization, on the final network's downstream task.

2.3.2 Sentence representation

In many NLP tasks, neural network models create a fixed-size representation of the whole sentence. A simple approach for such a mapping is to aggregate word embedding vectors across dimensions. Kalchbrenner, Grefenstette, and Blunsom [55] uses the sum of the word embeddings as the aggregation operation. However, more recently, empirical results show the superiority of the averaging operation (see Deep Averaging Networks [49]):

$$\mathbf{z} = \frac{1}{|s|} \sum_{i=0}^{|s|} \mathbf{w}_i, \tag{2.3}$$

A deep feed-forward network takes the vector $\mathbf{z}$ to perform classification. Despite the effectiveness of these simple sentence representations, these approaches do not preserve the word's natural order in the sentence.

For solving specific problems, it may be essential to encode the words in the original order they appear in the sentence. Standard neural network techniques to encode such information are Convolutional Neural Networks (CNNs) and Recurrent Neural Network (RNNs).

2.3.3 Convolutional Neural Networks

By design, Convolutional Neural Networks (CNNs) [70] capture the local spatial relations of an input. Due to their efficiency they are very popular in computer vision [63, 69, 56]. At the same time, they have also been proven effective in modeling sentences [60, 55], in several NLP tasks.

In computer vision, CNN applies the same transformation over small image patches obtained by sliding over the image. While images are two dimensional, the text has only one temporal dimension, as it is a sequence of words. For textual inputs, the convolution operation is an affine transformation applied to a fixed size window, for all the words in a sequence. Similarly to feed-forward networks, such linear transformation can include a bias term and

non-linearities. Typically, out-of-sequence tokens in the window are mapped to zero vectors. This operation is known as padding and results in having an output sequence of the same length of the input sequence. Without passing, the resolution will depend on the window size and sequence length.

A single convolution operation over a fixed size window is known as convolution filter or kernel. Let $\mathbf{w}_i \in \mathbb{R}^d$ be the d-dimensional word vector corresponding to the i-th word in S. The sentence, which contains n words, is represented as a concatenation of vectors:

$$\mathbf{w}_{1:n} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n]. \tag{2.4}$$

A windows of n words around the word i is defined as $\mathbf{w}_{i-\frac{n}{2}:i+\frac{n}{2}}$ and it is the concatenation of the vectors $\mathbf{w}_i - \frac{n}{2}, \ldots, \mathbf{w}_i, \ldots, \mathbf{w}_{i+\frac{n}{2}}$. Then, a convolution operation is the result of the dot product between a filter $\mathbf{W} \in \mathbb{R}^{d_w \times d_h}$, where d_w consists in the size of the word embedding d times the length of the window and d_h is the size of the output vector $\mathbf{c} \in \mathbb{R}^d_h$. The convolution operation is represented as:

$$\mathbf{c} = \sigma(\mathbf{U} \cdot \mathbf{w}_{i-\frac{n}{2}:i+\frac{n}{2}} + b). \tag{2.5}$$

$b \in \mathbb{R}^d_h$ is a bias vector and σ is a non-linear activation function. The filter is applied to all the word windows from the sentence $\mathbf{w}_{1:n}$ and the result is the following sentence representation:

$$\mathbf{C} = \begin{bmatrix} | & | & | \\ \mathbf{c}_1 & \ldots & \mathbf{c}_n \\ | & | & | \end{bmatrix}$$

with $\mathbf{C} \in \mathbb{R}^{n \times d_h}$. This matrix is typically encoded in a single vector using a max-pooling operation [26]. Alternatively, the network can use Average pooling (see Equation 2.3) to map the sentence matrix to a fixed size vector. This vector constitutes the fixed size sentence representation, which can be further transformed, or used for classification.

2.3.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [31] are one of the main approaches for modeling sequences, and they have been widely adopted for NLP tasks. Vanilla RNNs consume a sequence of vectors one step at the time and update their internal state h_t as a function of the new input x_t and their previous internal state h_{t-1}.

$$h_t = \sigma(Wx_t + Vh_{t-1} + b) \tag{2.6}$$

This recursive structure is theoretically capable of encoding the entire history of previous states in the vector h_t. However, these networks may suffer from the vanishing gradient problem [8]. This problem is typically mitigated using popular RNN variants, such as the Long Short Term Memory (LSTM) network [45] or Gated Recurrent Unit (GRU) [23].

2.3.4.1 Long Short-Term Memory

An LSTM uses a gating mechanism to control the amount of information from the input that affects its internal state, the amount of unnecessary information in the internal state, and how the internal state affects the output of the network.

For each time step t the cell state c_t and $h_t \in \mathbb{R}^{d_1}$ are computed for each element of the sequence by iteratively applying the following equations, where $h_0 = c_0 = 0$:

$$
\begin{aligned}
o_t &= \sigma(W_o x_t + V_o h_{t-1} + b_o) \\
f_t &= \sigma(W_f x_t + V_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + V_i h_{t-1} + b_i) \\
g_t &= \tanh(W_g x_t + V_g h_{t-1} + b_g) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

The W, V matrices, and the b vectors are parameters of the model. $\odot$ denotes element-wise (Hadamard) multiplication. σ is the sigmoid function, and $tanh$ is the hyperbolic tangent function. o_t, f_t, and i_t are the output, forget and input gates respectively, since their values are bounded between 0 and 1, and control the rescaling of vectors.

2.3.4.2 The Gated Recurrent Unit

The Gated Recurrent Unit (GRU) [23] is similar to LSTM, it retains the same performance but has less parameters, thus faster to train. In the following equations, s_t defines the state at time-step t. Given a sequence of input vectors, the GRU computes a sequence of states $(s_1, ..., s_T)$ according to:

$$
\begin{aligned}
z &= \sigma(x_t U^z + s_{t-1} W^z) \\
r &= \sigma(x_t U^r + s_{t-1} W^r) \\
h &= \tanh(x_t U^h + (s_{t-1} \odot r) W^h) \\
s_t &= (1 - z) \odot h + z \odot s_{t-1}
\end{aligned}
$$

This recurrent unit has an update, z, and reset gate, r, and does not have an internal memory beside the internal state. The U and W matrices are parameters of the model.

2.3.4.3 Bi-Directional RNNs

LSTMs and GRUs iterate over the input in one direction. Therefore earlier hidden states of the network do not have access to future steps. Bidirectional RNNs [103] solve this issue by iterating over the sequence in both directions, i.e., forward and backward. At any given step, the state will be the concatenation of the state for each direction at that step and will contain useful information from both the left and right context of a word.

2.3.5 Attention Mechanism

Another essential component of modern NLP Neural Networks models is the attention mechanism. This method was first applied for machine translation [4] to automatically align the words of the source sentence and the words in the target sequence.

Given a query vector $q \in \mathbb{R}^d$ a keys matrix $K \in \mathbb{R}^{d \times N}$ and a value matrix $V \in \mathbb{R}^{d \times N}$ the attention mechanism computes the similarity between q and each vector in the matrix $k_i \in K$, using an alignment function $a : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such as the dot product of the two vectors, and normalizing the resulting value using a softmax function:

$$a_i = \frac{e^{a(q,k_i)}}{\sum_{j=1}^{K} e^{a(q,k_j)}} \tag{2.7}$$

a_i, in this case, represents a probability distribution, i.e., the relevance of the key k_i with respect to the query q. The probability distribution vector $\mathbf{a} = \{a_1, \ldots, a_n\}$ is used to perform a weighted sum of all the value vectors, i.e. $\mathbf{a}V$.

In the context of Machine Translation (MT), we can use the vector representing the words to be predicted as a query q and the words in the source sequence as both keys K and values V.

2.3.6 Transformers

Nowadays, many different models for NLP use the attention mechanism. In particular, the Transformer architecture in Vaswani et al. [136] uses an attention mechanism, i.e., self-attention, to generate a sentence representation.

In *self-attention* the query q is a vector from the sentence S, and performs attention on the sentence itself, therefore $S = Q = K = V$. This operation is performed multiple times with different linear transformation of the embedding space – the matrix S is multiplied by a learnable weight matrix – and the output of the multiple attention is concatenated. This operation is called multi-head self-attention and captures multiple input sentences' alignments, capturing various nuances of the input example.

The transformer is a multi-layer network. It performs a self-attention step at each layer, followed by a time-distributed feed-forward layer, which means that the same multi-layer perceptron process each word in the sequence. Self-attention and the time-distributed feed-forward layer do not encode temporal information. To encode such information the, transformer concatenates to each embedding in the sentence a temporal embedding representing the position of the word in the original sequence.

Transformers have been proved successful in many areas of NLP. The main reasons are that they can be easily parallelizable on GPU, and therefore it is possible to stack many layers of transformer and train on massive datasets without losing huge performance drawbacks.

2.3.7 Deep Contextualized Language Models

Typically, sentence encoders are randomly initialized and trained to solve a downstream task on the final task's training set. Similarly, to what is done for embeddings (see Section 2.3.1.1, it is possible to pre-train sentence encoders using a proxy task on massive datasets. For example, in ELMo [94], a deep biRNN is trained as a language modeling task, i.e., predicting a word in a sequence given the context on the left and the right. This model is then used in place of the pretrained embeddings in the network for the downstream task. This model has many advantages in contrast with standard embeddings. Among others, (i) at inference time can create a word representation of the word in its context, (ii) it uses character representation of words. Therefore, it can generalize on unseen words without the need of a dedicated uninformative UNK token.

Despite its effectiveness, ELMo suffers from performance issues as it operates at a character level and uses a stack of BiLSTM that are not parallelizable to encode the sentence representation. Therefore, recent approaches uses a transformer-based sentence encoder for pre-training, e.g, GPT [97], BERT [29], RoBERTa [74], XLNet [151].

Among others, the most common approach to pre-train sentence representation is BERT architecture. BERT uses a stack of 12 or 24 layers of transformers to solve a masked language model task and a next sentence prediction task. In masked language models, one or more words in the input sentence are replaced with a special *[MASK]* token, and the task is to predict the original words in those positions. The sentence prediction task further enriches the capabilities of the model. In this setting, the sentence encoders take in input two sentences (divided by a special *[SEP]* token), and the task is to predict if the second sentence was the sentence after the first in the original document.

In contrast to ELMo, this model does not use character embeddings but sub-word embeddings to reduce the vocabulary size and deal with unknown words. Like RoBERTa or XLNet, other architectures extended the same framework with small changes in the training process and the architecture to achieve better results. However, the base idea and the framework remain unchanged.

Once pre-trained on large corpora (Wikipedia or Commoncrawl), these models can be used for finetuning on a downstream task by replacing the last layer of the network and updating the network parameters using SGD a very low learning rate. As of today, this is the de facto state-of-the-art for Natural Language Processing as these architecture are capable of achieving very high results on downstream tasks that are unmatched by other architectures that used standard pretrained word embeddings [29, 74].

Despite their success, these models have many parameters (up to 340 million for BERT Large). Therefore they require much time and powerful GPUs for both pre-training and finetuning. Therefore, these models have a substantial environmental footprint, as pointed out by Strubell, Ganesh, and McCallum [120].

2.4 Support Vector Machines and Kernel Methods

Another Machine Learning method used in this thesis is Binary Support Vector Machines (SVM). An SVM is defined as follow:

$$f(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x}) \tag{2.8}$$

where $\mathbf{x} \in \mathbb{R}^d$ is the input vector and $\mathbf{w} \in \mathbb{R}^d$ the weights of the model. The above formula defines a linear decision hyperplane that separates the examples in the input space. However, in most real-world problems, the input space is not linearly separable. To classify examples in a complex input space X, we need to be mapped in an hyperspace $\phi(x) \in \mathbb{R}^d$ where the examples are linearly separable, using a feature map function $\phi : X \to \mathbb{R}^d$. This operation is doable for many simple problems. However, in complex problems, the function ϕ may be unknown or intractable. The kernel trick was introduced for this reason.

Kernels methods are a set of functions that operates in complex input spaces. A kernel function $K : X \times X \to \mathbb{R}$ maps two input examples to a single score. This function represents the dot product between the feature map $\phi : X \to \mathbb{R}^d$ of two input examples in complex input space X, i.e., $\phi(x)^T\phi(x')$. The kernel function $K(x, x') = \phi(x)^T\phi(x')$ allows computing this dot product without knowing the explicit form of ϕ.

This function is not directly usable in SVM as defined in Equation 2.8. SVM in its dual form however has a dot product operation in the input space. The dual formulation of SVM $\mathbf{w}$ is defined as a linear combination of a subset of the training instances, i.e., the support vectors:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x_i} \tag{2.9}$$

therefore we substitute w in Equation 2.8 to obtain the dual form of SVM inference function:

$$f(\mathbf{x}) = sign(\sum_i \alpha_i y_i \mathbf{x_i}^T\mathbf{x}) \tag{2.10}$$

It is important to notice that this formulation of the SVM function includes a dot product operation in the input space. From that this dot product can be replaced with the kernel function $K(\mathbf{x_i}, \mathbf{x_j})$,

$$f(\mathbf{x}) = sign(\sum_i \alpha_i y_i K(\mathbf{x_i}, \mathbf{x})), \tag{2.11}$$

and, therefore, implicitly performing the dot product in a new feature space defined by ϕ.

2.4.1 Kernel Functions

The choice of Kernel function to use depends on the task and the type of input data. Some examples of commonly used kernel functions are:

- *Linear*: $K(\mathbf{x_i}, \mathbf{x}) = \mathbf{x_i}^T\mathbf{x}$
- *Polynomial*: $K(\mathbf{x_i}, \mathbf{x}) = (1 + \mathbf{x_i}^T\mathbf{x})^d$
- *Gaussian/RBF*: $K(\mathbf{x_i}, \mathbf{x}) = exp(-||\mathbf{x_i} - \mathbf{x}||/2\sigma)$

All the kernels above consider $\mathbf{x}$ as numerical vectors of fixed size. However, given that the kernel function only requires to provide a dot product between objects in the implicit input space, it can be applied to other types of input objects, e.g., lists, trees, or graphs. In the next section, we describe a type of kernel widely used in natural language processing applications, i.e., the tree kernel.

2.4.2 Tree Kernels

The Tree Kernels are a set of functions that can detect the similarity between two trees by counting the number of common sub-parts.

Convolution Tree Kernels (TKs) are a type of Tree Kernels that compute the number of sub-structures between two trees T_1 and T_2 without explicitly enumerating all the possible tree fragments, which would be a costly operation.

Let $\mathcal{T} = \{t_1, ..., t_{|\mathcal{T}|}\}$ be the set of all possible trees in the space of structures, and $\chi_i(n)$ an indicator function, which is equal to 1 if the target t_i is rooted ad a node n, and equal to 0 otherwise. We can define a tree kernel over T_1 and T_2 as:

$$K_{TK}(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \tag{2.12}$$

where N_{T_1} and N_{T_2} are the sets of nodes of the T_1 and T_2 trees, and

$$\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{T}|} \chi_i(n_1)\chi_i(n_2) \tag{2.13}$$

computes the number of common tree fragments rooted at the n_1 and n_2 nodes. In general, the specification of eq. 2.13 determines the TK expressivity. The Tree Kernel function K_{TK} function can be implemented efficiently in the SVM framework. This technique allows training and performing inference over Tree structures, therefore encoding sentences' syntactic structure in machine learning algorithms.

Chapter 3

Answer Sentence Selection

The task of Answer Sentence Selection (A2S) consists in assigning a score s_i to each answer candidate c_i in a collection of sentences $C = \{c_1, c_2, \ldots, c_n\}$ such that the sentence receiving the higher score is the sentence that most likely answers the question q. In this framework, A2S is a short-text ranking problem.

In Section 3.1 we briefly introduce A2S as a task of Short Text Reranking. Section 3.2 and Section 3.3 presents the most relevant machine learning frameworks to create models for A2S. Section 3.4 presents the three most representative networks for A2S. Finally, In Section 3.5, we present an analysis regarding the efficiency of various models for A2S.

3.1 Short Text Reranking

As previously introduced, Short-Text Reranking is a task that consists in assigning a score s_i to each candidate c_i in a collection of sentences $C = \{c_1, c_2, \ldots, c_n\}$ such that the sentence receiving the higher score is the sentence the most relevant candidate for the query q.

This task is typically performed by building a machine learning model $f(\cdot)$ that, during inference: (i) takes as input the query q and the list of candidate C, i.e., $f(q, C)$, (ii) returns a score s_i for each candidates $s_1, \ldots, s_n = f(q, C)$., and (iii) sorts the candidates C according to the relevance to the score s_i. The resulting list of candidates will have the candidate receiving the highest score in the rank's top position.

3.1.1 Evaluation

The performance of the decision function $f(\cdot)$ can be tested using standard retrieval metrics on a labeled test set, i.e., Precision at 1 (P@1), Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP). The test set T is set of triplet $<q_j, C_j, Y_j> \in T$, where q_j is the query, C_j is a list of candidates, and Y_j is a list of gold labels that indicates whether each candidate $c_j \in C_j$ is a good match for the query q_j, i.e., a list of binary values 1 or 0. For each triplet, the machine learning model produces a list of scores $S_j = f(q_j, C_j)$ and sorts both C_j and Y_j according to the scores S_j. The sorted list Y'_j is the input of the evaluation functions.

Precision at 1 (P@1): estimates the percentage of queries that have a "positive" candidate ranked at the first position of the rank.

$$P@1 = \frac{1}{|T|}\sum_{j=1}^{|T|} y'_{j,1}, \tag{3.1}$$

, where $y'_{j,1}$ is the label of the candidates, ranked first.

Mean Reciprocal Rank (MRR): P@1 only takes into consideration the first candidates in the final rank. In contrast, MRR assigns a small weight, i.e., the reciprocal of the rank $1/r$, to the elements that received a lower rank in the sequence. For example, if the good candidate for the query q_j is ranked second, the metric assigns $1/2$ as a value for that query.

$$MRR = \frac{1}{|T|}\sum_{j=1}^{|T|} \frac{y'_{j,i}}{i}, \tag{3.2}$$

where i is the position of the first positive candidate in the rank.

Mean Average Precision (MAP): Similarly to MRR, the MAP takes in consideration the entire rank in computing the score. In the case that there are multiple positive candidates for a query q_j, MAP assigns an higher score to a model capable of ranking all the positives before the negatives, i.e.,:

$$MAP = \frac{1}{|T|}\sum_{j=1}^{|T|} \frac{\sum_{i=1}^{|Y'_j|} y'_{j,i} \sum_{t=1}^{i} \frac{y'_{j,t}}{t}}{\sum_{i=1}^{|Y'_j|} y'_{j,i}} \tag{3.3}$$

3.1.2 Training

There are three different ways to train machine learning models for re-ranking:

Pointwise Ranking is the simplest way to approach a ranking problem [109, 124]. It is, in fact, a binary classification problem where each example in the training set is a triple $\langle q, c_i, y_i\rangle$, where q is the question, c_i is an answer candidate, and y_i is the label indicating whether c_i is an answer of q. During inference the final rank is defined by score assigned by the classifier to each $\langle q, c_i\rangle$ pair $s_i = f(q, c_i)$.

Pairwise Ranking is a more advanced approach to the ranking problem [99, 47]. The pairwise ranking consists in training a model that scores positive input pairs $\langle q, c_+\rangle$ higher than negative pairs $\langle q, c_-\rangle$, with a certain margin, ϵ:

$$f(q, c_+) \geq f(q, c_-) + \epsilon \tag{3.4}$$

This approach better models the task of ranking by comparing positive and negative pairs.

However, it requires more training data to train since at least a positive and a negative example is required for each question q.

Listwise Ranking uses the whole list of candidates for each question q [10]. This strategy provides a better approximation of the ranking task, exploiting all the possible comparisons between the candidates.

The difference between the approaches above is how the model processes the training instances, which is used to compute the error. This choice can have an impact on the training process and on how the dataset is built. The pointwise approach requires a dataset of positive and negative pairs of questions. The pairwise approach requires that for each question, a positive and a negative pair is present. The listwise approach is the more demanding in terms of annotation, i.e., the annotator needs to annotate an entire rank of candidates for each input question.

3.1.3 Machine Learning Methods for A2S

Although different machine learning models use different training strategies, nature, and structure, the model $f(\cdot)$ remains largely unchanged among different approaches.

Early approaches on short-text re-ranking use feature-based linear classifiers with features vectors as input [123, 122]. Surdeanu, Ciaramita, and Zaragoza [123] in particular uses a large amount of (i) *similarity* features (mainly lexical and syntactic similarity features), (ii) *translation* features (such as the probability that q is a translation of a_i), (iii) *density and frequency* features (that counts the amount of overlapping words and text spans between the two sentences) (iv) and *web correlation* features (that counts the amount occurrences of a given answer in the web). Although these feature-based approaches have been proved competitive at question answering, they require a huge effort to define and select the end task features. Despite using fast linear classifiers, some of the features are computationally expensive to compute, particularly web correlation features that require interaction with the web, and therefore not being suitable in a production environment where high throughput is often required.

In recent years, most short-text re-ranking approaches use techniques capable of automatically extracting features from the input sentences. The main approaches following this paradigm are tree kernel-based models and neural networks.

3.2 Tree Kernel Based approaches

In Section 2.4, we described how Kernel Methods are used to perform classification or inference in complex input spaces. In particular, Kernels allows training on input organized in structure rather than feature vectors.

The syntactic structure of sentences – which is how words are organized to convey the intended meaning – plays a crucial role in many Natural Language Processing tasks [85, 107]. The Syntactic Structure of sentences can be represented using trees (see. Figure 3.2).

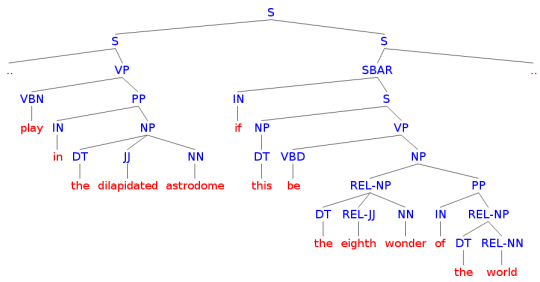

Figure 3.1: Example of constituency tree

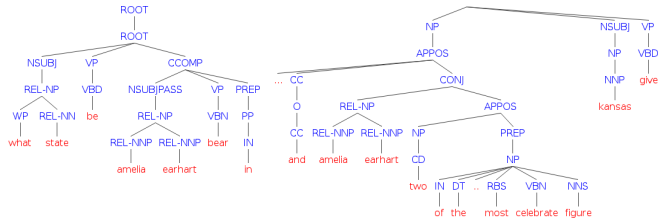

Figure 3.2: Example of dependency tree of a Question and Answer candidate

For A2S, tree kernel exploits the syntactic structures of the question and the answer candidate.

3.2.1 Relational Structural Representation for QA

In the literature there are three syntactic structures that are used for A2S: Constituency Tree, Dependency Tree, and Shallow tree [86, 106].

Constituency trees and the Dependency tree both represent the syntactic structure of sentences. The main difference between the two approaches is that the first models the implicit phrase structure of sentences, whereas the seconds look at the dependencies between the words in the sequence. An example of a constituency tree is represented in Figure 3.1 and an example of a Dependency Tree is represented in Figure 3.2. Despite constituency trees and dependencies tree are the most expressive types of syntactic structure Severyn and Moschitti [107] shows that a simple (shallow) syntactic representation of the sentence minimizes the noise introduced in the training process and provides the best results for A2S. The shallow representation uses part-of-speech information and the chunk information, which is the first representation from the constituency tree. The shallow tree is represented in Figure 3.3.

Syntactic trees encode the sentence structure, but they do not encode relational information between the constituents of the question and the answers, i.e., if a word or a chunk appears in both sentences. This feature is relevant, as previously pointed out in [107]. Severyn and Moschitti [107] proposes to encode this relational information by concatenating to the label of the shared node in the tree a special token $[REL]$. Tymoshenko and Moschitti [133] further extended this approach by marking related words in the tree that not only have a direct lexical

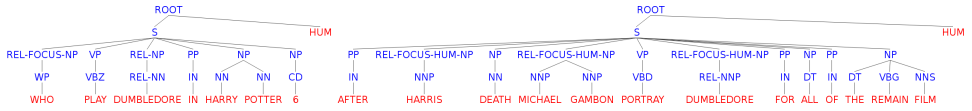

Figure 3.3: Shallow chunk-based tree for the Q/AP pair in the running example.

overlap, but they are related in a Knowledge Graph.

These structures represent the question and answer candidates and can be used as input of standard Tree Kernel functions for classification.

3.2.2 Ranking with Kernels

Kernel Methods for QA are typically trained using a pairwise ranking approach [87, 107, 133]. Given two pairs $o_1 =< q_i, c_i >$ and $o_2 =< q_j, c_j >$, and a pair of support vectors o'_1 and o'_2. The *preference kernel* [116] is defined as follows:

$$P_K(\langle o_1, o_2\rangle, \langle o'_1, o'_2\rangle) = K(o_1, o'_1) + K(o_2, o'_2) - K(o_1, o'_2) - K(o_2, o'_1) \tag{3.5}$$

where $K(o_i, o_j) = TK(q_i, q_j) + TK(c_i, c'_j)$ and TK is a tree kernel function.

3.3 Deep Learning Models

In recent years, neural architectures have been successfully applied to the task of short-text re-ranking, becoming the new de facto state-of-the-art [75, 47, 109, 124]. Despite some differences between approaches, they share the same underlying structure (Figure **??**). Given a question and a candidate answer, the model (i) Encodes the words in the question and answer-candidate into word embeddings sentence matrices (ii) it captures inter-sentence relational information using relational or lexical features; (iii) it intra-sentence relations using some class of sentence encoder (CNN or RNN), (iv) it applies pooling on the resulting question and answer sentences, and (v) finally, a sentence matcher maps the sentence representations to a score representing the relatedness of the answer to the question.

Steps (i), (ii), and (v) are present in every network for short-text ranking. Depending on the architecture, steps (ii) and (iii) are optional [47], they can be repeated multiple times [157, 36], and they can have a different order, i.e., the word matching step is performed after the sentence encoding [101].

3.3.1 Sentence Encoders

A Sentence Encoder is the Answer Sentence Selector module that learns and encodes infra-sentence relations between words of each sentence. In section 2.3.2, we presented different types of sentence encoders, for example, Convolutional Neural Network (CNN) encoder that

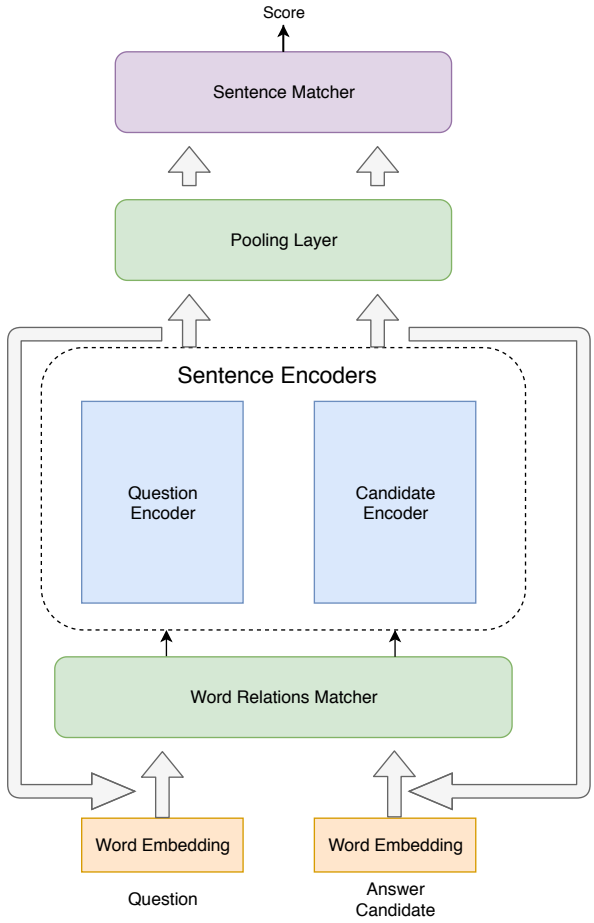

Figure 3.4: The high-level structure of Deep Learning architectures for A2S

captures local ordering information of the word in the sentence, and Recurrent Neural Network encoders that can model the global ordering of words in the sequence.

Hu et al. [47], Yu et al. [160] and Severyn and Moschitti [109] were the first to use CNN encoders encode the question and answer pairs to obtain results comparable to the previous, feature-based, state-of-the-art. Other more recent approaches [145, 10], used a set of CNNs encoder with different window sizes to capture the local ordering of words at different resolutions, i.e., window sizes of $1, 2, 3, 4, 5$.

In contrast with the CNN approaches, several other models [139, 24, 128, 115, 101] uses LSTM-based or BiLSTM-based encoders. Additionally, Tan et al. [124] combines a BiLSTM encoder followed by a CNN. Despite there is not clear winner between CNN encoders and BiLSTM encoders at Short Text Reranking, [101] shows that basic CNN encoders typically result in better performance on the A2S datasets WikiQA and TrecQA.

Typically, Sentence Encoders process the question and the answer candidate independently, using two different sentence encoders. However, Wang and Nyberg [139] and Cohen and Croft [24] concatenates the question and candidate word embeddings in a unique embedding matrix and separates the two sentences with a separation token (i.e., $<s>$). Novel methods have adopted this approach, e.g., [36], that use a pretrained Transformer sentence encoder, such as BERT [29] or RoBERTa [74]. Pretrained Transformer based approaches tend to achieve overall better results than previous approaches on benchmark datasets, e.g., WikiQA. However, they have many more parameters, so they are much slower than CNN based approaches.

3.3.2 Encoding Word Relations

Sentence encoders capture inter-sentence information. However, the latent information captured by the encoders in the question and candidate representations does not suffice at capturing infra-sentence relations between words. For example, simple CNN baselines that do not use infra-sentence word-level features fail at achieving competitive performances on benchmark datasets. This problem was empirically shown by Yu et al. [160] and Severyn and Moschitti [109] and further confirmed in Chen et al. [21], where the authors explored the impact of standard lexical matching features. In contrast with feature-based approaches, lexical infra-sentence information can be automatically inferred with an attention mechanism [157, 101, 145]. Additionally, Chen et al. [21] shows that lexical matching features and attention do not provide the same information since, when combined, the two techniques provide overall better results on standard benchmarks than the individual techniques.

3.3.2.1 Lexical Matching Features

Yu et al. [160] is the first model combining CNN encoders with lexical matching and similarity features. The authors train a logistic regression that combines the score of a neural network model with a simple feature that counts the number of co-occurring words in the question and the answer candidate, and a third feature that uses the IDF-weighted counts of co-occurring words. Severyn and Moschitti [109] adopts a similar approach, the count of overlapping words is concatenated with the question and candidate representations to be passed to the final scoring layer (after the pooling layer). Differently from Yu et al. [160], Severyn and Moschitti [109] includes the feature in the model without requiring a two-step training process,i.e., training the neural network and using its score as a feature in logistic regression. Chen et al. [21] concatenates 21 lexical matching, semantic matching, and readability features to the question and answer representation. The inclusion of such features provides a consistent performance improvement over the baseline models.

Severyn and Moschitti [110] uses a word-level binary feature that indicates whether a word in the question appears in the answer and vice versa. The feature is concatenated to the word embedding of each word (see. Section 3.4.1 for more details), before the Sentence Encoder.

3.3.2.2 Attention

The adoption of attention mechanisms was recently proposed for the task of question answering [157, 124]. The attention mechanism is a technique proposed and popularized in Bahdanau, Cho, and Bengio [4] for the task of machine translation. This technique is used to automatically infer an alignment between the source text and the target language text. Unfortunately, we do not have any strict alignment between question and answer candidate in short text re-ranking. For example, a sentence that is not an answer to the question could not have any alignment. Despite this fact, attention has been proven an effective way to extract word matching features, e.g., see [157, 145].

Yin et al. [157] and Wang and Jiang [145] align all the words in the question with all the words in the answers. They use each word in the question as a query vector q for individually attending over the answer words, thus inferring a probability distribution. The probability distribution is used to create a vector that is the weighted average of the candidate's word vectors. The same process is done between answer and question.

3.3.3 Pooling

The sentence representation in output of the sentence encoder is a new sentence matrix that encodes at each position a contextualized representation of the word. Most neural network based approaches uses standard pooling technique to reduce the sentence matrix to a fixed size vector, typically Average Pooling [47, 160, 139, 24] or Max Pooling [145, 10, 110, 109].

An advanced pooling strategy technique was proposed to exploit the intra-sentence interactions between the question and the candidate words. A sentence-level approach with attention has been proposed by Tan et al. [124]. The encoder of the question (an LSTM based encoder with average pooling) generates a unique vector for the question used as a query for the attention mechanism over the answer candidate. This attention mechanism generates a probability distribution representing the importance of each word in the candidate answer with respect to the entire question. This distribution is then used to perform a weighted average of the words in the candidate. The same can be done between the candidate and the question, thus generating a sentence representation conditioned on the representation of the other sentence, e.g., an answer candidate representation conditioned on the question.

A different pooling strategy making use of attention, has been proposed by Santos et al. [101]. A dot product operation between all the words in the question, and the answer is applied to generate a matrix representing the relational information between the question and the answer, similarly to the word-level attention model. The model performs a max-pooling operation over the attention matrix's two dimensions before applying the softmax operation, resulting in distribution over the question and candidate word representations. Intuitively, the model learns to assign a high probability to words with the highest similarity score with another word in the related sentence.

3.3.4 Sentence Matching and Scoring

The pooling operation typically provides an n-dimensional vector representation for both the question and the candidate. The sentence matching component takes these two representations and combines them for scoring. A simple way to score the question and the candidate is to map the representation in the same vector space. This scoring is achievable by using the same sentence encoder for both the sentences, i.e., both the question and the candidate encoder uses the same parameters. This technique is known as the Siamese network, [22]. Once mapped to the same vector space, the question and answers relatedness can be measured with standard similarity functions, e.g., cosine similarity:

$$\mathbf{s}_{q,a} = cos(\boldsymbol{q}, \boldsymbol{a}) \tag{3.6}$$

or negative distances, e.g., the negative Euclidean distance:

$$s(q, a) = -\|\boldsymbol{q} - \boldsymbol{a}\|_2. \tag{3.7}$$

A simple extension to the use of similarity functions is to use a bilinear layer as proposed by [160]. A bilinear layer is a similarity function that can be trained, enabling the learning of more complex relations in the vector space. Hence, the scoring function of Yu et al. [160] is formalized as:

$$s(q, a) = \sigma(\boldsymbol{q}^T M \boldsymbol{a} + b) \tag{3.8}$$

where $q \in \mathbb{R}^{dim}$ and $a \in \mathbb{R}^{dim}$ are the question and answer representation, $M \in \mathbb{R}^{dim \times dim}$ is the weight matrix and $b \in \mathbb{R}$ is the bias. $\sigma : \mathbb{R} \rightarrow [0, 1]$ is the sigmoid function.

Another simple but effective approach is proposed by Hu et al. [47]. The model simply concatenates the question and the answer representation and use a Multi-Layer Perceptron (MLP) to score the combined vector:

$$s(q, a) = MLP([\boldsymbol{q}; \boldsymbol{a}]). \tag{3.9}$$

Severyn and Moschitti [109] combines the two approaches by using an MLP with the concatenation of question, answer, and a bilinear layer score:

$$s(q, a) = MLP([\boldsymbol{q}; \boldsymbol{q}^T M \boldsymbol{a}; \boldsymbol{a}]). \tag{3.10}$$

This approach differs from the aforementioned Siamese approaches as the sentence encoders are independent and do not share the weights. Hence, the sentence representations are in two different vector spaces. Intuitively, using two different encoders can help since the question answering task is not symmetrical ad the syntactic structure of question and answer is

inherently different.

3.3.5 Training

The most straightforward approach for training a network for Answer Sentence Selection is using the pointwise approach. In the pointwise approach, the model takes as input a question-candidate pair and learns to classify it as positive of the candidate answer the question or as negative otherwise.

Hu et al. [47] uses a ranking-based loss as objective function, i.e. a *pairwise approach*:

$$loss(q, a^+, a^-) = max(0, M + s(q, a^-) - s(q.a^+)) \tag{3.11}$$

where M is a constant margin ($M = 1$ in [47]) a^- is a not valid answer for the question q whereas a^+ is an answer for q.

Although the former has been widely used in several models, there is no apparent difference in performance between the two losses. An advantage of the triplet loss is that it automatically balances the dataset; for each positive example, a negative is used. However, the loss does not reflect the actual distribution of data since, depending on the negative examples that are chosen for comparison, the results can vary. [99] shows that depending on the dataset using randomly sampled negatives, hard negatives (a^- that is close to q in the vector space) or a combination of both can lead to different results. An advantage of using the classification loss of [109] is that it is not necessary to sample negatives for the training.

Another kind of loss that have been used by Wang and Jiang [145] and later by Bian et al. [10] is the listwise approach. The listwise approach scores all the candidate answers $a_1, .., a_n$ for the same q questions $s_1, .., s_n$ applies a softmax on top of the scores mapping them to probabilities and uses the KL-divergence loss to assign a high probability (therefore score) to the answers for the question. Additionally, Wang and Jiang [145] generated the scores $s_1, .., s_n$ by concatenating the representations for the answer candidates $a_1, .., a_n$ for all the candidates into a unique vector to be passed to a multi-layer perceptron, i.e.:

$$[s_1, .., s_n] = S = MLP([a_1, .., a_n]) \tag{3.12}$$

Although the model should be able to capture the global ranking structure, this technique suffers from two drawbacks: it requires the number of candidate answer to be fixed since it does not share weights, and it can not be used for datasets that have an artificial ranking (for examples in TrecQA the dataset always provide the answer at the top positions of the ranking).

3.3.5.1 Transformers

Recent approaches to A2S uses a Transformer based encoder [36]. This encoder uses a stack of several layers of self-attention and a time-distributed layer, up to 24, when using pre-trained

BERT. In the Transformer architecture question and candidate, answers are concatenated and separated by a unique $<sep>$ token. The self-attention layer is word-level attention that creates an alignment between every word in the sentence and all the other words in the same sequence. Since the question and the answer candidate are concatenated, this attention mechanism capture infra-sentence and intra-sentence word relations. The feed-forward layer is a Convolutional Neural Network with a window size of 1. By performing this operation several times, the models can capture higher-level relations between the words in the question and answer. The ordering of words in the sequence is captured by explicitly encoding the temporal information using temporal embeddings concatenated with the input sequence.

The pooling layer for the Transformer model could use Average Pooling or more complex pooling strategies [66]. However, the common strategy is to include a unique token $<cls>$ in the input sequence and use the vector in the final layer that corresponds to the $<cls>$ token as a representation of the question-candidate pair.

3.3.5.2 Transfer Learning and Task Adaptation

Transformer architectures typically have several layers, and therefore a high number of parameters to be trained. Standard approaches with Transformer typically use pre-training on massive datasets, at the web-scale, as a strategy to initialize pre-initialize the weight of the model for the final task, e.g., BERT [29], RoBERTa [74]. This training strategy is typically done using language modeling as a proxy task.

The resulting model is then fine-tuned for A2S on the target dataset, e.g., WikiQA. This technique is known as *transfer learning*, i.e., transferring the knowledge acquired on a dataset to the target dataset. However, Garg, Vu, and Moschitti [36] showed that using the general pretrained weight of BERT of RoBERTa for training on the small datasets for A2S can result in model instability, i.e., small modification on the training strategy or the learning rate can produce very different results on the target dataset. The paper suggests to *adapt* the pre-trained transformer weight using a big dataset for Machine Reading comprehension, i.e., Natural Questions, adapted to the task of A2S, before fine-tuning on the small real-world datasets for A2S. The model trained in this way achieved the current state of the art results on standard A2S datasets, i.e., WikiQA and TrecQA.

Min, Seo, and Hajishirzi [83] presents a related technique, where a model initially trained for the task of Machine Reading Comprehension (SQuAD) is adapted to the task of A2S, obtaining consistent improvements.

3.4 State of the art

This section describes the details of two neural network approaches that are heavily used in this thesis, i.e., the RelCNN [111] and the Compare-Aggregate architecture [145].

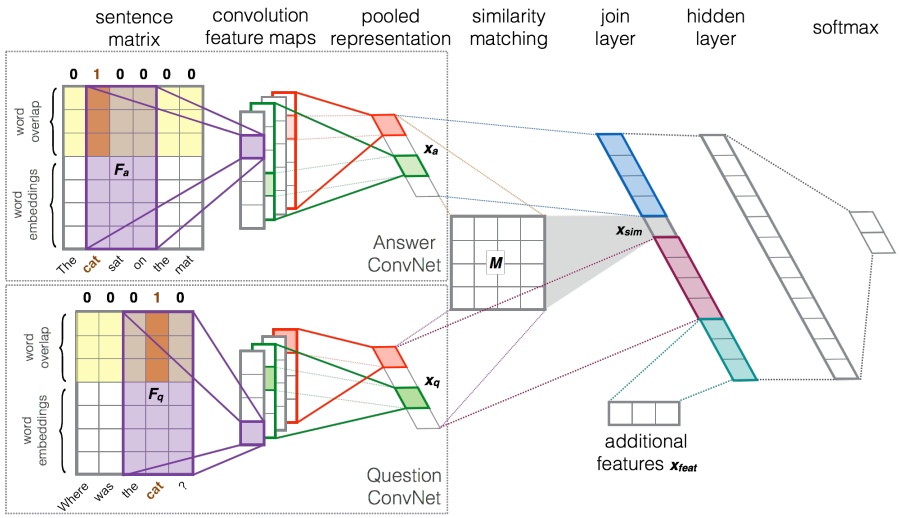

Figure 3.5: The RelCNN architecture. *Image from Severyn and Moschitti [111].*

3.4.1 Relational CNN

The RelCNN architecture in Severyn and Moschitti [111] is one of the first models for A2S that achieved performance comparable to previous feature-based and kernel-based approaches. The model is based on the CNN architecture for sentence encoding, a max-pooling operation to encode the sentence representations into fixed-sized vectors, and an MLP for binary classification.

The model is depicted in Figure 3.5, from left to right the model:

- encodes the words of the question and the answer (the candidate) in word embeddings producing a sentence matrix $\mathbf{S} \in \mathbb{R}^{l \times d_w}$.
- The sentence matrix is augmented with word-overlap embeddings; if a word is present in both the question and the answer candidate, the word-embedding for this word is augmented with a "positive" word-overlap embedding (of size $d_{wo} = 5$), if the word does not overlap it is augmented with a "negative" word-overlap embedding. The weights of these two vectors are updated during training. The concatenation of the word embedding matrix and the word-overlap embedding matrix results in the sentence matrix $\mathbf{S}' \in \mathbb{R}^{l \times d_w + d_{wo}}$. This new sentence matrix encodes infra-sentence word relations.
- A convolution operation with window size equals to 5 is applied on the sentence matrix to encode local word-order information of the sequence. The matrix $\mathbf{C}$ as the output of the convolution operation $C = Conv(S')$ is a matrix representation of the sentence.
- A max-pooling operation is applied on the matrix representation $\mathbf{C} \in \mathbb{R}^{l \times d_h}$ to reduce the sentence representation to a fixed size vector $\mathbf{c} \in \mathbb{R}^{d_h}$ of size d_h.

- The vector $\mathbf{c}_q$ of the question q and the vector $\mathbf{c}_a$ for the answer candidate a are compared using a bilinear similarity matrix $\mathbf{M} \in \mathbb{R}^{d_h \times d_h}$ to be trained, i.e, $x_{sim} = \mathbf{c}_q \mathbf{M} \mathbf{c}_a$.
- the join layer concatenates the question representation $\mathbf{c}_q$ the answer candidate question representation $\mathbf{c}_a$ the similarity score x_{sim} and the feature vector x_{feat}, that can encode external word-overlap count features.
- The join layer is passed to an MLP for scoring and classification. The MLP has a hidden layer of size equal to the size of the join layer.

The network uses word embeddings of size $d_w = 50$ pre-trained using the skip-gram model of the word2vec library on a big dataset, i.e., English Wikipedia and AQUAINT corpora. And a convolutional hidden size $d_h = 100$. Therefore the sentence embeddings $\mathbf{c}_q$ and $\mathbf{c}_a$ are 100-dimensional vectors. The model is trained for a maximum of 25 iterations (epochs) on the training set using Stochastic Gradient Descent (SGD) with the Adadelta update rule [162]. The model is trained with early stopping, stopping training when the development set's model performance does not improve for 5 epochs.

The model is tested on the two popular datasets for A2S, WikiQA, and TrecQA, achieving competitive results, i.e., 69.51 MAP on WikiQA and 76.54 MAP on TrecQA[1].

Additionally, the paper highlights the importance of adding the word overlap features added to the word embeddings, in contrast to the base CNN that do not use lexical matching (relational) information :

Model	MAP	MRR
CNN	66.61	68.51
RelCNN	69.51	71.07

Table 3.1: Comparison on WikiQA between the base CNN model and the model described in this section (RelCNN)

Although the model does not achieve a state of the art results on the benchmark datasets (the model was first published in 2016), it remains competitive as it has a very low number of parameters to be trained.

3.4.2 Compare-Aggregate

The Compare-Aggregate architecture was first presented in Wang and Jiang [145]. This architecture's key innovation is the use of word-level attention to automatically infer relations between the words in the question and the answer.

The encoders of the model are depicted in Figure 3.6. The architecture is structured as follows:

- The model encodes the words of the question and the candidate in word embeddings, producing a sentence matrix $\mathbf{S} \in \mathbb{R}^{l \times d_w}$.

[1]These results are reported in the TrecQA (CLEAN) setting `https://aclweb.org/aclwiki/Question_Answering_(State_of_the_art)`

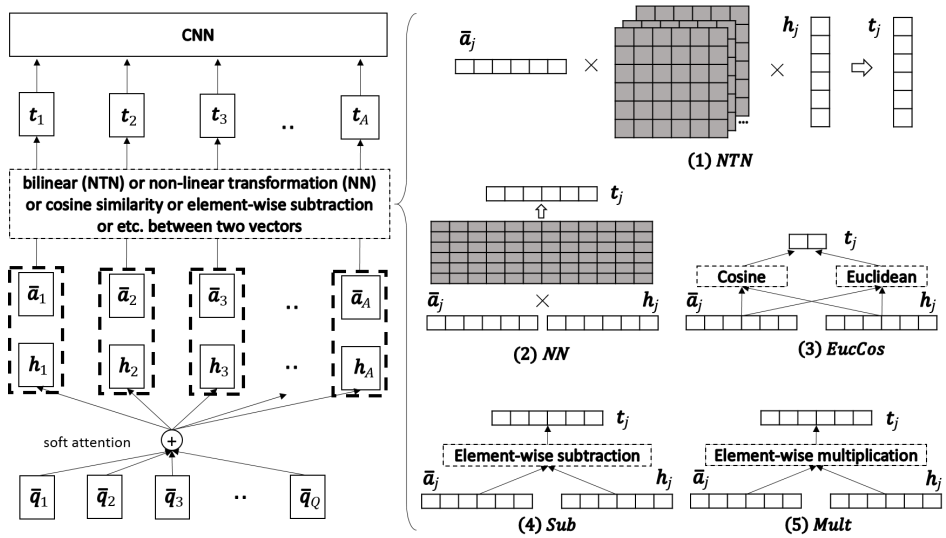

Figure 3.6: The Relational Compare Aggregate architecture. *Image from Wang and Jiang [145].*

- Each word embedding in the sentences $\mathbf{s_i} \in S$ is transformed using a gated non-linear layer, i.e., a non-linear layer with tanh activation is multiplied with the output of a non-linear layer with sigmoid activation.

$$\bar{\mathbf{s}}_i = \sigma(\mathbf{W}_1\mathbf{s}_i + \mathbf{b}_1) \odot tanh(\mathbf{W}_2\mathbf{s}_i + \mathbf{b}_2) \tag{3.13}$$

  where $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ are the weights to be trained, $\odot$ is the element-wise multiplication, and $\bar{\mathbf{s}}_i$ is the transformed word embedding.

- With $\bar{\mathbf{q}}_i \in \bar{\mathbf{Q}}$ and $\bar{\mathbf{a}}_j \in \bar{\mathbf{A}}$ being transformed words embeddings of the question and the answer candidate respectfully, the model performs an attention between each word in the answer and all the words in the question, i.e.,

$$\mathbf{h}_i = Softmax(\bar{\mathbf{a}}_i\bar{\mathbf{Q}})\bar{\mathbf{Q}} \tag{3.14}$$

  In the original formulation [145] the vector $\bar{\mathbf{a}}_i$ is further transformed using a linear layer before using it for attention. However, this layer have been removed in further implementations [10].

- The resulting vector $\mathbf{h}_i$ is a fixed size representation of the answer conditioned on the word embedding of the answer candidate $\bar{\mathbf{a}}_i$.

- The vectors of the question $\bar{\mathbf{a}}_i$ and the vector $\mathbf{h}_i$ are combined as in Figure 3.6. The paper empirically demonstrated that for the task of A2S the simple Hadamard product produces the best results .

- The resulting Sentence matrix is given in input to five independent CNN layers with different window sizes, i.e., $1, 2, 3, 4, 5$, and ReLU activation function.
- A max-pooling operation maps the vector in the output of each convolution to a fixed-size representation. The five resulting vectors are concatenated for classification.
- [145] concatenates these vectors for each answer candidates and uses a multi-layer perception to evaluate and score the answer candidates for the same question altogether. However, in later implementations, it has been shown that this layer introduces noise in the training process, and it has been removed.
- The scores for each answer candidates are concatenated, normalized with a softmax activation, and trained with the KL-divergence loss function on the final task, i.e., a listwise training approach.

Bian et al. [10] extended and modified the approach above, performing the operations as mentioned earlier for the question and the answer independently. Additionally, the authors proposed a more complex attention strategy that used a two-step training process and removed the feed-forward layer on top of the scoring function.

Model	MAP	MRR
Wang and Jiang [145]	74.33	75.45
Bian et al. [10]	75.40	76.40

Table 3.2: Comparison on WikiQA between the two major implementations of the Compare Aggregate architecture

Table 3.2 shows the results obtained by the two major implementations of the Compare-Aggregate framework on the WikiQA dataset.

3.5 Efficiency

In this section, we present an analysis of the efficiency of the major building blocks of the model described above.

The computational complexity of the Convolution operation for the question depends on the following factors: the window size W, the size of the vector as a result of the convolution H, the size of the input in the convolution operation, and the length of the input sequence, i.e., N_q for the question $O(W \times H \times E \times N_q)$. $P = W \times H \times E$ is the number of parameters of the convolution. The computational complexity of the operation is, therefore, $O(P \times N_q)$ Given the fact that the sequence length is much lower than the number of parameters $P >> N_q$ we can say that the computational complexity of the convolution operation greatly depends on the number of parameters of the model P, $O(P)$. Given that the number of parameters is fixed and decided a priory, a simple CNN performs inference in constant time $O(1)$ and is not impacted by the sequence length.

The attention operation on the other end compares each word in the question N_q with all the words in the answer candidate N_a; the attention operation is, therefore, $O(N_q \times N_a)$ – It is important to notice that this complexity is the same for the word overlap features in Severyn and Moschitti [111]. Therefore, the length of the input sequences greatly impacts the overall performance of the model. This behavior impacts both the Compare Aggregate architecture and, in particular, the Transformer. The Transformer architecture performs self-attention on the concatenated sequence of the question and the answer candidate $N = N_q + N_a$. This means that a layer of the Transformer performs four times more operations than the Compare-Aggregate architecture. In the BERT architecture, the attention operation is performed multiple times at each layer (Multi-Head Self-Attention) and for several layers (up to 24), thus greatly impacting the model's performance.

In summary, we have seen that the complexity of a model depends on two factors: the way the alignments between the question and the answer candidates are computed, i.e., the attention or the word-overlap features the number of parameters of the model. Therefore we can approximate the complexity of state-of-the-art models for A2S as $O(P \times N^2)$.

The models described above are very different in both aspects. The RelCNN network has the least parameters $P \sim 50k$, without counting the word-embedding matrix, and the fastest alignment function (the word overlap embeddings). The training process of the RelCNN model on the WikiQA datasets takes ~ 3 seconds on a powerful GPU (Nvidia 1080 Ti) and can train on CPU in 30 seconds. The Compare-Aggregate architecture has two orders of magnitude more parameters $P \sim 1M$ and a slower alignment mechanism, thus taking ~ 20 seconds to train (on GPU).

Although BERT achieves the best performance on the task, it uses $P \sim 340M$ parameters and multiple alignments on the concatenation of question and answer, this results in training times in the order of ~ 20 minutes on the small WikiQA dataset, without counting the time needed for pre-training. Although those numbers are reasonable on small datasets, on bigger datasets, training times become prohibitive, i.e., on the bigger datasets of Natural Questions: RelCNN ~ 30 minutes, Compare-Aggregate ~ 5 hours, and BERT ~ 7 days. These limitations are evident even at test time; thus, having big models with many parameters translated in increased cost for both training and maintaining the model in a production environment.

In this thesis, we focus on building efficient models for A2S capable of achieving competitive performance compared with the state-of-the-art. To achieve this goal, we focus on the following design assumption for our network:

- We want to limit as much as possible the number of parameters of the architecture.
- We want a fast way to capture the alignments between the input examples.

To achieve our goal, we build networks specific to the task, in contrast to the more general BERT architecture. This is achieved by enabling models to capture various features of the A2S, such as the document structure, the syntactic structure of the sentence, or advanced relational information.

Part II

Encoding Local Structure

Chapter 4

Convolutional Neural Networks vs Convolution Kernels

This chapter analyzes the differences between two machine learning methods for A2S: Convolutional Neural Networks (CNNs) and Convolution Tree Kernels (CTKs). These two methods enable automatic feature engineering for syntactic and semantic tasks. Manual feature engineering typically requires a considerable effort to encode rules based on syntactic and semantic features manually. In this perspective, automatizing feature engineering methods are remarkably essential to enable fast prototyping of commercial applications. To the best of our knowledge, two of the most effective methods for engineering features are (i) kernel methods, which naturally map feature vectors or directly objects in richer feature spaces, and more recently (ii) approaches based on deep learning, which is very effective.

Regarding the former, Severyn and Moschitti [107] used CTKs in Support Vector Machines (SVMs) to generate features from a question (q) and their candidate answer candidates (c_i). CTKs enable SVMs to learn in the space of convolutional subtrees of syntactic trees used for representing q and c_i. This, de facto, automatically engineers syntactic features for a complex task. One of their approach's main characteristics was the use of relational links between q and c_i, which merged the two syntactic trees in a relational graph (containing relational features).

Although based on different principles, also CNNs can generate powerful features, e.g., see [54, 58]. CNNs can effectively capture the compositional process of mapping individual words' meanings in a sentence to a continuous representation of the sentence. This way, CNNs can efficiently learn to embed input sentences into a vector space, preserving important syntactic and semantic aspects of the input sentence.

In this chapter, we aim to compare CTKs and CNNs' ability to extract the **local structure** of the question and answer candidate pairs. We first explore CTKs applied to **linguistic structures** – syntactic structures – for automatically learning classification and ranking functions with SVMs. Simultaneously, we experiment with the efficient RelCNN architecture for effectively modeling q and c_i pairs generating relational features. The main building blocks of this approach are two sentence models based on CNNs. These work in parallel, mapping questions

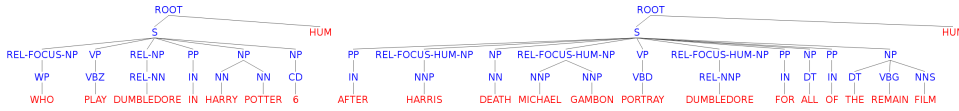

Figure 4.1: Shallow chunk-based tree for the q/c_i pair in the running example.

and answer sentences to fixed-size vectors, which are then used to learn the semantic similarity. The RelCNN architecture captures **relational information** by injecting overlapping words directly into the word embeddings as additional dimensions. The augmented word representation is then passed through the convolutional feature extractors' layers, which encode the relatedness between q and c_i pairs in a more structured manner. Moreover, the embedding dimensions encoding overlapping words are parameters of the network and are tuned during training.

We experiment with two different Open-Domain QA benchmarks for sentence reranking TrecQA [144] and WikiQA [149]. We compare CTKs and CNNs, and then we also combine them. Our CTK-based models achieve competitive results on TrecQA, obtaining an MRR of 85.53 and a MAP of 75.18. On WikiQA, our CNNs performs almost on par with tree kernels, i.e., an MRR of 71.07 vs. 72.51 of CTK. The combination between CTK and CNNs produces a further boost, achieving an MRR of 75.52 and a MAP of 73.99, confirming that the research line of combining these two interesting machine learning methods is very promising.

Additionally, we tested the different approaches to the task of community Question Answering (cQA). More specifically, we participated in the Semeval 2016 Task 3: Community Question Answering competition [89]. The approach was tested on the Subtask A that consists of a question q and a set of comments c_i that are extracted from a forum thread. To this end, we proposed a modified Syntactic Structure for CTK combined with the RelCNN vector representation. The resulting system ConvTK, ranked second at the competition.

4.1 Encoding Local Structure with CTKs

In this work, our approach to learning relations between two texts is to first convert them into a richer structural representation based on their syntactic and semantic structures and then to apply CTKs. To make our approach more effective, we further enriched structures with relational semantics by linking the related constituents with lexical and other semantic links.

4.1.1 Shallow Representations of A2S

In our study, we employ a modified version of the shallow structural representation of question and answer pairs, **CH**, described in [113, 129]. We represent a pair of short texts as two trees with lemmas at the leaf level and their part-of-speech (POS) tags at the preterminal level. Preterminal POS-tags are grouped into chunk nodes, and the chunks are further grouped into sentences. Figure 4.1 provides an example of this structure.

We enrich the above representation with the information about question class and question focus. Questions are classified in terms of their expected answer type. [113] employed coarse-grained classes from [71], namely HUM (person), ENTY (an entity), DESC (description), LOC (location), NUM (number). In this work, we split the NUM class into three sub-categories, DATE, QUANTITY, CURRENCY and train question classifiers as described in [113]. Differently from their work, we add the question class node as the rightmost child of the root node to the question and the answer structures.

We detect question focus using a focus classifier, FCLASS, trained as in [113]. When predicting, they iterate over chunks in the question and pick the one with the highest FCLASS prediction score as a focus even if it is negative. In our work, if FCLASS assigns negative scores to all the question chunks, we consider the first question chunk, which is typically a question word, to be a focus. We mark the focus chunk by prepending the REL-FOCUS tag to its label.

Previous work has shown the importance of encoding information about relatedness between q and c_i into their structural representations. In our work, we employ lexical and question class match.

Lexical match. Lemmas that occur both in q and c_i are marked by prepending the **REL** tag to the labels of the corresponding preterminal nodes and their parents.

Question class match. We detect named entities (NEs) in c_i and mark the NEs of type compatible[1] with the question class by prepending the REL-FOCUS-QC label to the corresponding pre-preterminal in the trees. The QC suffix in the labels is replaced by the question class in the given pair.

For example, in Figure 4.1, the *Dumbledore* lemma occurs in both q and c_i, therefore the respective POS and chunk nodes are marked with **REL**. The named entities, *Harris*, *Michael Gambon* and *Dumbledore* have the type Person compatible with the question class HUM, thus their respective chunk nodes are marked as REL-FOCUS-HUM (overriding the previously inserted REL tag for the *Dumbledore* chunk).

4.1.2 Syntactic Relational Representations for cQA

In our experiments for community Question Answering (cQA), we adjust the structural representation to better model the cQA task, i.e., we enrich the structures with additional relational information and cQA-specific thread knowledge. Additionally, we found that using a constituency structure **CONST** is more effective than the shallow representation **SH**. Moreover, differently from Severyn, Nicosia, and Moschitti [113], we do not remove the punctuation marks from the trees, as, for example, the comments that ask additional questions and thus contain a question mark are unlikely to answer the original question. Another significant

[1]Compatibility is checked using a predefined table, namely Person, Organization→HUM, ENTY; Misc→ENTY; Location→LOC; Date, Time, Number→DATE; Money, Number→CURRENCY; Percentage, Number→QUANTITY

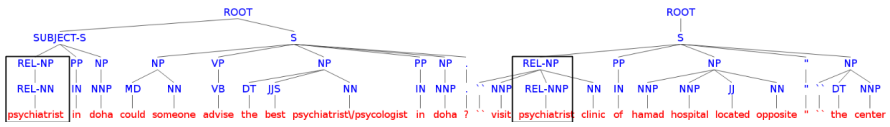

Figure 4.2: **SH** tree for the Q/Comment pair.

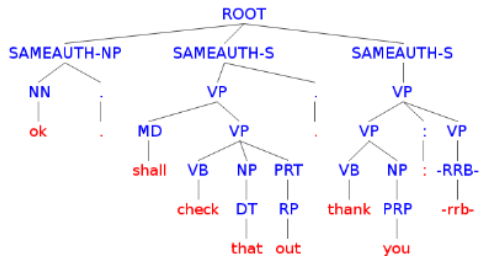

Figure 4.3: **CONST** tree enriched with question author information

difference between the representations for open-domain QA and cQA is that the latter do not benefit from Question type and Question focus information.

In contrast with Open-Domain QA, we encode the following cQA-related information: (i) the question subject is a separate sub-tree under the **SUBJECT-S** root in the question tree; (ii) following the intuition that the question author unlikely will correctly answer his/her question, we add the **SAMEAUTH** label to the sentence nodes in the comment tree. Additionally, when constructing the CONST representation of the question tree, we do not use the question body, i.e., a longer description of the question, but only its subject[2]. Figure 4.1 provides an example of the **REL-** encoding in the SH tree with lemma *psychiatrist* marked with REL- in both trees and question subject separated from its body. Fig. 4.3 illustrates a CONST representation of the author's comment on her question.

4.2 Encoding Semantic Representations with CNNs

The architecture of the RelCNN architecture for matching q and c_i pairs is presented in Fig. 4.4 and detailed in Section 3.4.1. Its main components are: (i) sentence matrices $\mathbf{s}_{c_i} \in \mathbb{R}^{d \times |\mathbf{c}_i|}$ obtained by the concatenation of the word vectors $\mathbf{w}_j \in \mathbb{R}^d$ (with d being the size of the embeddings) of the corresponding words w_j from the input sentences (Q and c_i) $\mathbf{c}_i$; (ii) a convolutional sentence encoder $f : \mathbb{R}^{d \times |\mathbf{c}_i|} \rightarrow \mathbb{R}^m$ that maps the sentence matrix of an input sentence $\mathbf{s}_i$ to a fixed-size vector representations $\mathbf{x}_{s_i}$ of size m; (iii) a layer for computing the similarity between the obtained intermediate vector representations of the input sentences, using a similarity matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ – an intermediate vector representation $\mathbf{x}_{s_1}$ of a sentence $\mathbf{s}_1$ is projected to a $\tilde{\mathbf{x}}_{s_1} = \mathbf{x}_{s_1}\mathbf{M}$, which is then matched with $\mathbf{x}_{s_2}$ [14], i.e., by computing a dot-product $\tilde{\mathbf{x}}_{s_1}\mathbf{x}_{s_2}$, thus resulting in a single similarity score x_{sim}; (iv) a set of

[2]Our preliminary experiments showed that adding the question body to the CONST representation does not affect the performance.

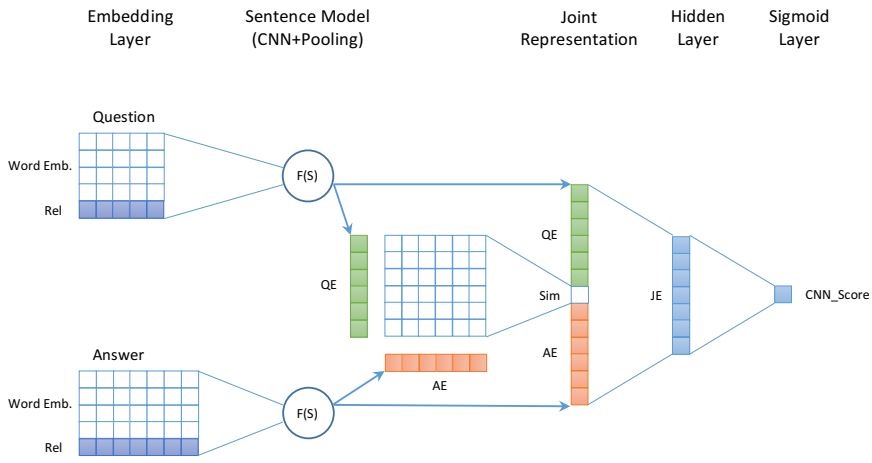

Figure 4.4: CNN for computing the similarity between question and answer.

fully-connected hidden layers that model the similarity between sentences using their vector representations produced by the sentence model (also integrating the single similarity score from the previous layer); and (v) a *sigmoid* layer that outputs probability scores reflecting how well the q/c_i pair match with each other.

To capture relational information, we follow the approach of Severyn and Moschitti [111], in the input sentence, we associate an additional *word overlap* indicator feature $o \in \{0, 1\}$ with each word w, where 1 corresponds to words that overlap in a given pair and 0 otherwise. To decide if the words overlap, we perform string matching. This small feature vector plays the role of the REL tag added to the CTK structures.

For the task of community Question Answering, additional features are encoded in the joint representation layer. In particular, we encode the $SAMEAUTH$ feature that indicates whether the author of the comment c_i is the same as the author of the question q and $SAMEAUTH^+$ indicates whether the comment c_i is followed by a comment c_{i+1} from the same author of the question.

4.2.1 Representation Layers

It should be noted that NNs non-linearly transforms the input at each layer. For instance, the output of the convolutional and pooling operation $f(s_i)$ is a fixed-size representation of the input sentence s_i. In the remainder of the chapter, we will refer to these vector representations for the question and the answer passage as the question embedding (QE) and the answer embedding (AE), respectively. Similarly, the output of the network's penultimate layer (the hidden layer whose output is fed to the final classification layer) is a compact representation of the input Question and Answer pair, which we call Joint Embedding (JE).

It should be noted that the vectors QE, AE, and JE contain relevant information for classi-

fication. Therefore, they embed the q/c_i pair into fixed-size representations.

4.3 Experiments on Open Domain QA

In these experiments, we compare the accuracy impact of two main methods for automatic feature engineering, i.e., CTKs and CNNs, for relational learning, using two different answer sentence selection datasets, WikiQA and TrecQA. We propose several strategies to combine CNNs with CTKs, and we show that the two approaches are complementary as their joint use significantly boost both models.

4.3.1 Experimental Setup

We utilized two datasets for testing our models:

TrecQA. This is the factoid open-domain TrecQA corpus prepared by [144]. The training data was assembled from the 1,229 TREC8-12 questions. The training questions' answers were automatically marked in sentences by applying regular expressions; therefore, the dataset can be noisy.

The test data contains 68 questions, whose answers were manually annotated. We used ten answer passages for each question for training our classifiers and all the answer passages available for each question for testing.

WikiQA. TrecQA is a small dataset with an even smaller test set, making the system evaluation rather unstable, i.e., a small difference in parameters and models can produce very different results. Moreover, as pointed by [155], it has a significant lexical overlap between questions and answers candidates. Therefore simple lexical match models may likely outperform more elaborate methods if trained and tested on it. WikiQA dataset [149] is a larger dataset, created for open-domain QA, which overcomes these problems. Its questions were sampled from the Bing query logs, and candidate answers were extracted from the associated Wikipedia pages' summary paragraphs. The train, test, and development set contain 2,118, 633, and 296 questions, respectively. There is no correct answer sentence for 1,245 training, 170 development, and 390 test questions. Consistently with [157], we remove the questions without answers for our evaluations.

Preprocessing. We used the Illinois chunker [95], question class, and focus classifiers trained as in [108] and the Stanford CoreNLP [77] toolkit for the needed preprocessing.

CTKs. We used SVM-light-TK[3] to train our models. The toolkit enables the use of structural kernels [84] in SVM-light [53]. We applied (i) the partial tree kernel (PTK) with its default parameters to all our structures and (ii) the polynomial kernel of degree 3 on all feature vectors we generate.

Metaclassifier. We used the *scikit*[4] logistic regression classifier implementation to train the meta-classifier on the outputs of CTKs and CNNs.

[3] http://disi.unitn.it/moschitti/Tree-Kernel.htm
[4] http://scikit-learn.org/stable/index.html

	MRR	MAP	P@1
State of the art			
CNN_c [149]	66.52	65.20	n/a
ABCNN [157]	71.27	69.14	n/a
$LSTM_{a,c}$ [78]	70.41	68.55	n/a
$NASM_c$ [78]	70.69	68.86	n/a
Our Individual Models			
RelCNN	71.07	69.51	57.20
CH_{coarse}	71.63	70.45	56.79
CH	72.30	71.25	58.44
V_{AE+QE}	68.29	67.24	55.56
V_{JE}	67.07	65.76	52.26
Our Model Combinations			
CH+V_{AE+QE}	72.51	71.29	59.26
CH+V_{JE}	73.18	71.56	60.49
CH'+V'_{AE+QE}	75.88	74.17	64.61
CH'+V'_{JE}	75.52	73.99	63.79
Meta: CH, V_{JE}, RelCNN	75.28	73.69	62.96
Meta: CH, V_{JE}	75.08	73.64	62.55
Meta: CH+V_{JE}, RelCNN	73.94	72.25	61.73

Table 4.1: Performance on the WikiQA dataset

CNNs. We pre-initialize the word embeddings by running the `word2vec` tool [82] on the English Wikipedia dump and the ACQUAINT corpus as in [109]. We opt for a skip-gram model with window size five and filtering words with a frequency of less than 5. The dimensionality of the embeddings is set to 50. The input sentences are mapped to using a convolution with a hidden size of 100 and a window size of 5 and a max-pooling operation. We use a single non-linear hidden layer (with hyperbolic tangent activation, Tanh), whose size is equal to the previous layer's size. The network is trained using SGD with shuffled mini-batches using the Adam update rule [61]. The batch size is set to 100 examples. The network is trained for a fixed number of epochs (i.e., 3) for all the experiments. We decided to avoid using early stopping not to overfit the development set and have a fair comparison with the CTKs models.

QA metrics. We used standard QA metrics: Precision at rank 1 (P@1), i.e., the percentage of questions with a correct answer ranked in the first position, the Mean Reciprocal Rank (MRR) and the Mean Average Precision (MAP).

4.3.2 Experiments on WikiQA

Previous work Table 4.1 reports the results obtained on the WikiQA test set by state-of-the-art systems (lines 1-4) and our models when removing the questions with no correct answers (this to be aligned with previous work). More in detail:
CNN_c is the Convolutional Neural Network with word count,

	TRAIN50			DEV		
	MRR	MAP	P@1	MRR	MAP	P@1
CH	69.97	68.77	55.14	67.23	65.93	51.44
V_{AE+QE}	68.7	67.18	54.32	68.14	66.46	54.73
V_{JE}	70.43	68.67	57.61	68.90	67.14	55.56
Model Combinations						
CH+V_{AE+QE}	74.4	72.63	62.55	70.01	68.60	57.61
CH+V_{JE}	73.53	71.69	60.49	70.10	68.55	58.44
Metaclassifiers:						
CH, V_{JE}, RelCNN	74.01	72.31	62.14	n/a	n/a	n/a
CH, V_{JE}	73.95	72.15	62.14	n/a	n/a	n/a
CH+V_{JE}, RelCNN	73.43	71.58	60.49	n/a	n/a	n/a

Table 4.2: Performance on the WikiQA using the development set or half of the training set for training

	TRAIN			TRAIN50		
	MRR	MAP	P@1	MRR	MAP	P@1
CH	74.87	74.17	63.49	71.31	70.45	57.94
V_{AE+QE}	70.32	69.75	56.35	71.06	70.33	57.14
V_{JE}	69.86	69.24	55.56	71.11	70.43	57.14
CH+V_{AE+QE}	71.29	70.79	57.94	72.62	72.18	59.52
CH+V_{JE}	71.36	70.81	57.94	71.96	71.55	59.52
CH'+V'_{AE+QE}	76.66	75.50	66.67	75.23	74.54	64.29

Table 4.3: Performance on the WikiQA on the development set

ABCNN is the Attention-Based CNN,
LSTM$_{a,c}$ is the long short-term memory network with attention and word count, and
NASM$_c$ is the neural answer selection model with word count.
RelCNN is the relational CNN described in Section 4.2.
CH is a tree kernel-based SVM reranker trained on the shallow pos-chunk tree representations of question and answer sentences (Sec. 4.1.1), where the subscript $_{\text{coarse}}$ refers to the model with the coarse-grained question classes as in [129].
V is a polynomial SVM reranker, where the subscripts AE, QE, JE indicate the use of the answer, question, or joint embeddings (see Sec. 4.2.1) as the feature vector of SVM and $+$ means that two embeddings were concatenated into a single vector.

The results show that our RelCNN model performs comparably to ABCNN [157], which is the most recent and accurate NN model and to CH$_{\text{coarse}}$. The performance drops when the embeddings AE, QE, and JE are used in a polynomial SVM reranker. In contrast, CH (using our tree structure enriched with fine-grained categories) outperforms all the models, showing the importance of syntactic relational information for the answer sentence selection task.

4.3.2.1 Combining CNN with CTK on WikiQA

We experiment with two ways of combining CTK with RelCNN: (i) at the kernel level, i.e., summing tree kernels with the polynomial kernel over different embeddings, i.e., CH+V, and (ii) using the predictions of SVM and RelCNN models (computed on the development set) as features to train logistic regression meta-classifiers (again only on the development set). These are reported in the last three lines of the table, where the name of the classifiers participating with their outputs is illustrated as a comma-separated list. The results are very interesting as all kinds of combinations largely outperform the state of the art, e.g., by around 3 points in terms of MRR, 2 points in terms of MAP, and 5 points in terms of P@1 in contrast with the strongest standalone system, CH. Directly using the predictions of the RelCNN as features in the meta-classifier does not impact the overall performance. It should be noted that the meta-classifier could only be trained on the development data to avoid predictions biased by the training data.

4.3.2.2 Using less training data

Since we train the weights of RelCNN on the training set of WikiQA, to obtain the embeddings minimizing the loss function, we risk to have overfitted, i.e., "biased", JE, AE, and QE on the questions and answers of the training set. Therefore, we conducted another set of experiments to study this case. We randomly split the training set into two equal subsets. On one of them, we train the RelCNN model, whereas, in the other subset (referred to as **TRAIN50**), we produce the embeddings of questions and answers.

Table 4.2 reports the results on the WikiQA test set, which we obtained when training SVM on TRAIN50 and the development set, **DEV**. We trained the meta-classifier on the predictions of the standalone models on DEV. Consistently with the previous results, we obtain the best performance combining the RelCNN embeddings with CTK. Even when we train on the 50% of the training data only, we still outperform state of the art, and our best model CH+V_{JE} performs only around 2 points lower in terms of MRR, MAP, and P@1 than when training on the full training set.

Finally, Table 4.3 reports the performance of our models when tested on the development set and demonstrates that the improvement obtained when combining CTK and RelCNN embeddings also hold on it. Note that we did not use the development set for any parameter tuning, and we train all the models with the default parameters.

4.3.3 Experiments on TrecQA dataset

TrecQA corpus has been used for evaluation in several works starting from 2007. Table 4.4 reports our as well as some state-of-the-art system results on TrecQA. It should be noted that to be consistent with the previous work; we evaluated our models in the same setting as [144, 156], i.e., we (i) remove the questions having only correct or only incorrect answer sentence candidates and (ii) used the same evaluation script and the gold judgment file as they used. As pointed out by Footnote 7 in [154], the evaluation script always considers four questions to be answered incorrectly, thus penalizing the overall system score.

Models	MRR	MAP
State of the art		
Wang, Smith, and Mitamura [144]	68.52	60.29
Heilman and Smith [43]	69.17	60.91
Wang and Manning [142]	69.51	59.51
Yao et al. [152]	74.77	63.07
Severyn and Moschitti [108]	73.58	67.81
Yih et al. [156]	77.00	70.92
Yu et al. [160]	78.64	71.13
Wang and Ittycheriah [147]	77.40	70.63
Tymoshenko and Moschitti [129]	82.29	73.34
Yin et al. [157]	76.33	69.51
Miao, Yu, and Blunsom [78]	81.17	73.39
Individual Models		
RelCNN	77.93	71.09
V_{AE+QE}	79.32	73.37
V_{JE}	77.24	71.34
CH	85.53	75.18
Model Combinations		
CH+V_{JE}	79.75	74.29
CH+V_{AE+QE}	79.74	75.06
Meta: CH, V_{AE+QE}, RelCNN	81.67	75.77
Model Combinations using simpler CH		
CH$_{smpl}$	78.66	71.18
CH$_{smpl}$+V_{AE+QE}	80.19	75.01
CH$_{smpl}$+V_{JE}	80.42	74.16

Table 4.4: Results on the TrecQA, answer selection task.

We note that our models, i.e., RelCNN, V_{JE}, V_{AE+QE}, again align with state of the art. In contrast, our CTK using CH largely outperforms all previous work, e.g., 7.6 points more than RelCNN in terms of MRR. Considering that CH's evaluation with a script that does not penalize systems would show real MRR and MAP of 90.56 and 80.08, respectively, there is little room for improvement with combinations. Indeed, the table shows no improvement in model combinations over CH.

Therefore, we trained a simplified version of CH CH$_{smpl}$, which employs shallow chunk-based representations without the question focus or question class information, i.e., only using the necessary relational information represented by the lexical match **REL** tags. CH$_{smpl}$ performs comparably to RelCNN, and the combination with embeddings produced by RelCNN, i.e., CH$_{smpl}$+V_{AE+QE}, outperforms both CH$_{smpl}$ and RelCNN.

4.4 Experiments on Community Question Answering

In these experiments, we compare CTKs and CNNs, also combining them with traditional feature vector representations on the Semeval 2016 cQA dataset[89].

4.4.1 Experimental setup

Experimental dataset. We use the Subtask A *Question-Comment Similarity* subset of the Semeval-2016 Task 3 English cQA dataset[5]. This enables a direct comparison with results of the challenge. The dataset of questions and comments was extracted from the *Qatar Living forum*[6]. For each question, the first ten comments were collected and manually annotated as good, i.e., answering the question, and potentially useful or bad. The Potentially useful class was relabeled as bad during the challenge evaluation.

The train, development, and test sets contain 1790, 244, and 327 questions.

QA metrics. We report our results in terms of Mean Average Precision (MAP)[7] and Mean Reciprocal Rank (MRR).

CTKs. We trained our models with SVM-light-TK[8]. It enables the use of CTKs [84] in SVM-light [53]. We used the partial and the subset tree kernels (PTK and STK) with their default parameters and the polynomial kernel (P) of degree 3 on all feature vectors.

Preprocessing We truncate all the comments to 2000 symbols and all the sentences to 70 words. When generating the structural representations described in Section 4.1, we used the first three sentences of a question or a comment. We used the Illinois chunker [95] and the Stanford CoreNLP [77] toolkit for the needed preprocessing.

Signatures. If a specific user always signs their posts with the same phrase, e.g., "*The tough gets going.*.", we consider this phrase irrelevant and remove it. More specifically, we delete all the strings with a length exceeding 20 characters that occur at the end of more than one comment by the same user.

cQA-relevant features vectors (QF). We used the thread-level features by one of the top-performing SemEval-2015 systems [91]. Among others, they include the features encoding whether the comment (c) is authored by the question (q) author, whether c contains a question, an acknowledgment word or a URL, and other intuitions.

4.4.2 Neural Network experiments

We pre-initialize the word embeddings with the skip-gram embedding of dimensionality 50 trained on the English Wikipedia dump [82]. The input sentences are encoded with fixed-sized vectors using a convolutional operation of size five and a k-max pooling operation with $k = 1$. We use a single non-linear hidden layer (with hyperbolic tangent activation, Tanh),

[5] http://alt.qcri.org/semeval2016/task3/index.php?id=description-of-tasks
[6] http://www.qatarliving.com/forum
[7] the official Subtask A metric
[8] http://disi.unitn.it/moschitti/Tree-Kernel.htm

Encoders	MAP	MRR	Features	MAP	MRR
W2V	0.6284	68.53	CNN	0.6496	71.26
LSTM	0.6477	**71.78**	+$overlap$	0.6648	73.46
CNN	**0.6496**	71.26	+SAMEAUTH	0.6684	72.56
			+SAMEAUTH$^+$	**0.6741**	**73.64**

Table 4.5: NNs results on the development dataset

whose size is equal to the sentence embeddings' size, i.e., 200. The network is trained using SGD with shuffled mini-batches using the Rmsprop update rule. The model is trained until the validation loss stops improving.

For computational reasons, we decided to limit the size of the input sentences to 100 words. This did not degrade the observed performance. To improve learning generalization and avoid co-adaptation of features (which negatively impact the sentence embeddings), we opted for adding dropout [118] between the layers of the network. Table 8.2 reports the results of our NNs on the development set. The first sentence model is a word embedding averaging model (W2V), where the sentence embeddings are calculated by averaging the word vectors. Then, two strong baseline sentence models are presented, a Long-Short Term Memory (LSTM) sentence model [45] and our vanilla CNN, presented in Section 4.2. Our CNN, without any additional features, outperforms W2V, obtaining similar accuracy than LSTM. Thus, we decided to use the convolutional sentence model mainly for computational reasons. The second part of the table shows CNN incrementally enhanced by the $overlap$, SAMEAUTH, and SAMEAUTH$^+$ features (see Sec. **??**), further improving its performance.

4.4.3 Results

Table 4.6 reports the performance of our systems and compares it to the top systems in the Semeval-2016 Subtask A competition[9]. The participants were allowed to submit one primary and two contrastive runs. The teams were ranked according to the MAP score obtained by their primary system (the official rank is reported in parentheses). The remainder of the table describes our systems' performance on the development (DEV) and the test (TEST) sets. We did not use DEV to tune any parameters.

The *Kernel* column reports the name of the kernel used in SVMs.

V_{QF} is an SVM using a polynomial kernel over the QF feature vector. CNN is the convolution neural network described in Sec. 4.2. CTK_{SH} and CTK_C are the CTK-based SVMs trained on the SH and CONST representations described in Sec. 4.1, respectively. CTK+V denotes the composite kernel that is a sum of two kernels.

Notably, both CTK models that encode only one cQA task-specific intuition (i.e., whether the question and the comment have the same author) are only slightly outperformed by CNNs and achieve a MAP only one point lower than ConvKN-primary, the second best-ranking

[9] http://alt.qcri.org/semeval2016/task3/data/uploads/semeval2016_task3_submissions_and_scores.zip

Models	Kernel	DEV		TEST	
		MAP	MRR	MAP	MRR
1. Baseline models					
Kelp-primary [34] (#1)	n/a	n/a	n/a	79.19	86.42
ConvKN-contrastive1 [6]	n/a	n/a	n/a	78.71	86.15
SUper team-contrastive1 [79]	n/a	n/a	n/a	77.68	84.76
ConvKN-primary [6] (#2)	n/a	n/a	n/a	77.66	84.93
2. CNN and CTK models					
V_{QF}	P	63.45	70.51	73.50	82.98
RelCNN	n/a	67.41	73.64	77.13	83.85
CTK_{SH}	PTK	64.10	71.97	76.67	83.53
CTK_C	STK	65.30	73.24	75.42	82.35
CTK_C	PTK	63.82	70.53	76.39	82.94
CTK_{SH}+V_{QF}	PTK, P	68.45	74.49	**78.80**	86.16
CTK_C+V_{QF}	STK, P	67.26	74.07	**78.78**	86.26
3. Combining CTK and CNN models					
$V_{QE\|CE}$	P	65.63	72.69	75.15	82.37
$V_{QE\|CE\|QF}$	P	**68.17**	75.32	**77.22**	83.98
CTK_C+$V_{QE\|CE}$	STK,P	66.71	**75.18**	**76.25**	83.33
CTK_C+$V_{QE\|CE\|QF}$	STK,P	**68.92**	**76.61**	**77.25**	**84.16**

Table 4.6: Performance of CTK, CNN and QF models on the development (DEV) and test (TEST)

system in official ranking of the competition.

Additionally, we observe that CTK_{SH} with the PTK and CTK_C with STK obtain almost the same performance. This is important, as PTK generates a richer feature space, but STK has a lower computational complexity. Training an SVM with PTK and STK on CTK_C took 4213.93 and 1006.58 cpu-seconds. Next, the CTK and V combinations, CTK_C+V_{QF} (STK) and CTK_{SH}+ V_{QF} (PTK), obtained the MAPs of 78.78 and 78.80 on TEST, respectively. They would have ranked second in the competition.

Finally, since both CNNs and CTKs achieve the competitive performances on cQA, we combined the two approaches using the question, comment, and joint embeddings learned by CNNs as feature vectors in V and combined them with QF and CTK_C. We have experimented with all possible combinations: Sec. 3 of Table 4.6 lists the best ones. The subscripts QE and CE indicate the use of the question and comment embeddings (Sec. 4.2) as the feature vector, whereas | denotes that the respective vectors were concatenated into a single vector.

None of the combinations improved over the CTKs models with QF features; however, an interesting finding is that, in general, V systems, which use only embeddings as features, outperform V_{QF}. Concatenating the QE, CE, and QF into a single feature vector results in further improvement, e.g., $V_{QE|CE|QF}$ achieves a MAP of 77.22 on the TEST, which is only 0.44 points lower than the MAP of the **#2** ranked system. CTK_C+$V_{QE|CE}$, a combination of QE and CE with CTK_C (without QF) slightly improves over CTK_C (using STK) on both

DEV and TEST sets. However, the CTK_C+$V_{AE|CE|QF}$, a combination of CTKs, CNN, and QF, scores lower than $CTK_C + V_{QF}$, thus additional investigation is needed to understand how to combine CTK and CNN successfully.

4.5 Summary

In this chapter, we compare two machine learning methods, namely CTKs and CNNs, for A2S. The two approaches are capable of automatically extract **local features** from question-answer pairs. In particular, CTKs take as input syntactic structures for both the question and an answer candidate and extract relevant features to solve the task. Conversely, CNNs are a set of efficient architectures that compositionally maps the semantic of individual words in a sentence to a fixed size continuous representation.

In order to have a meaningful comparison with previous work, we have set the best configuration for CTK by defining and implementing innovative linguistic structures enriched with semantic information from statistical classifiers (i.e., question and focus classifiers). At the same time, we have developed a fast and accurate CNN architecture that can embed relational information in their representations. The proposed model is inspired by the work of Severyn and Moschitti [111], which was the best available neural architecture when we performed this empirical study.

We tested our models for A2S on two benchmarks, WikiQA and TrecQA. Thus, they are directly comparable with many systems from previous work. The results show that CTKs outperform our CNNs but uses more structural information, e.g., on TrecQA, CTKs obtain an MRR, and MAP of 85.53 and 75.18 vs. 77.93 and 71.09 of the RelCNN. On WikiQA, CNNs combined with tree kernels achieve an MRR of 75.88 and a MAP of 74.17; this result is still unmatched by any model that does not make use of attention mechanism or large pretrained transformer architectures.

To further validate the proposed approach, we participated in the Semeval 2016 community Question Answering competition. To this end, we propose a novel linguistic structure specific to the cQA task. The combination of the RelCNN architecture, together with CTK, outperformed most of the proposed models to achieve second place at the competition, with a MAP of 77.66.

The competitive results obtained in both tasks by the combined model suggest that CTK and CNNs exploit different types of local structures. This indicates that CTKs require the latent semantic information of the word embeddings, and most importantly, for the focus of this thesis, CNNs benefits from the syntactic structure.

Chapter 5

Injecting Syntactic Information in Neural Networks

In the previous chapter, we explored combining Tree Kernels (CTKs) based approaches with Convolutional Neural Networks (CNNs). The results suggest that the two approaches learn very different representations and that combining the two approaches leads to an increase in the overall performance. However, the proposed approach can be problematic in a production environment as it relies on two different classifiers, i.e., the CNN produces the representations that the CTK uses for classification. This is a shame as NNs are very flexible in general and enable an easy system deployment in real applications, while TK models require syntactic parsing and longer testing time.

This chapter proposes an approach that aims to inject syntactic information in NNs, still keeping them simple. It consists of the following steps: train a TK-based model on a few thousands of training examples; apply such classifier to a much broader set of unlabeled training examples to generate automatic annotation; pre-train NNs on the automatic data; and fine-tune NNs on the smaller GS data.

We test this novel technique's capabilities on the Question Question similarity (Task B) task of community Question Answering (cQA). The task consists of selecting the most similar questions from a cQA website regarding a new question. The task was largely explored in the Semeval 2016 competition [89] and on the Quora dataset[1]. An interesting outcome of the SemEval challenge was that syntactic information is essential to achieve high accuracy in question reranking tasks. Indeed, the top-systems were built using Support Vector Machines (SVMs) trained with Tree Kernels (TKs), which were applied to a syntactic representation of question text [6, 34, 89].

Our experiments on two different datasets, i.e., Quora and Qatar Living (QL) from SemEval, show that when NNs are pre-trained on the predicted data, they achieve accuracy higher than the one of TK models, and NNs can be further boosted by fine-tuning them on the available GS data. This suggests that the TK properties are captured by NNs, which can exploit syntactic information even more effectively, thanks to their well-known generalization ability.

[1] https://www.kaggle.com/c/quora-question-pairs

In contrast to other semi-supervised approaches, e.g., self-training, we show that our approach's improvement is obtained only when a very different classifier, i.e., TK-based, is used to label a large portion of the data. Indeed, using the same NNs in a self-training fashion (or another NN in a co-training approach) to label the semi-supervised data does not provide any improvement. Similarly, when SVMs using standard similarity lexical features are applied to label data, no improvement is observed in NNs.

One evident consideration is the fact that TKs-based models mainly exploit syntactic information to classify data. Although assessing that NNs specifically learn such syntax should require further investigation, our results show that only the transfer from TKs produces improvement: this is significant evidence that makes it worth to investigate the central claim of this chapter further. In any case, our approach increases the accuracy of NNs, when small datasets are available to learn high-level semantic task such as question similarity. It consists of using more massive syntactic/semantic models, e.g., based on TKs, to produce training data; and exploit the latter to learn a neural model, which can then be fine-tuned on the small available GS data.

5.0.1 Question Matching as Short Text Ranking

Question similarity in forums can be set in different ways, e.g., detecting if two questions are semantically similar or ranking a set of retrieved questions regarding their similarity with the original question. We describe the two methods below:

The Quora task regards detecting if two questions are duplicate or not, or, in other words, if they have the same intent. The associated dataset [146] contains over $404,348$ pairs of questions, posted by users on the Quora website, labeled as the same pair. For example, *How do you start a bakery?* and *How can one start a bakery business?* are duplicated while *What are natural numbers?* and *What is a least natural number?* are not. The ground-truth labels contain some amount of noise.

In the QL task at SemEval-2016 [89] users were provided with a new (original) question q_o and a set of related questions $(q_1, q_2, ...q_n)$ from the QL forum[2] retrieved by a search engine, i.e., Google. The goal is to rank question candidates, q_i, by their similarity with respect to q_o. q_i were manually annotated as *PerfectMatch*, *Relevant* or *Irrelevant*, depending on their similarity with q_o. *PerfectMatch* and *Relevant* are considered as relevant. A question is composed of a subject, a body and a unique identifier.

5.0.2 Support Vector machines for cQA

A top-performing model in the SemEval challenge is built with SVMs, which learn a classification function, $f : Q \times Q \rightarrow \{0, 1\}$, on the relevant vs. irrelevant questions belonging to the question set, Q. The classifier score is used to rerank a set of candidate questions q_i provided in the dataset with respect to an original question q_o. Three main representations

[2] http://www.qatarliving.com/forum

were proposed: vectors of similarity feature derived between two questions; a TK function applied to the syntactic structure of question pairs; or , a combination of both.

Feature Vectors (FV) are built for question pairs, (q_1, q_2), using a set of *text similarity* features that capture the relations between two questions. More specifically, we compute 20 similarities $sim(q_1, q_2)$ using word n-grams ($n = [1, \ldots, 4]$), after stopword removal, greedy string tiling [148], longest common subsequences [2], Jaccard coefficient [50], word containment [76], and cosine similarity.

Tree Kernels (TKs) measure the similarity between the syntactic structures of two questions. Following Barrón-Cedeño et al. [6], we build two macro-trees, one for each question in the pair, containing the syntactic trees of the sentences composing a question. In addition, we link two macro-trees by connecting the phrases, e.g., NP, VP, PP, etc., when there is a lexical match between the phrases of two questions. We apply the following kernel to two pairs of question trees: $K(\langle q_1, q_2\rangle, \langle q'_1, q'_2\rangle) = TK(t(q_1, q_2), t(q'_1, q'_2))+TK(t(q_2, q_1), t(q'_2, q'_1))$, where $t(x, y)$ extracts the syntactic tree from the text x, enriching it with relational tags (REL) derived by matching the lexical between x and y.

5.1 Injecting Structures in NNs

We inject TK knowledge in two well-known and state-of-the-art networks for question similarity, enriching them with relational information.

5.1.1 NNs for question similarity

We implemented the RelCNN model proposed by Severyn and Moschitti [111]. This learns f, using two separate sentence encoders $f_{q_1} : Q \to \mathbb{R}^n$ and $f_{q_2} : Q \to \mathbb{R}^n$, which map each question into a fixed size dense vector of dimension n. The resulting vectors are concatenated and passed to a Multi-Layer Perceptron that performs the final classification. Each question is encoded into a fixed-size vector using an embedding layer, a convolution operation, and a global max pooling function. The embedding layer transforms the input question, i.e., a sequence of token, $X_q = [x_{q_1}, ..., x_{q_i}, ..., x_{q_n}]$, into a sentence matrix, $S_q \in R^{m \times n}$, by concatenating the word embeddings w_i corresponding to the tokens x_{q_i} in the input sentence.

[111] showed that relational information encoded in terms of overlapping words between two pairs of text could positively improve accuracy. Thus, we mark each word with a binary feature for both networks above, indicating if a word from a question appears in the other pair question. This feature is encoded with a fixed size vector (in the same way it is done for words).

Additionally, we implemented a Bidirectional (BiLSTM) variant, using the standard LSTM by Hochreiter and Schmidhuber [45]. An LSTM iterates over the sentence one word at the time by creating a new word representation h_i by composing the representation of the previews word and the current word vector $h_i = LSTM(w_i, h_{i-1})$. A BiLSTM iterate over the sentence in both directions, and the final representation is a concatenation of the hidden

representations, h_N, obtained after processing the whole sentence. We apply two-sentence models (with different weights), one for each question, then we concatenate the two fixed-size representations and feed them to a Multi-Layer Perceptron.

5.1.2 Learning NNs with structure

To inject structured information in the network, we use a weak supervision technique: an SVM with TK is trained on the GS data; this model classifies an additional unlabelled dataset, creating automatically; and a neural network is trained on the latter data.

The pre-trained network can be fine-tuned on the GS data, using a lower learning rate of γ. This prevents catastrophic forgetting [38], which may occur with a larger learning rate.

5.2 Experiments

We experiment with two datasets comparing models trained on gold and automatic data and their combination, before and after fine-tuning.

5.2.1 Data

Quora dataset contains $384,358$ pairs in the training set and $10,000$ pairs both in the dev. and test sets. The latter two contain the same number of positive and negative examples.

QL dataset contains $3,869$ question pairs divided in $2,669$, 500 and 700 pairs in the training, dev. and test sets. We created 93k[3] unlabelled pairs from the QL dump, retrieving ten candidates with Lucene for $9,300$ query questions.

5.2.2 NN setup

We pre-initialize our word embeddings with skip-gram embeddings of dimensionality 50 jointly trained on the English Wikipedia dump [82], and the AQCUAINT corpus[4]. The input sentences are encoded with fixed-sized vectors using a CNN with the following parameters: a window of size 5, an output of 100 dimensions, followed by a global max pooling. We use a single non-linear hidden layer, whose size is equal to the sentence embeddings' size, i.e., 100. The word overlap embeddings are set to 5 dimensions. The activation function for both convolution and hidden layers is ReLU. During training, the model optimizes the binary cross-entropy loss. We used SGD with Adam update rule, setting the learning rate to γ to 10^{-4} and 10^{-5} for the pre-training and fine-tuning phases.

5.2.3 Results on Quora

Table 5.1 reports our different models, FV, TK, CNN, and LSTM described in the previous section, where the suffix, -10k or -5k, indicates the amount of GS data used to train them,

[3]Note that we will release the 400k automatically labeled pairs from Quora as well as the new 93k pairs of QL along with their automatic labels for research purposes.

[4]Embeddings are available in the repository: `https://github.com/aseveryn/deep-qa`

Model	Automatic data	GS data	DEV	TEST
FV-10k	–	10k	0.7046	0.7023
TK-10k	–	10k	0.7405	0.7337
CNN-10k	–	10k	0.7646	0.7569
LSTM-10k	–	10k	0.7521	0.7450
CNN(CNN-10k)	50k	–	0.7666	0.7619
CNN(CNN-10k)*	50k	10k	0.7601	0.7598
CNN(FV-10k)	50k	–	0.6960	0.6931
CNN(FV-10k)*	50k	10k	0.7681	0.7565
CNN(TK-10k)	50k	–	0.7446	0.7370
CNN(TK-10k)*	50k	10k	0.7748	0.7652
LSTM(TK-10k)	50k	–	0.7478	0.7371
LSTM(TK-10k)*	50k	10k	0.7706	0.7505
TK-5k	–	5k	0.6859	0.6774
CNN-5k	–	5k	0.7532	0.7450
CNN(TK-5k)	50k	–	0.7239	0.7208
CNN(TK-5k)*	50k	5k	0.7574	0.7493
CNN(TK-10k)	375k	–	0.7524	0.7471
CNN(TK-10k)*	375k	10k	**0.7796**	**0.7728**
Voting(TK+CNN)	–	10k	0.7838	0.7792

Table 5.1: Accuracy on the Quora dataset.

and the name in parenthesis indicates the model used for generating automatic data, e.g., CNN(TK-10k) means that a CNN has been pre-trained with the data labeled by a TK model trained on 10k GS data. The amount of automatic data for pre-training is in the second column, while the amount of GS data for training or fine-tuning (indicated by *) is in the third column. Finally, the results on the dev. and test sets are in the fourth and fifth columns.

We note that: first, NNs trained on 10k of GS data obtain higher accuracy than FV and TK on both dev. and test sets (see the first four lines);

Second, CNNs pre-trained with the data generated by FV or in a self-training setting, i.e., CNN(CNN-10k), and also fine-tuned do not improve[5] on the baseline model, i.e., CNN-10K, (see the second part of the table).

Third, when CNNs and LSTMs are trained on the data labeled by the TK model, match the TK model accuracy (the third part of the table). Most importantly, when they are fine-tuned on GS data, they obtain better results than the original models trained on the same amount of data, e.g., 1% accuracy over CNN-10k.

Next, the fourth part of the table shows that the improvement given by our method is still present when training TK (and fine-tuning the NNs) on less GS data, i.e., only 5k.

[5]The improvement of 0.5 is not statistically significant.

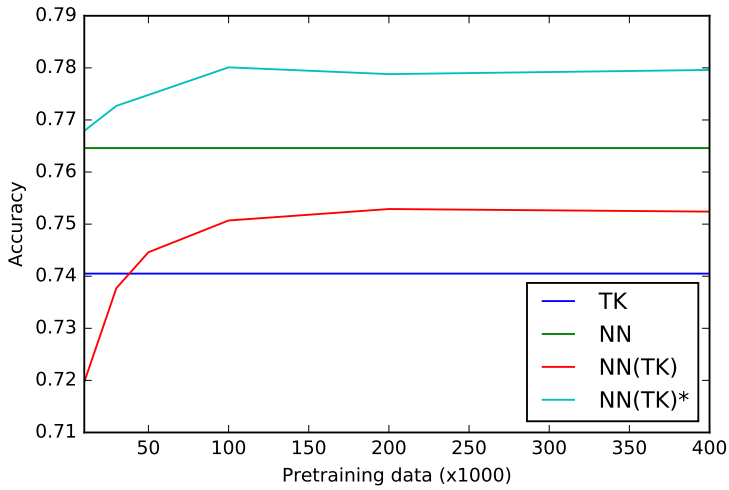

Figure 5.1: Impact of the pre-training data.

Additionally, the fifth section of the table shows significant improvement by training NNs on all available Quora data annotated by TK-10k (trained on just 10k). This suggests that NNs require more data to learn complex relational syntactic patterns expressed by TKs. However, the plot in Figure 5.1 shows that the improvement reaches a plateau around 100k examples.

Finally, in the last row of the table, we report the result of a voting approach using a combination of the normalized scores of TK-10k and CNN-10k. The accuracy is almost the same as CNN(TK-10k)*. This shows that NNs learn entirely combining a TK model, mainly exploiting syntax, and a CNN, only using lexical information. Note that the voting model is heavy to deploy as it uses syntactic parsing and the kernel algorithm, which has a time complexity quadratic in the number of support vectors.

5.2.4 Results on Qatar Living

Table 5.2 reports the results when applying our technique to a smaller and different dataset such as QL.

Here, CNNs have lower performance than TK models as 2,669 pairs are not enough to train their parameters, and the text is also noisy, i.e., there are many spelling errors. Despite this problem, the results show that CNNs can approximate the TK models well using a broad set of automatic data. For example, the CNN trained on 93k automatically annotated examples and then fine-tuned exhibits 0.4% accuracy improvement on the dev. set and almost 3% on the test set over TK models.

On the other hand, using too much automatically labeled data may hurt the test set's performance. This may be because the quality of information contained in the gold-labeled data deteriorates. In other words, using the right amount of weekly-supervision is an important

Model	Automatic Data	Dev	Dev (MAP)	Test	Test (MAP)
CNN		0.7000	0.6598	0.7514	0.7208
TK		0.7340	**0.6988**	0.7686	0.7424
CNN(TK)	50k	0.5580	0.6578	0.5428	0.7370
CNN(TK)*	50k	0.7160	0.6794	**0.7814**	0.7312
CNN(TK)	93k	0.7000	0.6782	0.6957	**0.7430**
CNN(TK)*	93k	**0.7380**	0.6782	0.7614	0.7320

Table 5.2: Accuracy on QL using all available GS data.

hyper-parameter that needs to be carefully chosen.

5.3 Related Work

Determining question similarity is one of the main challenges in building systems that answer real user questions [1] in community QA. Thus different approaches have been proposed. [51] used a language model based on the word translation table to compute the probability of generating a query question, given a target/related question.

[164] showed the effectiveness of phrase-based translation models on Yahoo! Answers.

[16, 30] proposed a similarity between two questions based on a language model that exploits the category structure of Yahoo! Answers.

[141] proposed a model to find semantically related questions by computing similarity between syntactic trees representing questions.

[52] and [163] used latent semantic topics that generate question/answer pairs.

Regarding the use of automatically labeled data, [11] applied semi-supervised approaches, such as self-training and co-training, to non-neural models. The main point of the chapter is using standard weakly-supervised methods to inject syntactic information in NNs.

[48] tried to combine symbolic representations with NNs by transferring structured information of logic rules into the weights of NNs. Our work is rather different as we inject syntactic, and not logic, information in NNs.

The work most similar to ours is the one by [27], who use Nystrom methods to compact the TK representation in embedding vectors and use the latter to train a feed-forward NNs. In contrast, we present a simpler approach, where NNs learn syntactic properties directly from data.

To our knowledge, ours is the first work trying to use NNs to learn structural information from data labeled by TK-based models. Finally, no systems of the SemEval challenges used NNs trained on syntactic information.

5.4 Summary

We have trained TK-based models in this work, which use structural information on relatively small data and apply them to new data to produce a much larger, automatically labeled dataset. Our experiments show that CNNs trained on automatic data improve their accuracy. We may speculate that CNNs learn syntactic structural information as TK models mainly use syntactic structures to label data, and other advanced models based on similarity feature vectors do not produce any improvement. Indeed, the latter only exploit lexical similarity measures, which are typically also generated by NNs. However, even if our conjecture were wrong, the bottom line would be that thanks to our approach, we can have efficient RelCNN models comparable to TK-based approaches without requiring a large amount of training data avoiding to use syntactic parsing and expensive TK processing at deployment time.

Chapter 6

CosineNet: Semantic Relation Learning

In the previous chapters, we have seen how the relational structure plays a crucial role in Answer Sentence Selection. The relational structure of question-answer candidate pairs is a type of local structure that consists of the set of relations between the words of the question and the words of the answer. The most straightforward relational information is the lexical overlap that indicates whether a word in the question appears in the answer and vice versa. Despite explicitly encoding this single feature that can improve A2S models [111, 161], these features explore the surface form of words and do not consider their semantics. Attention-based architecture has been recently proposed to overcome this challenge. For example, Wang and Jiang [145] exploits attention to perform semantic matching between the words in the question and the answer candidate – We have seen that the attention mechanism consists of a trainable similarity function between word vectors. A drawback of the attention mechanism is that the similarity function may be slow to compute since it is not only $O(N^2)$ in the number of words in the input text, but it typically requires additional parameters for training, increasing both the training time and the amount of data needed to obtain a stable model.

On the other end, Severyn and Moschitti [111] encode such lexical matching in the sentence encoders of both question and passage, obtaining a considerable improvement over the simple word overlap count models. However, word matching based on word surface form only captures a small part of all possible semantic or syntactic relations between words. Thus, we believe that enriching sentence models with partial matching, i.e., quantified by word similarity, could further improve the previous relational model based on word matching.

More specifically, in this chapter, we propose to model relations between words using their similarity in the embedding [81, 93] space. In particular, we exploit word embeddings' property to implicitly map similar words that are close to each other in the embedding space. For example, if the question contains the word *frog* while the passage contains *toad*, the model by Severyn and Moschitti [111] would not find any match and thus will not add any relational information to the sentence models. In contrast, in the embedding space generated by Glove

[93] on the CommonCrawl corpus[1], such words are the closest. Thus, if we consider embedding similarity, we can add many more important semantic relations between words to the sentence model. For this purpose, we match words in the deep neural network by Severyn and Moschitti [109] as follows: (i) we apply a cosine similarity score calculated between all the word embeddings of the question and answer passage (i.e., CosineNet); and (ii) we decide that two words match applying a max-pooling operation, i.e., selecting the word having the largest cosine with the target word.

Our model tested on the two benchmark datasets TrecQA [143] and WikiQA [149] produces a limited improvement of the base network. Such limited improvement may be caused by the fact that the information in word embeddings is noisy and not originally specialized for capturing the relatedness between two words [57, 32].

Recent work [57] proves that specializing word embeddings with the retrofitting approach improves the vectors' performance on a word similarity task. Retrofitting is a technique used to fine-tune pretrained embeddings to fit Knowledge Graphs. The process is done by minimizing the distance between a word and all its neighbors in the graph. Retrofitting's effectiveness has been further confirmed by its use in the best model of the Semeval 2017 Multilingual Word Similarity task [117]. The system provided multilingual word embeddings trained by retrofitting on a manually curated knowledgebase of common knowledge, i.e., Concepnet 5.5 [117].

Thus, we compare the impact of different publicly available word embeddings on APR using relational sentence models described above. Interestingly, when we use retrofitted embeddings in our cosine similarity based network, we obtain improvement of 6.5% and 2.7% on WikiQA and TrecQA, respectively, over the standard network, i.e., using standard Skip Gram or Glove embeddings. Furthermore, our model outperforms other efficient systems on the TrecQA dataset, targeted by many different neural network models.

6.1 Encoding Relational Information

6.1.1 Word Overlaps

Word overlaps are simple yet effective features for textual relatedness tasks. Those features count the number of overlapping words between the question and the answer passages in their simpler form. These simple features can achieve impressive results, which are superior or comparable to most systems published before 2013 on the benchmark datasets, particularly when the lexical overlaps are weighted with IDF features [105]. It is not surprising that Convolutional Neural Networks using those features produce strong baselines for the task [161, 109]. Despite its effectiveness, the count features are rather limited. A simple lexical overlap does not capture word similarity. This is made more critical by the fact that neural models cannot identify lexical overlap automatically, thus show limited ability at automatically extracting representation of document directly from raw text.

[1] https://nlp.stanford.edu/projects/glove/

6.1.2 Attention

The attention mechanism is the standard for capturing semantic textural relatedness in a neural network for A2S. The component implicitly creates alignments between all the words in the question and all the answer candidates' words. This is done in three steps: (i) a similarity function is used to compare each word in the question and the candidate's words. (ii) the alignment scores are normalized using a softmax function, enforcing a probability distribution. (iii) the probability distribution is used to create contextual words representation for each word i, performing a weighted average of all the word representation of the answer.

Although the technique has been proven successful in many models, it has two main drawbacks. First, the similarity function to compute the alignment is typically trained; hence it can create instability during training, mainly when the data is scarce. Second, the Softmax function enforces the alignment between the two sentences, even though it may not be present for negative candidates. These two drawbacks can introduce noise in the training process of Neural Architecture causing. The second drawback is in efficiency comes from the fact that the attention operation is $O(n^2)$ in the length of the sentence. Therefore, heaving complex alignment functions between the candidates significantly impact the efficiency of the model.

6.2 Word Embeddings

The core component of efficient Neural Architectures is word embeddings. Word embedding [81, 93] are fixed size dense vectors representing the words. Over the years, embedding pretrained on large corpora be an essential building block of many NLP systems [37]. Many reasons led to this success; among others, they can be used to transfer general information gathered from large corpora, and they can efficiently model word similarities. However, not all word embeddings show the same characteristics.

The two most popular techniques to create word embeddings are prediction based language and count based. An exemplar prediction based model is the Word2Vec skip-Gram model [81]. The skip-gram model predicts the context of a word (the words surrounding it), given the word's vector representation. This representation is learned over big corpora such as Wikipedia. A popular Count based algorithm is Glove [93]. In this case, the vectors are created by directly modeling the co-occurrences of words. Count based methods allow us to train the model on bigger corpora such as CommonCrawl, thus providing competitive or better results than prediction-based models.

Word embeddings can model syntactic and semantic relationships among words. Notably, unsupervised embeddings can capture word relatedness and word analogy information [102].

Despite word embeddings' success in capturing word relations, unsupervised pre-training of vectors often produces noisy representations. For example, Faruqui et al. [32] highlighted that word embeddings tend to cluster according to the frequencies of the words in the dataset. Therefore, this noise can limit their ability to model relationships among words that present a different frequency in the dataset.

Kiela, Hill, and Clark [57] proposed to use the retrofitting [33] technique to specialize embeddings at word relatedness tasks. This idea has been further confirmed by the winning team of the Semeval 2017: Multilingual Semantic Word Similarity challenge [117]. The authors specialized their word embeddings over the Concepnet knowledge graph. The main idea is to derive new vectors q_i, optimizing the objective of being close to the (i) original vector $\hat{q}_i$, which is computed using standard count and prediction based techniques; and (ii) embeddings of the neighbors in the graph E.

The loss is formally defined in the following way:

$$L(Q) = \sum_{i=1}^{n} \Big[\alpha_i \|q_i - \hat{q}_i\|^2 + \sum_{(i,j) \in E} \beta_{i,j} \|q_i - q_j\|^2 \Big]. \tag{6.1}$$

This model can infer embeddings for all the words in the knowledge graph. Words that do not have an original embedding $\hat{q}_i$ have an associated weight $\alpha_i = 0$. The scalar $\beta_{(i,j)}$ corresponds to the weight associated with the edge. Although these embeddings obtained an impressive result at the word similarity task, they have never been applied to downstream tasks such as Question Answering to model word relations between the question and answer passage.

6.3 Cosinenet

This section defines our neural network architecture to rank question and candidate pairs. It is an extension of the Convolutional Neural Network (CNNs) by Yu et al., Severyn and Moschitti [161, 109], which has been used as a baseline in several previous works.

The model $f : Q, A \rightarrow [0, 1]$ takes a pair of short texts, $q \in Q$ and $a \in A$, as input (where q is a question, and a is a candidate answer passage) and produces a score $y = f(q, a)$, representing the relevance of the answer passage with respect to the question. The model processes q and a independently using two different convolutional encoders. These operate in three steps: (i) map each word $i \in S$ in the sentence S to its corresponding word embedding $w_i \in \mathbb{R}^n$, (ii) perform a convolutional operation, and (iii) reduce the input sentence to a fixed size vector using a max-pooling operation. The output representations of the sentence encoders, i.e., q_{emb} and a_{emb}, are concatenated and then compared using a multi-layer perceptron (MLP). The latter provides a single score in the interval $[0, 1]$

Despite its effectiveness, the architecture still requires hand-crafted relational features to achieve state-of-the-art results [149, 21].

For that reason, as described in Section 6.1.1, different techniques to model interaction between sentences have been developed. The most used features for the task are based on lexical overlap [109]. This provides information about words appearing in both questions and answer candidates. The information is then encoded in the form of a global count feature for the pair (i.e., the total number of overlapping words, [161], or in the form of word overlap

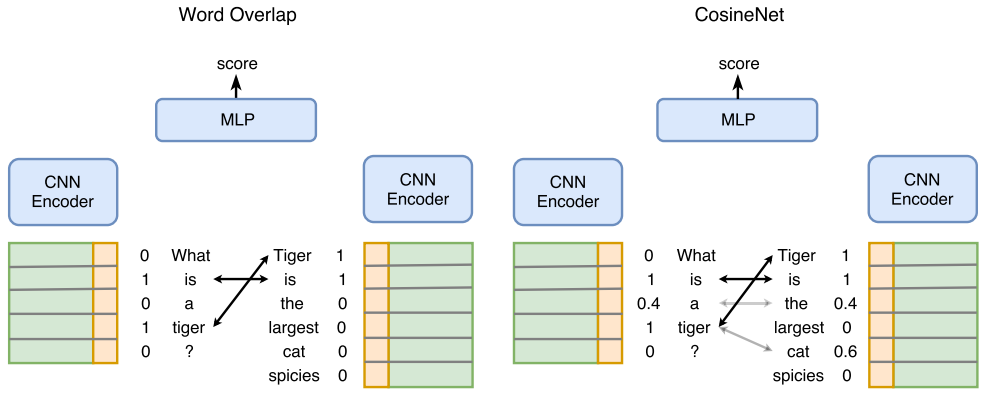

Figure 6.1: On the left, a representation of the network using word overlaps (only exact match of words are marked with ones). The figure on the right defines CosineNet; in this setting, word overlaps are substituted with the score of the more similar word present in the other document.

features [111]. The latter mark each word in the pair with a binary feature indicating if a word appears in both text pieces.

6.3.1 Word overlaps

This technique is first introduced by Severyn and Moschitti [111] and then adopted in subsequent works [105]. In this setting, the word overlap feature is a word-level binary feature marking words that appear in both question and passage, i.e., a word in a sentence is marked with 1 if an exact match of the word appears in the other sentence 0 otherwise. Formally:

$$OV_{word}(i) = \begin{cases} 1 & \text{if} \quad i \in A \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

i is a word in the question Q, and A is the candidate answer passage. This feature can be directly encoded in the model by concatenating it to the word embedding vector relative to each word.

Alternatively, the feature can be modeled using an embedding as proposed by Severyn and Moschitti [111]. Feature embeddings are fixed size trainable vectors $wo_{0/1} \in \mathbb{R}^m$ (word overlap embeddings) that are concatenated to the word embedding of each word and used as input of the convolutional operation (as depicted in Figure **??**).

$$OV_{emb}(i) = \begin{cases} wo_1 & \text{if} \quad i \in A \\ wo_0 & \text{otherwise} \end{cases} \tag{6.3}$$

We refer to these two models as Binary Word Overlaps (OV-Bin) and Embedded Word Overlaps (OV-Emb) respectfully for the rest of the work.

6.3.2 Cosine word overlaps

Word overlap features are an effective way to encode relations between short texts [105]. However, they cannot capture similarities and relatedness between words. Utilizing an exact lexical match, it is impossible to identify related words such as synonyms, hypernyms, or syntactic variations of the same words.

Word embeddings can model relations thanks to the latent information present in the vectors [81, 93]. For this reason, we propose to replace the lexical overlaps of the model with a cosine similarity score computed among the words in the question and the answer. Rather than encoding a feature that tells if *there is the same word in the other document*, the cosine word overlap feature tells *how much is similar the most similar word in the other document.* Formally, the feature is computed by the following:

$$OV_{cos}(i) = \max_{j \in A} \quad \text{similarity}(w_i, w_j), \tag{6.4}$$

where i is a word in the question q and w_i is its word embedding. The semantic word overlaps are then included in the model by concatenating the feature with each word's word embedding. For the rest of the discussion, this model will be called CosineNet.

6.4 Experiments

We experiment with our neural networks on two benchmark datasets for A2S. To verify the effectiveness of CosineNet, (i) we analyze the performance of the model when compared with a simple CNN without word overlaps (CNN), (ii) the network extended with simple binary word overlaps (OV-Bin), and (iii) word overlap embeddings (OV-Emb) as in [111]. Additionally, (iv), we analyze the impact of different word embeddings on the overall performance and the k-best similarity scores. Finally, (v) CosineNet is compared with the state of the art approaches.

6.4.1 Datasets

We tested our model on two datasets:
TrecQA: this is a factoid open-domain answer passage reranking dataset defined in [143]. The training set, composed of 1229 questions, was automatically constructed using regular expressions to mark relevant answers. The development and the test data contains 82 and 100 questions, respectively. In this work, we use the so-called Raw TrecQA evaluation setup, where questions with all positive and all negative answers are included in the dataset. This setup has been previously used by several related works[2].

WikiQA: TrecQA is a small and noisy dataset. In particular, the test set contains only 100 questions, of which only 65 contains both positive and negative answers, thus negatively affecting the reliability of the evaluation. Thus, to better assess our system's quality, we carry

[2] `https://aclweb.org/aclwiki/Question_Answering_(State_of_the_art)`

our evaluations on the WikiQA dataset [149]. This is larger than TrecQA, and it is created in a more realistic setting. Questions are real queries of the Bing search engine users, and answers are extracted from Wikipedia and annotated by crowd workers. Thus, the dataset is cleaner and an order of magnitude bigger than TrecQA for a total of 2,118 training, 296 development, and 633 test questions. Following the setup of Yin et al. [158], we remove the question without positive answers.

6.4.2 Model Setup

For both datasets, texts are tokenized using the Spacy v. 2.0.3[3] tokenizer. Words are mapped to fixed size word embeddings (whose size depends on the embeddings being used). All words that do not have an associated embedding are mapped to an unknown token. Embeddings are kept fixed (i.e., not fine-tuned on the task) for all the experiments. The convolution operations use a window of size five and maps each "5-gram" to a feature vector of size 100. The multi-layer perceptron uses a hidden layer of 200 dimensions with a dropout regularization of 0.5. The activation used for both the convolution and the hidden layers is the hyperbolic tangent ($tanh$).

The network is trained in classification setting (i.e., with binary cross-entropy loss) using the Adam optimizer with mini-batches of size 50, learning rate 10^{-5}, and weight decay of 10^{-5}. We use early stopping on the development set. The model is implemented in PyTorch v1.3.

6.4.3 Embeddings

We perform experiments with four different types of word embeddings. All the vectors are popular and used in different Question Answering systems, except the Numberbatch [117] embeddings.

The embeddings used in this work are the following:

Wiki+AQUAINT: Skip-gram [81] embeddings of size 50 trained on a joint corpus, the English Wikipedia, and the Aquaint corpus that are the sources of WikiQA and TrecQA, respectively. The vectors are used in several works, e.g., [109, 111, 130, 126]. The embeddings are available at the GitHub repository of Severyn and Moschitti [109][4].

Google News: Skip-gram [81] embeddings of size 300 trained on a joint corpus, trained on a corpus of news. A variation of the vectors are used in several works [138]. The embeddings are available Severyn and Moschitti [109][5].

Glove: Glove embeddings [93] of size 300 trained on the Common Crawl corpus using 840 billion tokens. Those are the most popular word vectors used in question answering [41, 145, 115][6].

[3] https://spacy.io/
[4] https://github.com/aseveryn/deep-qa
[5] https://code.google.com/archive/p/word2vec/
[6] https://nlp.stanford.edu/projects/glove/

Numberbatch: embeddings [117] of size 300 are an ensemble of different word embeddings trained on different sources and then retrofitted [33] on the Concepnet 5.5 [117] knowledge base. The embeddings are publicly available on their github repository[7].

6.4.4 Results

		DEV		TEST	
Model	**Embeddings**	MAP	MRR	MAP	MRR
CNN	Wiki+AQUAINT	65.85	70.30	60.95	65.89
CNN	Google News	61.53	64.53	61.61	67.91
CNN	Glove	67.81	70.13	65.52	72.15
CNN	Numberbatch	63.58	66.66	65.46	72.02
OV-Emb	Wiki+AQUAINT	76.94	83.05	71.75	77.32
OV-Emb	Google News	76.00	82.26	70.20	76.16
OV-Emb	Glove	80.08	84.87	76.84	80.87
OV-Emb	Numberbatch	77.05	84.45	73.09	78.47
OV-Bin	Wiki+AQUAINT	77.39	82.98	74.87	79.67
OV-Bin	Google News	75.33	80.97	73.67	79.88
OV-Bin	Glove	79.13	84.10	75.83	81.53
OV-Bin	Numberbatch	76.05	82.10	72.70	77.79
CosineNet	Wiki+AQUAINT	78.99	84.71	77.33	82.16
CosineNet	Google News	76.39	81.05	76.25	81.84
CosineNet	Glove	80.53	84.61	74.81	78.67
CosineNet	Numberbatch	81.10	85.44	79.39	83.57
Word overlaps	N/A	-	-	64.96	68.11
Idf-weighted word overlaps	N/A	-	-	70.14	76.88
Severyn and Moschitti [109]	Wiki+AQUAINT	-	-	74.59	80.78
Tay et al. [126]	Wiki+AQUAINT	-	-	74.99	81.53
Tay, Tuan, and Hui [125]	Pointcare	-	-	77.00	82.50
Rao, He, and Lin [99]	Glove	-	-	78.00	83.40

Table 6.1: Results on the TrecQA dataset

Table 6.1 presents the results on the TrecQA dataset. The first set of experiments exhibit the performance using CNNs without word overlap features. The results of the models vary depending on the word embeddings being used. However, the results are below the IDF-weighted word overlap baseline, stressing the need for including the word overlap baseline. Additionally, from the results, it is possible to understand that Glove embeddings that are trained on bigger unsupervised datasets perform better than word2vec-based embeddings.

The second set of experiments presents the result of our reimplementation of the [111] system with word overlap embeddings (OV-Emb). In this setting, the model achieves $\sim 10\%$ points of improvement compared with models not using lexical overlap features.

The third set of experiments shows that using word overlaps as binary (OV-Bin) feature is a

[7] `https://github.com/commonsense/conceptnet-numberbatch`

		DEV		TEST	
Model	**Embeddings**	MAP	MRR	MAP	MRR
CNN	Wiki+AQUAINT	70.41	70.99	63.56	64.93
CNN	Google News	68.93	69.00	60.97	62.31
CNN	Glove	70.69	71.17	64.88	66.31
CNN	Numberbatch	70.29	70.95	65.84	67.83
OV-Emb	Wiki+AQUAINT	71.61	71.92	69.74	71.21
OV-Emb	Google News	71.94	72.78	69.14	70.51
OV-Emb	Glove	73.06	73.72	68.14	69.74
OV-Emb	Numberbatch	71.32	71.81	70.07	71.52
OV-Bin	Wiki+AQUAINT	72.69	72.82	68.4	69.89
OV-Bin	Google News	72.73	72.89	64.66	66.25
OV-Bin	Glove	74.14	74.35	67.81	69.85
OV-Bin	Numberbatch	73.71	74.01	68.38	70.14
CosineNet	Wiki+AQUAINT	74.11	74.86	68.34	70.26
CosineNet	Google News	**76.43**	76.90	67.03	69.12
CosineNet	Glove	71.11	71.41	61.87	63.53
CosineNet	Numberbatch	75.70	76.22	**72.79**	**75.07**
Severyn and Moschitti [111]	Wiki+AQUAINT	-	-	69.51	71.07
Tay et al. [126]	Wiki+AQUAINT	-	-	70.90	72.30
Tay, Tuan, and Hui [125]	Pointcare	-	-	71.25	72.30
He, Gimpel, and Lin [41]	Glove	-	-	71.80	73.10
Shen, Yang, and Deng [115]	Glove	-	-	73.30	75.00
Wang and Jiang [145]	Google News	-	-	73.41	74.18
RelCNN+CTK[130]	Wiki+AQUAINT	75.50	76.66	74.17	**75.88**
Wang and Jiang [145]	Glove	-	-	**74.33**	75.40

Table 6.2: Results on the WikiQA dataset

viable way to model word overlaps and obtain comparable (or better) results in the OV-Emb model.

The fourth set of experiments shows that an embedding similarity is a practical approach that outperforms the lexical overlap feature of OV-Bin. It is important to note that the capability of performing word similarity of different types of embeddings determines the amount of improvement. In particular, CosineNet using retrofitted word overlap embeddings gives $\sim 5\%$ of improvement. The obtained result, i.e., 79.39, outperforms the state-of-the-art models on the Raw TrecQA dataset.

We performed the same set of experiments on the WikiQA dataset. Interestingly, the overlap features still affect the model's performance, but the impact of the overlap is lower. This is expected since the WikiQA dataset is more challenging and relies less on lexical overlap [149]. CosineNet improves the overall performance of the system when using the Numberbatch retrofitted embeddings. The model achieves a MAP score of 72.79, which is an improvement of ~ 3 over the OV-Emb baseline model.

The proposed similarity-based model obtains results comparable to the attention-based mod-

els without requiring any additional trainable parameters (except Compare-Aggregate CNN [145] that exhibits slightly better results, ~ 1.5 MAP and ~ 0.33 MRR).

6.5 Summary

In this chapter, we presented CosineNet, a DNN model for Question Answering, which exploits word embedding similarities to encode relations among words from the question and answer candidates. CosineNet achieves competitive results when compared with the attention-based model when using word embeddings trained with retrofitting. The proposed model is much more efficient than attention based architecture[8]. The raw TrecQA dataset results show a ~ 2.5 improvement over the MAP achieved by the previous best system, which does not use contextualized language models. Interestingly, our model can achieve results comparable to complex attentive models for the more challenging WikiQA dataset. The results further confirm that to exploit word embeddings similarities in a downstream task fully, the embedding model needs to be specialized to the word relatedness task, e.g., using retrofitting. CosineNet does not require any additional parameter than the baseline RelCNN model, thus preserving its simplicity and efficiency while obtaining results comparable with models based on the attention mechanism.

[8] See Chapter 7 for additional results on efficiency

Part III

Encoding Global Structure

Chapter 7

Encoding the Document Structure

Although Answer Sentence Selection is a structured output problem, most neural models assign a score to each sentence, independently of the others, using a point-wise approach both for training and inference. A possible explanation for this choice could be that in standard IR-based ranking, such as TrecQA [144], there is a lack of strong dependencies between ranking candidates since passages are retrieved from different sources by a search engine.

However, QA shows somewhat different settings, e.g., in the WikiQA task [149], all the candidates for each question are extracted from the same document, a Wikipedia article. Thus, although the initial document is retrieved with a search engine, the set of answer candidates is given by all the sentences in the first paragraph of the article. WikiQA provides sentences in their *natural* order, thus preserving the source document discourse structure. This also induces a not uniformly answer distribution. For example, Figure 7.1 shows that the first sentence of the document answers 46.09% of the questions, which would lead to an MRR of 64.27 evaluated over the sentence rank preserving the original sequential structure. This performance is close to a basic Convolutional Neural Network (CNN) trained for the task.

In this chapter, we first verify the hypothesis that in several QA applications, the data often presents an underlying global structure. The latter refers to the relation between a question with all its candidates and the candidates' inter-dependencies. Then, we propose a neural network that can capture the above structure in the actual rank. For this purpose, we need to model two main logic blocks: (i) An efficient model that captures intra-pair relations between the question and each candidate; and (ii) the structure of the sentence rank, e.g., the similarities and dissimilarities between candidates for the same question.

For the first point, we capitalize on the work in our previous chapter, i.e., the Cosinenet. Regarding the second block, we use an additional layer constituted by a Recurrent Neural Network, which is fed with the representation of question/answer candidate pairs. The latter are joint representations of the question and the answer, obtained throughout the Cosinenet.

We tested our models on three different datasets, the well-known WikiQA dataset, the adaptations of SQuAD [98], and Natural Questions [65] datasets to the AS2 task. Additionally, the results show that the global component outperforms the same models, not exploiting it.

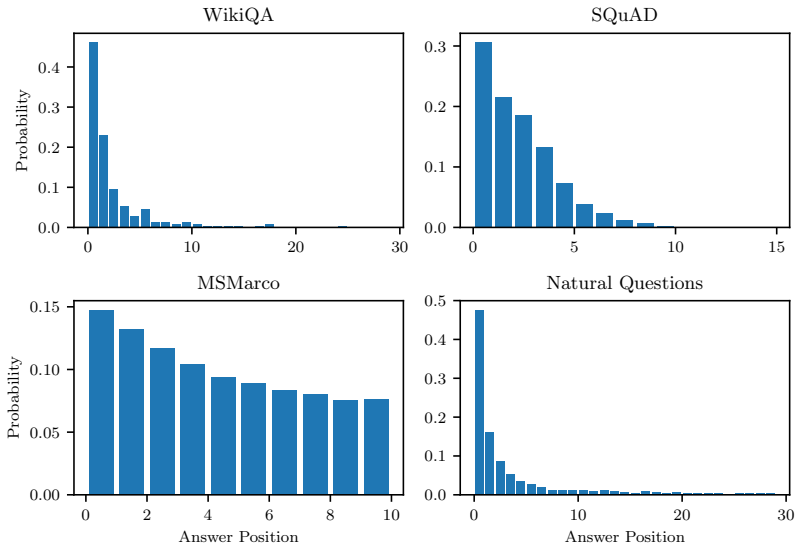

Figure 7.1: The community Question Answering task structure

Despite this adds a small overhead during training and testing, we show that the RNN component is crucial to capture the structure of data. For example, The results on WikiQA show that BiRNN added to the Cosinenet improves of $\sim$ 4 points previous baselines.

The proposed model results do not compare with the current state-of-the-art in terms of pure performances; see Table 7.2. The main reason for that comes from the fact that the best performing system at the task of A2S is based on the Transformer architecture, and they are pre-trained on a large amount of data. These models constitute a problem for many real-world scenarios because the size of large pre-trained models like BERT limits their applicability in production environments where high transactions per second are required. Additionally, transformer-based architectures require many pre-training resources, e.g., both data and compute power (TPUs). These resources may not be available for low resource languages or domain-specific applications.

In this work, we focus on efficient architectures that can obtain high transactions per second. Our approach can process 700 questions per second (with all their answer candidates) during training. In comparison, BERT-base processes six questions per second on the same machine.

7.1 Answer Sentence Selection Datasets

AS2 datasets can be divided into two categories: retrieval based and document-based. The difference between the two categories resides in the source of the answer candidates. In the former, answer candidates are retrieved from a search engine, i.e., TrecQA [144] and, more recently, MSMarco [5]. For the latter, a search engine is often used to retrieve the

	WikiQA	SQuAD	NQ-LA
# questions (Q)	633	11873	6230
# sentences (C)	6165	63959	193k
% Q answered	38.39	49.92	55.47
avg. # passages	9.74	5.38	30.95
avg. Q lenght	7.28	10.02	9.38
avg. C lenght	25.36	23.75	98.76
P@1 (random)	14.43	18.34	3.24
MAP (random)	25.15	43.81	12.33
P@1 (RR)	46.09	30.54	46.06
MAP (RR)	64.21	53.53	57.30
P@1 (WO)	32.51	65.48	23.06
MAP (WO)	51.02	77.90	38.08
P@1 (WO+RR)	56.38	73.12	41.01
MAP (WO+RR)	68.25	83.60	53.98

Table 7.1: Statistics of the different datasets (the test-set are taken into account).

relevant document, but the task is to select the relevant answer candidate from the document itself. Notable examples of document-based AS2 are the WikiQA dataset [149] and the "long-answer" version of Natural Question [64].

Despite the heterogeneous nature of the datasets, both types present strong features for detecting relevant answers in the candidate set: substantial lexical overlap between the question and the answer candidate and a global structure, i.e., the original order of the sentences in the rank.

7.1.0.1 Lexical Overlap

One of the strongest features in AS2 datasets is the lexical overlap, i.e., whether words appear in both questions and answer candidates. The importance of this feature is highlighted in Table 7.1. We used the number of unique words in both the question and the candidates as a single feature to rank question-answer pairs. From the table, it is clear that this feature alone significantly outperforms the random baseline in most datasets. For Squad-sent, which is an adaptation of the SQuAD v. 2.0 dataset where the task is to identify the sentence containing the answer, in particular, this feature alone identifies the sentence containing the answer 65.48% of the times. All of the recent literature models have tried to model such features; for example, [111] uses the relational feature that marks words appearing in both the question. The answer attention mechanism has been primarily used by many state-of-the-art approaches [145, 10, 114].

7.1.1 Global Structure

Another relevant feature for AS2 datasets is the global structure present in the original rank. The structure of the document in a document-based dataset provides a critical signal needed to answer sentence selection. Table 7.1 shows that in the case of WikiQA, SQuAD, and Natural

Questions, there is a high chance that the answer is contained in the first sentence/paragraph. This is particularly true for WikiQA and Natural Questions. In these datasets, the P@1 using the reciprocal of the rank in the original sequence as a signal for ranking is ~ 46, i.e., the reciprocal of the original rank $1/rank$. There may be several reasons for this distribution. For example, we believe that there is an intrinsic correlation between the real world distribution of questions and the Wikipedia document structure. We believe that encyclopedic knowledge is usually organized in a way that more general information about a topic is summarized and organized at the beginning of the document.

In contrast, by construction, the signal is less present in datasets such as SQuAD, where annotators are asked to write questions after reading the whole paragraph. Thus the answer distribution is less skewed. However, for the same reason, it is essential to note that annotators tend to introduce more lexical overlap bias when writing questions after reading the source of the answers.

Additionally, Table 7.1 shows that the combination of the two features, word-overlap, and reciprocal rank, gives a strong baseline for all the datasets in consideration. This simple rule-based model ranks candidates according to the lexical overlap between question and candidates, and, in the case when two sentences have the same amount of overlapping words, it uses the reciprocal rank as a discriminator.

7.2 Efficient Model for AS2

To build an efficient yet accurate model for AS2, it is crucial to leverage all the strong signals present in the dataset without increasing the model's complexity. For this reason, we design a model as follow:

- We build an efficient encoder to capture the question-candidate pair's lexical-overlap, i.e., the cosinenet.
- We add a recursive neural network on top of the question-candidate pairs to capture the original rank's global structure.
- We apply a global, list-wise, optimization approach to rank all the candidate pairs jointly.

7.2.1 Cosinenet

The Cosinenet has three building blocks: (i) a word-relatedness encoder that performs the cosine similarity between the word embeddings in the question and the answer, which extracts word relatedness features; (ii) similarly to [111], the relational features are concatenated to the word embeddings and fed to one layer of CNN, to create a representation for the question and candidate pair; and (iii) similarly to [20], the information of the question and the candidate is combined at classification stage, by concatenating the vectors. As a result of the point-wise multiplication between the vector of the question and the answer, their difference.

7.2.1.1 Word-Relatedness encoder

To encode the word-relatedness information, we first map the words in the question and the answer to their respective word embeddings. We then perform a comparison between all the embeddings in the question w_i^q and all the embedding of the answer w_j^c using the cosine similarity.

$$r_{i,j} = \frac{w_i^q w_j^c}{\|w_i^q\| \|w_j^c\|} \tag{7.1}$$

Instead of performing the weighted sum of the embeddings as in standard attention, for each word in the question, we take the maximum relatedness score between the word embedding of the question and each word embedding of the candidate, i.e., $\mathbf{r}_i = max_j(r_{i,j})$. The same process is performed for each word in the answer. This feature represents *how much a word is similar to the most similar word in the other text.*

This simple feature is concatenated with the word embedding of the question $\hat{w}_i^q = [\hat{w}_i^q; \mathbf{r}_i]$ and vice-versa $\hat{w}_j^c = [\hat{w}_j^c; \mathbf{r}_j]$. The resulting word representations are passed to the question-candidate encoder to create the pair representation.

It is important to note that we keep the word embedding static during training, so this operation does not cause much overhead during training as we do not need to back-propagate through it.

For this model, we use Numberbatch word embeddings [117], as unsupervised word embedding representations, such as Glove [93] often provides noisy representations for semantic relatedness. For example, [32] showed that standard word embeddings tend to cluster according to the frequency of words in the unsupervised dataset. For this reason, [117] adopted retrofitting. This technique aims to reduce the distance between word embeddings of entities related to a knowledge base, i.e., ConcepNet.

7.2.1.2 Question-Candidate encoder

Similarly to [111] we encode the question and candidate independently using two single layers of CNN with a kernel size of 5 and global max pooling. This results in an embedding for the question q_e and the candidate c_e. The two embeddings are combined using them in a pair embedding, concatenating the element-wise multiplication of the two embeddings with their difference, i.e., $qc_e = [q_e \odot c_e; q_e - c_e]$

7.2.2 Global optimization

Standard approaches take the pair representation qc_e and apply a feed-forward network that outputs the score s_i for the pair (q, c_i). However, this simple model cannot capture the inter-dependencies between the candidates for the question q. Thus, it does not capture the global structure of the rank.

In this work, to leverage the global structure of the rank, we make use of a Recurrent Neural Network applied on top of the qc_e representations for each c_i of a given question q. The

resulting contextual representations $\hat{qc}_e$ are passed to the feed-forward network (in our experiments, we use a single layer to produce the final score).

Finally, similarly to [10], we apply a softmax function to the scores $s^1, .., s^n$ of all the n candidate answers of a given question. Then, we minimize the KL-divergence of the predicted probabilities and the normalized gold labels. [10] proved that this approach could improve the convergence speed of the model compared with point-wise approaches.

7.3 Experiments

7.3.1 Datasets

To align with previous works, we remove questions without answers, and we lowercase and tokenize question and passages[1]. We used the following datasets:

WikiQA Questions are randomly sampled from the Bing search engine logs. The candidate answers are the sentences that constitute the first paragraph of the related Wikipedia article. Additionally, answers are concentrated in the first part of the paragraph.

SQuAD-sent For each question, the SQuAD dataset provides a paragraph and annotations for the exact answer span. For the answer sentence selection task, we split the paragraph into sentences using the SpaCy sentence tokenizer. The gold labels are inferred from the answer spans, i.e., the sentence that contains the answer. Since the SQuAD test set is not publicly available, we use the validation set for testing and 10% of the questions in training set for validation. In contrast with our dataset, QNLI (the GLUE adaptation of SQuAD [137]) provides an even amount of positive and negative question/answer pairs, sampled from the SQuAD dataset. This creates decontextualized sentences, which prevent to exploit the sequential structure. In contrast, we propose a dataset that maintains the original document structure. Our models are evaluated using two metrics: precision at 1 (P@1) and Mean Reciprocal Rank (MRR). The SQuAD dataset exhibits a much stronger lexical overlap between question and answers passages. This effect can be noted in Tab. 7.1: the simple word-overlaps count baseline, i.e., the number of unique words that appear in both question and passage, achieves a P@1 of 65.48.

NQ-LA The Natural Question dataset uses question sampled from the Google search engine logs. The questions are given to the annotators together with the retrieved Wikipedia page. The annotator is asked to select (i) a long answer, i.e., the smallest HTML bounding box containing all the information needed to answer the question, and (ii) a short answer (if available), that is, the actual answer to the question. Although the dataset can be useful in an industrial setting, we prefer to consider only paragraphs as long answers, thus removing tables and lists. A paragraph is defined by the HTML bounding box <p >and <\p >. The dataset has a similar answer distribution of the others, i.e., P@1 46.06% and MAP 57.30,

[1] Tokenization with SpaCy v2.1 `https://spacy.io`

Model	MAP	MRR
Baselines		
RR	64.21	64.26
WO	51.02	51.24
WO+RR	68.25	69.43
Related Work w/o pre training		
Tay, Tuan, and Hui [125]	71.20	72.70
Wang and Jiang [145]	74.33	75.45
Wang and Jiang [145]†	72.38 ± 1.4	73.44 ± 1.5
Sha et al. [114]	74.62	75.76
Bian et al. [10]	**75.40**	**76.40**
Related Word with pre-training		
Yoon, Shin, and Jung [159]	83.40	84.80
Lai et al. [67]	85.70	87.20
Garg, Vu, and Moschitti [36]	92.00	93.30

Table 7.2: Related Work on WikiQA test-set. †We run the official implementation with different random seeds.

even if the candidates are much longer (paragraphs). This may be considered surprising as a Wikipedia page contains an average of 30.95 paragraphs of 98.76 words. We note that most pages give essential information about an entity in the first paragraph, i.e., in the summary paragraph. Similarly to SQuAD, the annotations for the test set of Natural Questions are not publicly available. Therefore, we used the official development set as our test set and a portion of the validation training set.

7.3.2 Models and parameters

In our experiments, we used two different encoder architectures: the newly proposed Cosinenet and our re-implementation of the Compare-Aggregate CA architecture. The former uses static numberbatch embeddings[2] of size 300; the convolution hidden layer of size 300 and a kernel of size 5. For the compare aggregate architecture, we use the standard parameters of the original paper, but in contrast with it, we keep the embedding static as we empirically found that it leads to similar results while having the highest number of trainable parameters. For the RNN and the LSTM, we used the same hidden size as the input, i.e., double the size of the convolutional operation hidden size. For the Bidirectional variations, i.e., BiRNN and BiLSTM, we set the hidden size as half of the input size in each direction, resulting in a comparable number of parameters.

All the models were trained for three epoch using slanted triangular learning rate scheduling [46] without early stopping. In the case of the point-wise models, we used Adam optimizer with a maximum learning rate set at 2e-3, whereas for the list-wise approaches, we used a learning rate of 2e-4. All the experiments are performed on an Nvidia GTX 1080 ti GPU and an Intel Core I9-7900X processor.

[2] `https://github.com/commonsense/conceptnet-numberbatch`

Model	RNN	MAP	MRR	params	train-time
		Baselines			
RR	-	64.21	64.26	-	-
WO	-	51.02	51.24	-	-
WO + RR	-	68.25	69.43	-	-
		Our Models			
Cosinenet	-	70.95 ± 0.6	72.86 ± 0.7	904k	6 sec
Cosinenet$_{list}$	-	71.22 ± 0.2	73.07 ± 0.3	904k	5.5 sec
Cosinenet$_{list}$	RNN	74.78 ± 0.6	76.35 ± 0.6	1.17M	7.5 sec
Cosinenet$_{list}$	BiRNN	$\mathbf{75.62} \pm 0.8$	$\mathbf{77.13} \pm 0.9$	1.12M	8.9 sec
Cosinenet$_{list}$	LSTM	74.31 ± 0.8	75.78 ± 0.9	1.99M	7 sec
Cosinenet$_{list}$	BiLSTM	75.32 ± 0.6	76.85 ± 0.5	1.81M	9.5 sec
CA	-	72.03 ± 1.6	73.39 ± 1.7	2.89M	19 sec
CA$_{list}$	-	71.43 ± 1.0	73.55 ± 1.0	2.89M	18 sec
CA$_{list}$	RNN	74.73 ± 1.0	76.35 ± 1.2	5.05M	20 sec
CA$_{list}$	BiRNN	74.97 ± 1.2	76.44 ± 1.2	4.87M	21 sec
CA$_{list}$	LSTM	74.82 ± 1.1	76.42 ± 1.2	11.53M	25 sec
CA$_{list}$	BiLSTM	74.27 ± 1.0	75.74 ± 1.1	10.81M	25 sec
BERT$_{base}$	-	81.32	82.50	110.00M	17 min 50 sec

Table 7.3: Model comparison on the WikiQA test-set.

Model	RNN	P@1	MRR	params	train-time
		Baselines			
RR	-	30.55	53.53	-	-
WO	-	65.48	77.90	-	-
WO + RR	-	73.12	83.60	-	-
		Our Models			
Cosinenet		86.18 ± 0.2	91.81 ± 0.1	904k	1 min 47 sec
Cosinenet$_{list}$		85.12 ± 0.1	91.16 ± 0.1	904k	8 min 10 sec
Cosinenet$_{list}$	BiRNN	86.18 ± 0.2	91.97 ± 0.1	1.12M	12 min 30 sec
CA		85.71 ± 0.2	91.49 ± 0.1	2.89M	6min 30 sec
CA$_{list}$		85.17 ± 0.6	90.69 ± 1.0	2.89M	24 min 11 sec
CA$_{list}$	BiRNN	$\mathbf{86.32} \pm 0.3$	$\mathbf{92.05} \pm 0.2$	4.87M	28 min 30 sec
BERT$_{base}$	-	92.44	95.62	110.00M	6 hr 50 min

Table 7.4: Model comparison on the SQuAD sent test-set.

7.3.3 Results

Table 7.3 shows the results in the WikiQA dataset. The first block shows the baselines; these models are computed using the simple features described in Section 7.1: the lexical overlap and the reciprocal rank, both achieving results comparable with baseline CNN architectures. The second block of results shows the performance of models from the literature that do not use contextualized pretrained language models. The third section presents the results of models that make use of both pretrained language models and transfer learning. In particular,

Model	RNN	MAP	MRR	params	train-time
		Baselines			
RR	-	57.30	60.41	-	-
WO	-	38.09	39.57	-	-
WO + RR	-	53.98	56.45	-	-
		Our Models			
Cosinenet		69.74	72.69	904k	1 hr 13 min
Cosinenet$_{list}$		68.16	71.06	904k	17 min
Cosinenet$_{list}$	BiRNN	73.28	76.05	1.12M	34 min
CA		69.88	72.77	2.89M	5h 39min
CA$_{list}$		69.82	72.78	2.89M	2h
CA$_{list}$	BiRNN	**74.21**	**76.88**	4.87M	2h 10 min

Table 7.5: Model comparison on the NQ-LA test-set.

[159] uses the ELMO language model and transfer learning on the QNLI dataset; [67] uses BERT and performs transfer learning on the QNLI dataset, and [36] uses RoBERTa large and performs transfer learning from he Natural Question dataset. Our approach's performance is shown in table 7.3: the Cosinenet architecture achieves comparable results compared with the more complex compare aggregate framework while having much lower trainable parameters. Cosinenet and CA are the standard point-wise approach, trained on all the data using a fixed batch size and binary cross-entropy (BCE). $Cosinenet_{list}$ and CA_{list} are the same base architecture but trained with a listwise approach, with KL-Divergence loss on all the question-answer candidate pairs for the same question. We then analyzed what RNN architecture is best suited for identifying the structure of the original rank. For both the Cosinenet and the CA architecture, we found no statistical difference between the RNN and LSTM. However, Bidirectional RNNs seems to outperform the other models consistently.

The same is true for the SQuAD dataset in Table 7.4. Our base architecture achieves comparable results with the more complex CA model. Additionally, adding the BiRNN to the model further improves the results. It is important to note how simple semantic matching models can achieve very high results on the task. In SQuAD, the lexical overlap feature is more prominent than the global rank feature. Therefore the impact is given by the Global Optimization with the BiRNN.

Finally, Table 7.5 shows the results of the bigger Natural Question dataset. Despite the different nature of the data, i.e., the candidates are paragraphs rather than sentences, our proposed model improves the baselines in particular when combined with the BiRNN.

7.3.4 Performance analysis

From the experiments on all three datasets, the proposed architecture is more efficient than the compare aggregate architecture as it has much fewer parameters and a more efficient attention computation. On the small WikiQA dataset, the model takes up to 9.5 seconds to train, achieving results comparable with the best models that do not use pretrained language

models — in contrast, training BERT-base on the same dataset tasks requires 17 min and 50 sec. Additionally, our model has roughly 100x less trainable parameters than BERT-base. The difference between the two models is more apparent when comparing the training time on the SQuAD dataset. The BiRNN based Cosinenet trains in slightly more than 12 minutes, which is much lower than the 6 hours and 50 minutes needed to train BERT-base on the same task, and around half the time needed to train the Compare-Aggregate architecture.

7.4 Summary

In this work, we argue that by exploiting the original rank's intrinsic structure and an effective word-relatedness encoder, we can achieve competitive results compared to the state of the art while retaining high efficiency. We first analyzed the structure of standard datasets, highlighting the importance of the original rank's global structure. Capitalizing on this, we propose a model that both exploits the rank structure using a simple RNN and the standard word-relatedness features, while preserving high efficiency. The model uses around 1M parameters depending on the configuration and achieves better results than the previous work that does not use computationally expensive pre-trained language models. Our model takes 9.5 seconds to train on the WikiQA dataset, i.e., very fast compared to the $\sim$ 18 minutes required by a standard BERT-base fine-tuning. On SQuAD, this difference is even higher, i.e., minutes vs. hours, required by Transformer-based models.

Chapter 8

Encoding the Task Structure

Community Question Answering (cQA) websites enable users to freely ask questions in web forums and get some right answers in the form of comments from other users. Given many question/answer pairs available on cQA sites, researchers started to investigate the possibility of exploiting user-generated content for training automatic QA systems. Unfortunately, the cQA scenario's text is rather noisy; therefore, providing models that outperform the simple bag-of-words representation can appear rather complicated.

The task of cQA is formalized as follows: given a new user question, q_{new}, and a set of forum questions, Q, answered by a comment set, C, the main task consists in determining whether a comment $c \in C$ is a suitable answer to q_{new} or not. Interestingly, the task can be divided into three sub-tasks, as shown in Fig. 8.1: given q_{new}, the main Task C is about directly retrieving a relevant comment from the entire forum data. This can also be achieved by solving Task B to find a similar question, q_{rel}, and then executing Task A to select comments, c_{rel}, relevant to q_{rel}.

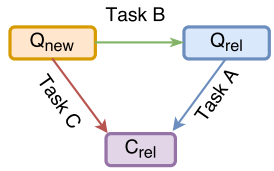

Figure 8.1: The 3 tasks of cQA at SemEval: the arrows show the relations between the new and the related questions and the related comments.

Figure 8.1 shows the structure of the task. The three tasks of cQA are semantically connected and follows a strict structure, i.e., if q_{new} and q_{rel} are semantically equivalent than a good comment c_{rel} for q_{rel} is likely to be a good comments for q_{new}. Most systems have extensively explored this relation at the Semeval 2016: Community Question Answering competition [89], which used a pipeline approach for solving Task C by solving Task B and Task A. In most cases, the pipeline approach outperformed the systems that were trying to solve Task C directly. One of the reasons for this may be due to the fact that between q_{new} and c_{rel} there is less lexical overlap than between q_{rel} and c_{rel}.

Another result of the Semeval competition is that traditional feature-based approaches and Structured Tree Kernel-based architectures outperform most neural network approaches to cQA. One reason for that comes from the fact that high-quality data for cQA is challenging to collect; hence, a dataset to effectively train neural architectures is expensive.

Finding a general solution to data scarcity for any task is an open issue; however, we can alleviate it for some classes of applications by using multi-task learning (MTL). Recent work has shown that it is possible to *jointly train* a general system for solving different tasks simultaneously. For example, [25] used MTL to train a neural network for carrying out many sequence labeling tasks (e.g., pos-tagging, named entity recognition, etc.), whereas [73] trained a DNN with MTL to perform multi-domain query classification and reranking of web search results with respect to user queries.

The above work has shown that MTL can be effectively used to improve NNs by leveraging different data kinds. However, the obtained improvement over the base DNN was limited to 1-2 points, raising the question if this is the kind of enhancement we should expect from MTL. Analyzing the different tasks involved in the model by [73], it appears that query classification provides little and very coarse information to the document ranking task. Indeed, although the vectors of queries and documents lie in the same space, the query classifier only chooses between four different categories, *restaurant*, *hotel*, *flight* and *nightlife*, whereas the documents can potentially span infinite subtopics.

In this chapter, we conjecture that when MTL tasks are more semantically connected, more considerable improvement can be obtained. More specifically, MTL can work more effectively when we can encode the instances from different tasks using the same representation layer expressing *similar semantics*.

Given the above setting, we define an MTL model that solves Task C, learning at the same time the auxiliary tasks A and B. Considering that (i) q_{new} and q_{rel} have the same nature and (ii) comments tend to be short. Their text is comparable to one of the questions,[1] we could model a significant, shared semantic representation. Indeed, our experiments with the data from SemEval 2016 Task 3 [90] show that our MTL approach improves the single DNN for solving Task C by roughly 8 points in MAP (almost 20% of relative improvement). Finally, given the strong connection between the objective functions of the DNNs, we could train our network with the three different tasks simultaneously, performing a single forward-backward operation over the network.

8.1 Our MTL model for cQA

MTL aims to learn several related tasks simultaneously to improve some (or possibly all) tasks using joint information [17]. MTL is particularly well-suited for modeling Task C as it is a composition of tasks A and B; thus, it can benefit from having both questions q_{new} and q_{rel} in input better model the interaction between the new question and the comment.

[1] In cQA domains, these are typically longer than standard questions, i.e., up to few paragraphs containing subquestions and an introduction.

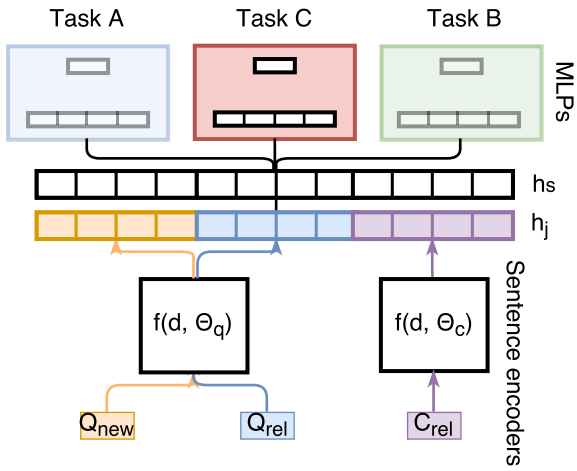

Figure 8.2: Our MTL architecture for cQA. Given the input sentences q_{new}, q_{rel} and c_{rel} (at the bottom), the NN passes them to the sentence encoders. Their output is concatenated into a new vector, h_j, and fed to a hidden layer, h_s, whose output is passed to three independent multi-layer perceptrons. The latter produce the scores for the individual tasks.

More precisely, it can use the triplets, $\langle q_{new}, q_{rel}, c_{rel} \rangle$, in the learning process, where the interaction between the triplet members is exploited during the joint training of the three models for the tasks A, B, and C.

A better model for question-comment similarity or question-question similarity can lead to a better model for new question-comment similarity (Task C).

Additionally, each thread in the SemEval dataset is annotated with the labels for all the three tasks, and therefore it is possible to apply joint learning directly (using a global loss) rather than training the network by optimizing the loss of the three single tasks independently. Note that, in previous work [25, 73], each input example was annotated for only one task, and training the model required to alternate examples from the different tasks.

8.1.1 Joint Learning Architecture

Our joint learning architecture is depicted in Figure 8.2, it takes three pieces of text as input, i.e, a new question, q_{new}, the related question, q_{rel}, and its comment, c_{rel}, and produces three fixed size representations, $x_{q_{new}}$, $x_{q_{rel}}$ and $x_{c_{rel}}$, respectively. This process is performed using the sentence encoders, $x_d = f(d, \theta_d)$, where d is the input text and θ_d is the set of parameters of the sentence encoder. In previous work, different sentence encoders have been proposed, e.g., Convolutional Neural Networks (CNNs) with max-pooling [59, 109] and Long-short term memory (LSTM) networks [45].

We concatenate the three representations, $h_j = [x_{q_{new}}, x_{q_{rel}}, x_{c_{rel}}]$, and fed them to a hidden layer to create a shared input representation for the three tasks, $h_s = \sigma(Wh_j + b)$. Next, we connect the output of h_s to three independent Multi-Layer Perceptrons (MLP), which produce

	Task A	Task B	Task C
Train	37.51%	39.41%	9.9%
Train + ED	37.47%	64.38%	21.25%
Dev	33.52%	42.8%	6.9%
Test	40.64%	33.28%	9.3%

Table 8.1: Percentage of positive examples in the training datasets for each task.

the scores for the three tasks.

At training time, we compute the global loss as the sum of the individual losses for the three tasks for each example, where each loss is computed as binary cross-entropy.

8.1.2 Shared Sentence Models

The SemEval dataset contains ten times less new questions than related questions by construction. However, all questions have the same nature (i.e., generated by forum users), thus, we can share the parameters of their sentence models as depicted in Figure 8.2. Formally, let $x_d = f(d, \theta)$ be a sentence model for a text, d, with parameters, θ, i.e., the embedding weights and the convolutional filters: in a standard setting, each sentence model uses a different set of parameters $\theta_{q_{new}}$, $\theta_{q_{rel}}$ and $\theta_{c_{rel}}$. In contrast, our proposed sentence model encodes both the questions, q_{new} and q_{rel}, using the same set of parameters θ_q.

8.2 Experiments

8.2.1 Setup

Dataset: the data for the tasks mentioned above are distributed in three datasets for Task A, which contains $6,938$ related questions and $40,288$ comments. Each comment in the dataset was annotated with a label indicating its relevancy to its thread question. Task B, which contains 317 new questions. For each new question, ten related questions were retrieved, summing to $3,169$ related questions. In this case, the related questions were annotated with a relevancy label, which tells if they are relevant to the new question. Task C contains 317 new questions, together with $3,169$ related questions (same as in Task B) and $31,690$ comments. Each comment was labeled with its relevancy with respect to the new question. Each of the three datasets is, in turn, divided into training, dev. and test sets.

Table 8.1 reports the label distributions with respect to the different datasets. The data for Task C presents a higher number of negative than positive examples. Thus, we automatically extended the set of positive examples in our joint MTL training set using Task A data. More specifically, we take the pair (q_{rel}, c_{rel}) from the training set of Task A and create the triples, $(q_{rel}, q_{rel}, c_{rel})$, where the label for question-question similarity is positive, and the labels for Task C are inherited from those of Task A. We ensured that the questions in the extended data (ED) generated from the training set do not overlap with questions from the dev. and test sets. The resulting training data contains $34,100$ triples: its relevance label distribution

Model	MAP	MRR
LSTM	43.91	49.28
CNN	44.43	49.01
CNN Train	44.43	49.01
CNN Train + ED[3]	**44.77**	**52.07**

Table 8.2: Impact of CNN vs. LSTM sentence models on the baseline network for Task C.

is shown in the row, Train + ED, of Table 8.1. [2]

Pre-processing: we tokenized and put both questions and comments in lowercase. Moreover, we concatenated the question subject and body to create a unique question text. For computational reasons, we limited the document size to 100 words. This did not cause any degradation in accuracy.

Neural Networks: we mapped words to embeddings of size 50, pre-initializing them with standard skip-gram embeddings of dimensionality 50. The latter embeddings were trained on the English Wikipedia dump using word2vec toolkit [82]. We encoded the input sentence with a fixed-sized vector, whose dimensions are 100, using a convolutional operation of size five and a k-max pooling operation with $k = 1$. Table 8.2 shows the results of our preliminary experiments with the sentence models of CNN and LSTM, respectively, on the dev. set of Task C. In our further experiments, we opted for CNN since it produced a better MAP and is computationally more efficient.

For each MLP, we used a non-linear hidden layer (with hyperbolic tangent activation, Tanh), whose size is equal to the previous layer's size, i.e., 100. We included information such as word overlaps [130] and rank position as embeddings with an additional lookup table with vectors of size $d_{feat} = 5$. The rank feature is provided in the SemEval dataset and describes the position of the questions/comments in the search engine output.

Training: we trained our networks using SGD with shuffled mini-batches using the rmsprop update rule [127]. We set the training to iterate until the validation loss stops improving, with patience $p = 10$, i.e., the number of epochs to wait before early stopping, if no progress on the validation set is obtained.

We added dropout [118] between all the layers of the network to improve generalization and avoid co-adaptation of features. We tested different dropout rates (0.2, 0.4) for the inputs and (0.3, 0.5, 0.7) for the hidden layers obtaining better results with the highest values, i.e., 0.4 and 0.7.

8.2.2 Results

Table 8.3 shows the results of our individual and MTL models, in comparison with the Random and IR baselines of the challenge (first two rows), and the SemEval 2016 systems (rows 3–12).

[2] We make out MTL data available at
`http://ikernels-portal.disi.unitn.it/repository/`

[3] Extended Dataset for Task C computed using questions from Task A.

Model	DEV		TEST	
	MAP	MRR	MAP	MRR
Random	-	-	15.01	15.19
IR Baseline	-	-	40.36	45.83
SUper-team	-	-	55.41	61.48
KeLP	-	-	52.95	59.23
SemanticZ	-	-	51.68	55.96
MTE-NN	-	-	49.38	51.56
ICL00	-	-	49.19	53.89
SLS	-	-	49.09	55.98
ITNLP-AiKF	-	-	48.49	55.21
ConvKN	-	-	47.15	51.43
ECNU	-	-	46.47	51.41
UH-PRHLT	-	-	43.20	47.79
$\langle q_{new}, c_{rel} \rangle$	44.77	52.07	41.95	47.21
$\langle q_{new}, q_{rel}, c_{rel} \rangle$	45.59	51.04	46.99	55.64
$\langle q_{new}, q_{rel}, c_{rel} \rangle$ + $\leftrightarrow$	47.82	53.03	46.45	51.72
MTL (BC)	47.80	52.31	48.58	55.77
MTL (AC)	46.34	51.54	48.49	54.01
MTL (ABC)	**49.63**	55.47	**49.87**	55.73
MTL + one feature	-	-	**52.67**	55.68

Table 8.3: Results on the validation and test set for the proposed models

Rows 13-15 illustrate the results of our models when trained only on Task C. $\langle q_{new}, c_{rel} \rangle$ corresponds to the basic model, i.e., the single network, whereas the $\langle q_{new}, q_{rel}, c_{rel} \rangle$ model only exploits the joint input, i.e., the availability of q_{rel}. Rows 16-18 report the MTL models combining Task C with the other two tasks. The difference with the previous group (rows 13-15) is in the training phase, which is also operated on the instances from tasks A and B.

We note that: (i) the single network for Task C cannot compete with the challenge systems, as it would be ranked at the last position, according to the official MAP score (test set result); (ii) the joint representation, $\langle q_{new}, q_{rel}, c_{rel} \rangle$, highly improves the MAP of the basic network from 41.95 to 46.99 on the test set. This confirms the importance of having highly related tasks using input encoding closely related semantics. (iii) The shared sentence model for q_{new} and q_{rel} (indicated with $\leftrightarrow$) improves MAP on the dev. set only. (iv) The MTL (ABC) provides the best MAP, improving BC and AC by 1.29 and 1.38, respectively. Most importantly, it also improves, $\langle q_{new}, q_{rel}, c_{rel} \rangle$ by 2.88 points, i.e., it improves the best model using the joint representation and no training on the auxiliary tasks.

Additionally, our full MTL model would have ranked 4^{th} on Task C of the SemEval 2016 competition.

This is an essential result since all the challenge systems use many manually engineered features, whereas our model does not (except for the necessary initial rank). If we add the most powerful feature used by the top systems to our model, i.e., the weighted sum between the score of the task A classifier and the Google rank [80, 35], our system would achieve a

MAP of 52.67, i.e., very close to the second system.

Finally, we do not report the results of the additional tasks for lack of space and also because our idea of using MTL is to improve the target Task C. Indeed, by their definition, tasks A and B are simpler than C and are designed for solving it. Thus, attempting to improve the more straightforward A and B tasks by solving the more complex Task C, although interesting, looks less realistic. Indeed, we did not observe any important improvement of tasks A and B in our MTL setting.

8.3 Related Work

The work related to cQA spans two major areas: question and answer passage retrieval. Hereafter, we report some important research about them and then conclude with specific work on MTL.

Question-Question Similarity. Early approaches to question similarity used statistical machine translation techniques, e.g., Jeon, Croft, and Lee, Zhou et al. [51, 164], to measure the similarity between questions. Language models for question-question similarity were explored by Cao et al. [16], who incorporated information from the category structure of Yahoo! Answers when computing similarity between two questions. Instead, Duan et al. [30] proposed an approach that identifies the topic and focuses on questions, and computes their similarity.

Ji et al. [52] and Zhang et al. [163] learned a probability distribution over the topics that generate the question/answers pairs with LDA and used it to measure the similarity between questions. Recently, Da San Martino et al. [28] showed that effectively combining tree kernels (TKs) with text similarity features can improve the results over strong baselines such as Google.

Question-Answer Similarity: Yao et al. [153] used a conditional random field trained on a set of powerful features, such as tree-edit distance between the question and answer trees. Heilman and Smith [42] used a linear classifier exploiting syntactic features to solve different tasks such as recognizing textual entailment, paraphrases, and answer selection. Wang, Smith, and Mitamura [144] proposed Quasi-synchronous grammars to select short answers for TREC questions. Wang and Manning [142] used a probabilistic Tree-Edit model with structured latent variables for solving textual entailment and question answering. Severyn and Moschitti [112] proposed SVM with tree kernels to learn structural patterns between questions and answers encoded in the form of shallow syntactic parse trees, whereas, in Tymoshenko, Bonadiman, and Moschitti, Barrón-Cedeño et al. [131, 7] the author's used TKs and CNNs to rank comments in web forums, achieving state of the art on the SemEval cQA challenge. Wang and Nyberg [140] trained an extended short-term memory model for selecting answers to TREC questions.

Finally, a recent work close to ours is Guzmán, Màrquez, and Nakov [40], which build a neural network for solving Task A of SemEval. However, this does not approach the problem as MTL.

Related work on MTL: A good overview of MTL, i.e., learning to solve multiple tasks using a shared representation with mutual benefit, is given in [17]. Collobert and Weston [25] trained a convolutional NN with MTL, which, given an input sentence, could perform many sequence labeling tasks. They showed that jointly training their system on different tasks, such as speech tagging, named entity recognition, etc., significantly improves the performance on the main task, i.e., semantic role labeling, without requiring hand-engineered features.

Liu et al. [73] is the closest work to ours. They used multi-task deep neural networks to map queries and documents into semantic vector representations. This representation is later used in two tasks: query classification and question-answer reranking. Their results showed a competitive gain over strong baselines. Our work presented an architecture that can also exploit the joint representation of questions and comments, given the strong interdependencies among the different SemEval Tasks.

8.4 Summary

We proposed an MTL architecture for cQA, where we could exploit the structure of auxiliary tasks, which are highly semantically connected with our main task. This enabled using the same semantic representation to encode the text objects associated with all the three tasks, i.e., new questions, related questions, and comments. Our shared semantic representation provides an essential advantage over previous MTL applications, whose subtasks share a less consistent semantic representation.

Our experiments on the dataset of SemEval 2016 Task 3 show that our MTL approach relatively improves the individual DNNs by almost 20%. This is due to the shared representation and training on the instances of the two auxiliary tasks.

Part IV

Conclusion and Future Work

Chapter 9

Conclusion

In this thesis, we propose an efficient solution for the task of Answer Sentence Selection (A2S). This is achieved by exploring different aspects of the task: the local structure of question and answer pairs, and the global structure. We refer to *local structure* as the syntactic structure of the question and the answer candidate, and the inter-sentence relational structure among the constituent of both sentences. In contrast, *global structure* includes features that are external to individual question-candidate pairs — for example, the structure of the original document that contains the answer candidate in context.

To this end, (i) in Chapter 4, we analyzed the differences between fast CNN architectures and Structural Tree Kernel approaches at modeling syntactic and relational information we proposed a solution for combining the two approaches obtaining competitive results on A2S. The result suggests that the two machine learning approaches learn different types of features from the input text. Capitalizing on this result, (ii) in Chapter 5, we used a teacher-student approach to inject syntactic and relational information of CTK architecture in an efficient Relational Neural Network. The resulting CNN model outperforms both the individual architectures. This result has multiple implications: it shows that we can capitalize on the fact that CTK architecture provides competitive results in a data scarcity regime, and that CNN architectures can implicitly encode the structural information of Tree-Kernels. (iii) Chapter 6 proposes a novel neural network architecture, the Cosinenet, that capitalize on ad-hoc retrofitted relational word embeddings to improve the capabilities of the network to encode inter-sentence relational structure. The proposed model achieves performances that are comparable with attention-based architectures while maintaining a limited number of parameters.

Finally, in Chapter 7, (iv) we presented a solution for encoding the document structure in A2S architecture. In particular, we propose extending standard point-wise A2S architectures with a component capable of encoding the global structure, i.e., a biRNN that takes as input all the question-candidate representations for the same question. We trained the model using list-wise learning to rank strategy obtaining consistent improvements over the point-wise counterparts. Moreover, we show that this simple modification to the architecture can lead to consistent improvements when combined with efficient architectures with little increase in

the parameters. In particular, we show that adding the document structure and listwise training to the Cosinenet architecture of Section 6 greatly increases its accuracy, outperforming all the other efficient approaches to the task. Additionally, we show that the proposed model can process up to 700 questions per second during training on a single GPU. In perspective, the more general architecture of BERT can process just six questions per second on the same GPU.

Additionally, in Chapter 8, we studied one aspect of the Global structure: the structure of the task of community Question Answering (cQA). In cQA, short text reranker, that are the basic models for A2S, can be applied to three different sub-tasks: (A) selecting the best answer to a user question from a thread in a forum, (B) selecting a relevant question to a new question, and (C) identifying the best answer to the new question. We show that not only task (C) is logically derived by the output for tasks (B) and (A), but that task (A) is semantically connected to the task (C) as it learns similar features. Therefore, we presented a novel architecture for task C, based on Relational CNNs, that learns to solve all the three sub-tasks simultaneously in a multi-task learning framework, effectively – and efficiently – encoding the task structure of cQA.

9.1 Future Work

In this thesis, we show that creating specialized architecture for the task of A2S is a fundamental step to achieve competitive results without exponentially increase the computational cost of the model. Although we proposed a model that outperforms all other efficient models for A2S, pretrained transformer architectures, such as BERT, are still unmatched. Moreover, BERT does not use any global structure, relying on features extracted by individual QA pairs.

In the future, we plan to extend BERT with global structural information. However, this advancement is currently limited by the current generation of GPUs, as BERT requires a considerable amount of memory during training, and it cannot process all the answer candidates for a single question at the same time.

On the other end, we plan to capitalize on the novelties introduced by the BERT architecture to improve efficient Neural Network architectures' performances further. We want to explore the possibility of pretraining the Cosinenet on related (possibly unsupervised) tasks. In particular, we would like to use unsupervised tasks for pretraining the neural network component that captures the document structure of sentences in the original document.

Bibliography

[1] Eugene Agichtein et al. “Overview of the TREC 2015 LiveQA Track”. In: *TREC*. 2015.

[2] Lloyd Allison and Trevor Dix. “A Bit-string Longest-common-subsequence Algorithm”. In: *Inf. Process. Lett.* 23.6 (Dec. 1986), pp. 305–310. ISSN: 0020-0190. URL: http://dl.acm.org/citation.cfm?id=8871.8877.

[3] Sören Auer et al. “Dbpedia: A nucleus for a web of open data”. In: *The semantic web*. Springer, 2007, pp. 722–735.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[5] Payal Bajaj et al. “MS MARCO: A human generated MAchine Reading COmprehension dataset”. In: *arXiv preprint arXiv:1611.09268* (2016).

[6] Alberto Barrón-Cedeño et al. “ConvKN at SemEval-2016 Task 3: Answer and Question Selection for Question Answering on Arabic and English Fora”. In: *Proceedings of the 10th International Workshop on Semantic Evaluation*. SemEval ’16. San Diego, California, USA, 2016, pp. 896–903.

[7] Alberto Barrón-Cedeño et al. “ConvKN at SemEval-2016 Task 3: Answer and question selection for question answering on Arabic and English fora”. In: *Proceedings of SemEval* (2016), pp. 896–903.

[8] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[9] Jonathan Berant et al. “Semantic Parsing on Freebase from Question-Answer Pairs”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1533–1544. URL: https://www.aclweb.org/anthology/D13-1160.

[10] Weijie Bian et al. “A compare-aggregate model with dynamic-clip attention for answer selection”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM. 2017, pp. 1987–1990.

[11] Avrim Blum and Tom Mitchell. “Combining labeled and unlabeled data with co-training”. In: *Proceedings of the eleventh annual conference on Computational learning theory*. ACM. 1998, pp. 92–100.

[12] Kurt Bollacker et al. “Freebase: a collaboratively created graph database for structuring human knowledge”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. AcM. 2008, pp. 1247–1250.

[13] Daniele Bonadiman, Antonio Uva, and Alessandro Moschitti. “Effective shared representations with multitask learning for community question answering”. In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. 2017, pp. 726–732.

[14] Antoine Bordes, Sumit Chopra, and Jason Weston. “Question Answering with Subgraph Embeddings”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 615–620.

[15] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT’2010*. Springer, 2010, pp. 177–186.

[16] Xin Cao et al. “The use of categorization information in language models for question retrieval”. In: *CIKM*. 2009.

[17] Rich Caruana. “Multitask learning”. In: *Machine learning* 28.1 (1997), pp. 41–75.

[18] Alessandra Cervone et al. “Natural Language Generation at Scale: A Case Study for Open Domain Question Answering”. In: *arXiv preprint arXiv:1903.08097* (2019).

[19] Danqi Chen et al. “Reading Wikipedia to Answer Open-Domain Questions”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 1870–1879.

[20] Qian Chen et al. “Enhanced LSTM for Natural Language Inference”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 1657–1668.

[21] Ruey-Cheng Chen et al. “On the benefit of incorporating external features in a neural architecture for answer sentence selection”. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2017, pp. 1017–1020.

[22] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a similarity metric discriminatively, with application to face verification”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 539–546.

[23] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).

[24] Daniel Cohen and W Bruce Croft. “End to end long short term memory networks for non-factoid question answering”. In: *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*. 2016, pp. 143–146.

[25] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 160–167.

[26] Ronan Collobert et al. “Natural language processing (almost) from scratch”. In: *The Journal of Machine Learning Research* 12 (2011), pp. 2493–2537.

[27] Danilo Croce and Roberto Basili. “Large-Scale Kernel-Based Language Learning Through

the Ensemble Nystr

\ddot {o}

$$mMethods$$

". In: *European Conference on Information Retrieval*. Springer. 2016, pp. 100–112.

[28] Giovanni Da San Martino et al. "Learning to re-rank questions in community question answering using advanced features". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM. 2016, pp. 1997–2000.

[29] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 4171–4186.

[30] Huizhong Duan et al. "Searching Questions by Identifying Question Topic and Question Focus." In: *ACL*. 2008.

[31] Jeffrey L Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211.

[32] Manaal Faruqui et al. "Problems with evaluation of word embeddings using word similarity tasks". In: *arXiv preprint arXiv:1605.02276* (2016).

[33] Manaal Faruqui et al. "Retrofitting Word Vectors to Semantic Lexicons." In: *CoRR* (2014). URL: http://dblp.uni-trier.de/db/journals/corr/corr1411.html#FaruquiDJDHS14.

[34] Simone Filice et al. "KeLP at SemEval-2016 Task 3: Learning Semantic Relations between Questions and Answers". In: *Proceedings of the Workshop on Semantic Evaluation*. SemEval '16. San Diego, California, USA, 2016, pp. 1116–1123.

[35] Simone Filice et al. "KeLP at SemEval-2016 Task 3: Learning semantic relations between questions and answers". In: *Proceedings of SemEval* 16 (2016), pp. 1116–1123.

[36] Siddhant Garg, Thuy Vu, and Alessandro Moschitti. "TANDA: Transfer and Adapt Pre-Trained Transformer Models for Answer Sentence Selection". In: *CoRR* abs/1911.04118 (2019). arXiv: 1911.04118. URL: http://arxiv.org/abs/1911.04118.

[37] Yoav Goldberg. "A primer on neural network models for natural language processing". In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 345–420.

[38] Ian J Goodfellow et al. "An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks". In: *stat* 1050 (2014), p. 6.

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[40] Francisco Guzmán, Lluís Màrquez, and Preslav Nakov. "Machine Translation Evaluation Meets Community Question Answering". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 460–466. URL: http://anthology.aclweb.org/P16-2075.

[41] Hua He, Kevin Gimpel, and Jimmy 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.

[42] Michael Heilman and Noah A Smith. "Tree edit models for recognizing textual entailments, paraphrases, and answers to questions". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2010, pp. 1011–1019.

[43] Michael Heilman and Noah A. Smith. "Tree Edit Models for Recognizing Textual Entailments, Paraphrases, and Answers to Questions". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. HLT '10. Los Angeles, California, USA, 2010, pp. 1011–1019. ISBN: 1-932432-65-5. URL: http://dl.acm.org/citation.cfm?id=1857999.1858143.

[44] William Hersh. *Information retrieval: a health and biomedical perspective*. Springer Science & Business Media, 2008.

[45] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[46] Jeremy Howard and Sebastian Ruder. "Universal language model fine-tuning for text classification". In: *arXiv preprint arXiv:1801.06146* (2018).

[47] Baotian Hu et al. "Convolutional neural network architectures for matching natural language sentences". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2042–2050.

[48] Zhiting Hu et al. "Harnessing Deep Neural Networks with Logic Rules". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 2410–2420. DOI: 10.18653/v1/P16-1228. URL: https://www.aclweb.org/anthology/P16-1228.

[49] Mohit Iyyer et al. "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: *Association for Computational Linguistics*. Beijing, China, 2015.

[50] Paul Jaccard. "Étude comparative de la distribution florale dans une portion des Alpes et des Jura". In: *Bulletin del la Société Vaudoise des Sciences Naturelles* (1901).

[51] Jiwoon Jeon, W Bruce Croft, and Joon Ho Lee. "Finding similar questions in large question and answer archives". In: *CIKM*. 2005.

[52] Zongcheng Ji et al. "Question-answer topic model for question retrieval in community question answering". In: *CIKM*. 2012.

[53] Thorsten Joachims. "Optimizing search engines using clickthrough data". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2002, pp. 133–142.

[54] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A Convolutional Neural Network for Modelling Sentences". In: *ACL* (2014).

[55] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. "A convolutional neural network for modelling sentences". In: *arXiv preprint arXiv:1404.2188* (2014).

[56] Andrej Karpathy et al. "Large-scale video classification with convolutional neural networks". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732.

[57] Douwe Kiela, Felix Hill, and Stephen Clark. "Specializing word embeddings for similarity or relatedness". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 2044–2048.

[58] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: Doha, Qatar, 2014.

[59] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *EMNLP*. Doha, Qatar, Oct. 2014, pp. 1746–1751.

[60] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[61] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[62] Bernhard Kratzwald and Stefan Feuerriegel. "Adaptive Document Retrieval for Deep Question Answering". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 576–581.

[63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[64] Tom Kwiatkowski et al. "Natural questions: a benchmark for question answering research". In: *Transactions of the Association for Computational Linguistics* 7 (2019), pp. 453–466.

[65] Tom Kwiatkowski et al. "Natural Questions: a Benchmark for Question Answering Research". In: *Transactions of the Association of Computational Linguistics* (2019).

[66] Tuan Lai et al. "A Gated Self-attention Memory Network for Answer Selection". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5953–5959. DOI: 10.18653/v1/D19-1610. URL: https://www.aclweb.org/anthology/D19-1610.

[67] Tuan Lai et al. "A Gated Self-attention Memory Network for Answer Selection". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 5955–5961.

[68] Amy N Langville and Carl D Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2011.

[69] Steve Lawrence et al. "Face recognition: A convolutional neural-network approach". In: *Neural Networks, IEEE Transactions on* 8.1 (1997), pp. 98–113.

[70] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[71] Xin Li and Dan Roth. "Learning question classifiers". In: *COLING*. 2002.

[72] Hugo Liu and Push Singh. "ConceptNet—a practical commonsense reasoning tool-kit". In: *BT technology journal* 22.4 (2004), pp. 211–226.

[73] Xiaodong Liu et al. "Representation learning using multi-task deep neural networks for semantic classification and information retrieval". In: *Proc. NAACL*. 2015.

[74] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[75] Zhengdong Lu and Hang Li. “A Deep Architecture for Matching Short Texts”. In: *NIPS*. 2013.

[76] Caroline Lyon, James Malcolm, and Bob Dickerson. “Detecting short passages of similar text in large document collections”. In: EMNLP. 2001.

[77] Christopher D. Manning et al. “The Stanford CoreNLP Natural Language Processing Toolkit”. In: *Association for Computational Linguistics (ACL) System Demonstrations*. 2014, pp. 55–60. URL: http://www.aclweb.org/anthology/P/P14/P14-5010.

[78] Yishu Miao, Lei Yu, and Phil Blunsom. “Neural Variational Inference for Text Processing”. In: *arXiv preprint arXiv:1511.06038* (2015).

[79] Tsvetomila Mihaylova et al. “SUper Team at SemEval-2016 Task 3: Building a Feature-Rich System for Community Question Answering”. In: *Proceedings of the Workshop on Semantic Evaluation*. SemEval ’16. San Diego, California, USA, 2016, pp. 836–843.

[80] Tsvetomila Mihaylova et al. “SUper Team at SemEval-2016 Task 3: Building a Feature-Rich System for Community Question Answering”. In: *SemEval@NAACL-HLT*. 2016.

[81] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic Regularities in Continuous Space Word Representations”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. NAACL-HLT ’13. Atlanta, GA, USA, 2013, pp. 746–751. URL: http://www.aclweb.org/anthology/N13-1090.

[82] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems 26*. Ed. by C.J.C. Burges et al. Curran Associates, Inc., 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[83] Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. “Question Answering through Transfer Learning from Large Fine-grained Supervision Data”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2017, pp. 510–517.

[84] Alessandro Moschitti. “Efficient convolution kernels for dependency and constituent syntactic trees”. In: *European Conference on Machine Learning*. Springer. 2006, pp. 318–329.

[85] Alessandro Moschitti, Daniele Pighin, and Roberto Basili. “Tree kernels for semantic role labeling”. In: *Computational Linguistics* 34.2 (2008), pp. 193–224.

[86] Alessandro Moschitti et al. “Exploiting Syntactic and Shallow Semantic Kernels for Question Answer Classification”. In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. ACL ’07. Prague, Czech Republic, 2007, pp. 776–783. URL: http://www.aclweb.org/anthology/P07-1098.

[87] Alessandro Moschitti et al. “Exploiting syntactic and shallow semantic kernels for question answer classification”. In: *ANNUAL MEETING-ASSOCIATION FOR COMPU-*

TATIONAL LINGUISTICS. Vol. 45. 1. 2007, p. 776. URL: http://acl.ldc.upenn.edu/P/P07/P07-1098.pdf.

[88] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[89] Preslav Nakov et al. "SemEval-2016 Task 3: Community Question Answering". In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. 2016, pp. 525–545.

[90] Preslav Nakov et al. "SemEval-2016 Task 3: Community Question Answering". In: *Proceedings of the 10th International Workshop on Semantic Evaluation*. SemEval '16. San Diego, California: Association for Computational Linguistics, June 2016.

[91] Massimo Nicosia et al. "QCRI: Answer Selection for Community Question Answering - Experiments for Arabic and English". In: *Proceedings of the 9th International Workshop on Semantic Evaluation*. SemEval '15. Denver, Colorado, USA, 2015. URL: http://www.aclweb.org/anthology/S15-2026.

[92] Kyosuke Nishida et al. "Multi-style Generative Reading Comprehension". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 2273–2284. DOI: 10.18653/v1/P19-1220. URL: https://www.aclweb.org/anthology/P19-1220.

[93] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[94] Matthew E. Peters et al. "Deep contextualized word representations". In: *Proc. of NAACL*. 2018.

[95] V. Punyakanok and D. Roth. "The Use of Classifiers in Sequential Inference". In: *NIPS*. MIT Press, 2001, pp. 995–1001. URL: http://cogcomp.cs.illinois.edu/papers/nips01.pdf.

[96] Silvia Quarteroni and Suresh Manandhar. "Designing an interactive open-domain question answering system". In: *Natural Language Engineering* 15.01 (2009), pp. 73–95.

[97] Alec Radford et al. "Language models are unsupervised multitask learners". In: ().

[98] Pranav Rajpurkar et al. "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250* (2016).

[99] Jinfeng Rao, Hua He, and Jimmy Lin. "Noise-contrastive estimation for answer selection with deep neural networks". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM. 2016, pp. 1913–1916.

[100] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[101] Cıcero Nogueira dos Santos et al. "Attentive pooling networks". In: *CoRR, abs/1602.03609* (2016).

[102] Tobias Schnabel et al. "Evaluation methods for unsupervised word embeddings." In: *EMNLP*. 2015, pp. 298–307.

[103] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[104] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. "Introduction to information retrieval". In: *Proceedings of the international communication of association for computing machinery conference*. Vol. 4. 2008.

[105] Royal Sequiera et al. "Exploring the effectiveness of convolutional neural networks for answer selection in end-to-end question answering". In: *arXiv preprint arXiv:1707.07804* (2017).

[106] Aliaksei Severyn. "Modelling input texts: from Tree Kernels to Deep Learning". PhD thesis. University of Trento, 2015.

[107] Aliaksei Severyn and Alessandro Moschitti. "Automatic Feature Engineering for Answer Selection and Extraction". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 458–467. URL: https://www.aclweb.org/anthology/D13-1044.

[108] Aliaksei Severyn and Alessandro Moschitti. "Automatic Feature Engineering for Answer Selection and Extraction". In: *EMNLP*. 2013.

[109] Aliaksei Severyn and Alessandro 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.

[110] Aliaksei Severyn and Alessandro Moschitti. "Modeling Relational Information in Question-Answer Pairs with Convolutional Neural Networks". In: *In Preprint arXiv:1604.01178* (2016).

[111] Aliaksei Severyn and Alessandro Moschitti. "Modeling relational information in question-answer pairs with convolutional neural networks". In: *arXiv preprint arXiv:1604.01178* (2016).

[112] Aliaksei Severyn and Alessandro Moschitti. "Structural relationships for large-scale learning of answer re-ranking". In: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2012, pp. 741–750.

[113] Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. "Learning Adaptable Patterns for Passage Reranking". In: *CoNLL-2013* (2013), p. 75. URL: http://acl.eldoc.ub.rug.nl/mirror/W/W13/W13-35.pdf#page=87.

[114] Lei Sha et al. "A Multi-View Fusion Neural Network for Answer Selection." In: *AAAI*. 2018.

[115] Gehui Shen, Yunlun Yang, and Zhi-Hong 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.

[116] Libin Shen and Aravind Joshi. "An SVM-based voting algorithm with application to parse reranking". In: *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*. 2003, pp. 9–16.

[117] Robert Speer and Joanna Lowry-Duda. "ConceptNet at SemEval-2017 Task 2: Extending Word Embeddings with Multilingual Relational Knowledge". In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. 2017, pp. 85–89.

[118] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[119] Andreas Stolcke et al. "Dialogue act modeling for automatic tagging and recognition of conversational speech". In: *Computational linguistics* 26.3 (2000), pp. 339–373.

[120] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *arXiv preprint arXiv:1906.02243* (2019).

[121] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. "YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia". In: ().

[122] Cheng-Lung Sung et al. "An alignment-based surface pattern for a question answering system". In: *2008 IEEE International Conference on Information Reuse and Integration*. IEEE. 2008, pp. 172–177.

[123] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. "Learning to Rank Answers on Large Online QA Collections". In: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics and the Human Language Technology Conference*. ACL-HLT '08. Columbus, Ohio, USA, 2008, pp. 719–727. URL: http://aclweb.org/anthology/P08-1082.

[124] Ming Tan et al. "Lstm-based deep learning models for non-factoid answer selection". In: *arXiv preprint arXiv:1511.04108* (2015).

[125] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. "Hyperbolic Representation Learning for Fast and Efficient Neural Question Answering". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM. 2018, pp. 583–591.

[126] Yi Tay et al. "Learning to rank question answer pairs with holographic dual lstm architecture". In: *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*. 2017, pp. 695–704.

[127] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural Networks for Machine Learning* 4 (2012).

[128] Quan Hung Tran et al. "The Context-dependent Additive Recurrent Neural Net". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Vol. 1. 2018, pp. 1274–1283.

[129] K. Tymoshenko and A. Moschitti. "Assessing the Impact of Syntactic and Semantic Structures for Answer Passages Reranking". In: *CIKM*. 2015.

[130] Kateryna Tymoshenko, Daniele Bonadiman, and Alessandro Moschitti. "Convolutional neural networks vs. convolution kernels: Feature engineering for answer sentence reranking". In: *Proceedings of NAACL-HLT*. 2016, pp. 1268–1278.

[131] Kateryna Tymoshenko, Daniele Bonadiman, and Alessandro Moschitti. "Learning to Rank Non-Factoid Answers: Comment Selection in Web Forums". In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM. 2016, pp. 2049–2052.

[132] Kateryna Tymoshenko, Daniele Bonadiman, and Alessandro Moschitti. "Ranking kernels for structures and embeddings: A hybrid preference and classification model". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017, pp. 897–902.

[133] Kateryna Tymoshenko and Alessandro Moschitti. "Assessing the Impact of Syntactic and Semantic Structures for Answer Passages Reranking". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19 - 23, 2015*. 2015, pp. 1451–1460.

[134] Christina Unger et al. "Template-based question answering over RDF data". In: *Proceedings of the 21st international conference on World Wide Web*. ACM. 2012, pp. 639–648.

[135] Antonio Uva, Daniele Bonadiman, and Alessandro Moschitti. "Injecting Relational Structural Representation in Neural Networks for Question Similarity". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2018, pp. 285–291.

[136] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[137] Alex Wang et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 2018, pp. 353–355.

[138] Bingning Wang, Kang Liu, and Jun Zhao. "Inner attention based recurrent neural networks for answer selection". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016, pp. 1288–1297.

[139] Di Wang and Eric Nyberg. "A Long Short-Term Memory Model for Answer Sentence Selection in Question Answering". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*. 2015, pp. 707–712. URL: https://www.aclweb.org/anthology/P15-2116/.

[140] Di Wang and Eric Nyberg. "A long short-term memory model for answer sentence selection in question answering". In: *ACL, July* (2015).

[141] Kai Wang, Zhaoyan Ming, and Tat-Seng Chua. "A syntactic tree matching approach to finding similar questions in community-based qa services". In: *SIGIR*. 2009.

[142] Mengqiu Wang and Christopher 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*. COLING '10. Beijing, China, 2010, pp. 1164–1172. URL: http://dl.acm.org/citation.cfm?id=1873781.1873912.

[143] Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. "What is the Jeopardy Model? A Quasi-Synchronous Grammar for QA". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. EMNLP-CoNLL '07. Prague, Czech Republic, 2007, pp. 22–32. URL: http://www.aclweb.org/anthology/D/D07/D07-1003.

[144] Mengqiu Wang, Noah A Smith, and Teruko Mitamura. "What is the Jeopardy model? A quasi-synchronous grammar for QA". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 2007.

[145] Shuohang Wang and Jing Jiang. "A compare-aggregate model for matching text sequences". In: *arXiv preprint arXiv:1611.01747* (2016).

[146] Zhiguo Wang, Wael Hamza, and Radu Florian. "Bilateral multi-perspective matching for natural language sentences". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. AAAI Press. 2017, pp. 4144–4150.

[147] Zhiguo Wang and Abraham Ittycheriah. "FAQ-based Question Answering via Word Alignment". In: *arXiv preprint arXiv:1507.02628* (2015).

[148] Michael Wise. "YAP3: Improved Detection of Similarities in Computer Program and Other Texts". In: *SIGCSE*. 1996.

[149] Yi Yang, Wen-tau Yih, and Christopher 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.

[150] Zhilin Yang et al. "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 2369–2380.

[151] Zhilin Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *arXiv preprint arXiv:1906.08237* (2019).

[152] Xuchen Yao et al. "Answer Extraction as Sequence Tagging with Tree Edit Distance". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. NAACL-HLT '13. Atlanta, Georgia, USA, 2013, pp. 858–867. URL: http://aclweb.org/anthology/N13-1106.

[153] Xuchen Yao et al. "Answer Extraction as Sequence Tagging with Tree Edit Distance." In: *HLT-NAACL*. Citeseer. 2013, pp. 858–867.

[154] Wen-Tau Yih, Xiaodong He, and Christopher Meek. "Semantic Parsing for Single-Relation Question Answering". In: *ACL*. 2014.

[155] Wen-tau Yih et al. "Question Answering Using Enhanced Lexical Semantic Models". In: *ACL*. Aug. 2013.

[156] Wen-tau Yih et al. "Question Answering Using Enhanced Lexical Semantic Models". In: *ACL*. Sofia, Bulgaria, Aug. 2013, pp. 1744–1753.

[157] Wenpeng Yin et al. "ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs". In: *arXiv preprint arXiv:1512.05193* (2015).

[158] Wenpeng Yin et al. "ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs". In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 259–272.

[159] Seunghyun Yoon, Joongbo Shin, and Kyomin Jung. "Learning to Rank Question-Answer Pairs using Hierarchical Recurrent Encoder with Latent Topic Clustering". In: *Proceedings of NAACL-HLT*. 2018, pp. 1575–1584.

[160] Lei Yu et al. "Deep Learning for Answer Sentence Selection". In: *CoRR* (2014).

[161] Lei Yu et al. “Deep learning for answer sentence selection”. In: *arXiv preprint arXiv:1412.1632* (2014).
[162] Matthew D. Zeiler. “ADADELTA: An Adaptive Learning Rate Method”. In: *CoRR* (2012).
[163] Kai Zhang et al. “Question retrieval with high quality answers in community question answering”. In: *CIKM*. 2014.
[164] Guangyou Zhou et al. “Phrase-based translation model for question retrieval in community question answer archives”. In: *ACL*. 2011.