4-2019

Question answering with textual sequence matching

Shuohang WANG
Singapore Management University, shwang.2014@phdis.smu.edu.sg

QUESTION ANSWERING WITH TEXTUAL SEQUENCE MATCHING

SHUOHANG WANG

SINGAPORE MANAGEMENT UNIVERSITY
2019

Question Answering with Textual Sequence Matching

by

Shuohang WANG

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Information Systems

Dissertation Committee:

Jing Jiang (Supervisor / Chair)
Associate Professor of Information Systems
Singapore Management University

Baihua Zheng
Associate Professor of Information Systems
Singapore Management University

David Lo
Associate Professor of Information Systems
Singapore Management University

Wei Lu
Assistant Professor of Information Systems Technology and Design
Singapore University of Technology and Design

Singapore Management University
2019

I hereby declare that this PhD dissertation is my original work and it has been written by me in its entirety.
I have duly acknowledged all the sources of information which have been used in this dissertation.

This PhD dissertation has also not been submitted for any degree in any university previously.

Shuohang Wang
16 April 2019

Question Answering with Textual Sequence Matching

Shuohang Wang

Abstract

Question answering (QA) is one of the most important applications in natural language processing. With the explosive text data from the Internet, intelligently getting answers of questions will help humans more efficiently collect useful information. My research in this thesis mainly focuses on solving question answering problem with textual sequence matching model which is to build vectorized representations for pairs of text sequences to enable better reasoning. And our thesis consists of three major parts.

In Part I, we propose two general models for building vectorized representations over a pair of sentences, which can be directly used to solve the tasks of answer selection, natural language inference, etc.. In Chapter 3, we propose a model named "match-LSTM", which performs word-by-word matching followed by a LSTM to place more emphasis on important word-level matching representations. On the Stanford Natural Language Inference (SNLI) [7] corpus, our model achieved the state of the art. Next in Chapter 4, we present a general "compare-aggregate" framework that performs word-level matching followed by aggregation using Convolutional Neural Networks. We focus on exploring 6 different comparison functions we can use for word-level matching, and find that some simple comparison functions based on element-wise operations work better than standard neural network and neural tensor network based comparison.

In Part II, we make use of the sequence matching model to address the task of machine reading comprehension, where the models need to answer the question based on a specific passage. In Chapter 5, we explore the power of

word-level matching for better locating the answer span from the given passage for each question in the task of machine reading comprehension. We propose an end-to-end neural architecture for the task. The architecture is based on match-LSTM and Pointer Net which constrains the output tokens coming from the given passage. We further propose two ways of using Pointer Net for our tasks. Our experiments show that both of our two models substantially outperform the best result [61] using logistic regression and manually crafted features. Besides, our boundary model also achieved the best performance on the SQuAD [61] and MSMARCO [57] dataset. In Chapter 6, we will explore another challenging task, multi-choice reading comprehension, where several candidate answers are also given besides the question related passage. We propose a new *co-matching* approach to this problem, which jointly models whether a passage can match both a question and a candidate answer.

In Part III, we focus on solving the problem of open-domain question answering, where no specific passage is given any more comparing to the reading comprehension task. Our models for solving this problem still rely on the textual sequence matching model to build ranking and reading comprehension models. In Chapter 7, we present a novel open-domain QA system called *Reinforced Ranker-Reader* (R^3), which jointly trains the *Ranker* along with an answer-extraction *Reader* model, based on reinforcement learning. We report extensive experimental results showing that our method significantly improves on the state of the art for multiple open-domain QA datasets. As this work can only make use of a single retrieved passage to answer the question, in the next Chapter 8, we propose two models, *strength-based* re-ranking and *coverage-based* re-ranking, which make use of multiple passages to generate their answers. Our models have achieved state-of-the-art results on three public open-domain QA datasets: Quasar-T [23], SearchQA [24] and the open-domain version of TriviaQA [40], with about 8 percentage points of improvement over the former two datasets.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Question answering (QA) [62, 34, 61, 10, 40, 1, 5] is one of the most important applications in natural language processing. The task is to build machines that can answer questions expressed in human language using information and knowledge found in sources such as a single article, a corpus of documents, a knowledge base, or even an image or a video. Intelligently solving this problem can not only help humans obtain useful information from huge amount of textual resources more efficiently, but also help develop intelligent machines such as conversational agents. Question answering has drawn much attention in recent years with the development of deep learning techniques. There're multiple QA tasks in different settings, such as answer selection (AS) [73], machine reading comprehension (MRC) [61], knowledge base question answering (KBQA) [5], visual question answering (VQA) [1], open-domain question answering (OQA) [10], etc.. All these tasks need to generate an answer for a given question, while the differences lie in the different context information given to answer the question. For example, the context information can be several candidate answers in the AS task, a question-related passage in MRC task, a knowledge base such as Freebase in the KBQA task, a question-related

Task :	**I. Answer Selection task**
Context:	(a) Yes, it be possible have auto insurance without own a vehicle. You will purchase what be call a name ... (b)Insurance not be a tax or merely a legal obligation because auto insurance follow a car... (c) You shall have auto insurance any time you own a car ...
Question:	Can I have auto insurance without a car?
Answer:	(a)
Task :	**II. Machine Reading Comprehension task**
Context:	In 1870, Tesla moved to Karlovac, **to attend school at the Higher Real Gymnasium**, where he was profoundly influenced by a math teacher Martin Sekulić. The classes were held in German, as it was a school within the Austro-Hungarian Military Frontier. Tesla was able to perform integral calculus in his head, which prompted his teachers to believe that he was cheating. He finished a four-year term in three years, graduating in 1873.
Question:	Why did Tesla go to Karlovac?
Answer:	attend school at the Higher Real Gymnasium
Task :	**III. Open-domain Question Answering task**
Context:	whole *Wikipedia* or any Web page
Question:	What is the largest island in the Philippines?
answer:	Luzon

Table 1.1: Examples for three different question answering tasks.

figure in the VQA task, or any information from the webs in the OQA task. The most challenging is arguably the the open-domain question answering task where only a question is given and we can actually make use of any resource to answer it. It's likely the final goal of question answering systems, which can answer anything using existing knowledge in the world. In this thesis, we will mainly focus on three tasks, namely Answer Selection, Machine Reading Comprehension and Open-domain Question Answering, which rely only on context information in raw text, as shown in Table 1.1.

In earlier days, people focused more on the task of answer selection, which relies on a sequence matching model to identify which context sequence can answer the question, as the first example shown in Table 1.1. Actually, there're also many other tasks relying on sequence matching model, such as paraphrase identification, natural language inference, etc.. Some classical methods [82]

train a matching model, like SVM, based on human crafted features, such as Rouge scores, BLEU scores, etc., to identify how well the sequence pairs can be matched. And the candidate sequence that can receive a higher matching score with the question would be predicted as the answer. However, these features, such as BLEU scores, are based on the exact N-gram match, and lack deep semantics (e.g., they fail to recognize that "dog" and "animal" are related.), lack consideration of context information (e.g. "dog" and hot "dog" are different), lack the relations between the matched phrases (e.g. "dog" can be entailed by "animal" and contradicted by "cat"). Later on, the sequences are represented by vectorized representations through neural networks, and the similarity between the representations is used to show how well the sequences are matched. Although with the help of some general neural frameworks, such as LSTM and CNN, the model achieved a good performance on answer selection tasks, the sequence level representations built by LSTM and CNN are still not powerful enough to represent all the semantic meanings of a sequence. We still need more interactive representations between sequences. In this thesis, we will propose a new textual sequence matching framework, which will build the phrase level matching representation between sequences through attention mechanism, and the aggregation of all the phrase level matching representations will represent how well the sequences are matched. Our models are designed to overcome the shortcomings of previous works as discussed above, and the experiment results shown that our models can achieve state-of-the-art performance on multiple answer selection tasks. Besides the answer selection tasks, our sequence matching framework is also one of the key component in the solutions to many other question answering tasks.

One of the most well-studied question answering tasks in recent years is machine reading comprehension. After I designed the aforementioned sequence matching model, a natural question is whether this model can be applied to machine reading comprehension. Specifically, in machine reading comprehension,

question related passage is given as context, and we can answer the question based on the information from the passage. The example in Table 1.1 is one setting of machine reading comprehension where we can directly extract the answer phrase from the context. Previous methods [61] on this task use human crafted feature based methods. They first extract all the noun and verb phrases from the passages by a parser, and then rank them with their corresponding contextual features. However, human crafted features are always not powerful enough to represent how the question and the context are matched in deep semantics. In this thesis, we propose the first neural network structure which can extract the answer in sequence for the question. Specifically, we first make use of our previously proposed sequence matching model to build the matching representation for every word in the passage, so that the matching representations can reflect how well the contextual information of the corresponding word can match the question and also reflect the probability of the word to be the answer. Then we further add a pointer [79] layer to point out the positions of the words that can compose the answer based on the matching presentations. According to the experiment result, our model can achieve a much better performance than human crafted feature based methods. This kind of framework has also been widely adopted in other models [66, 97, 10] for the machine reading comprehension tasks, and also the tasks of open-domain QA.

Another well-studied question answering tasks that is more challenging than machine reading comprehension is open-domain question answering, and I further explore how my previous work can be applied in this problem setting. In this setting, there's no specific context given any more. We can get the answer from any resource, while in this thesis, we mainly focus on making use of the raw text from Wikipedia to extract the answer. Previous work [10] on this task follows a pipeline work, "search and read". Specifically, they first make use of information retrieval (IR) models based on tf-idf values to retrieve the question-related passages from Wikipedia, and then train a reading

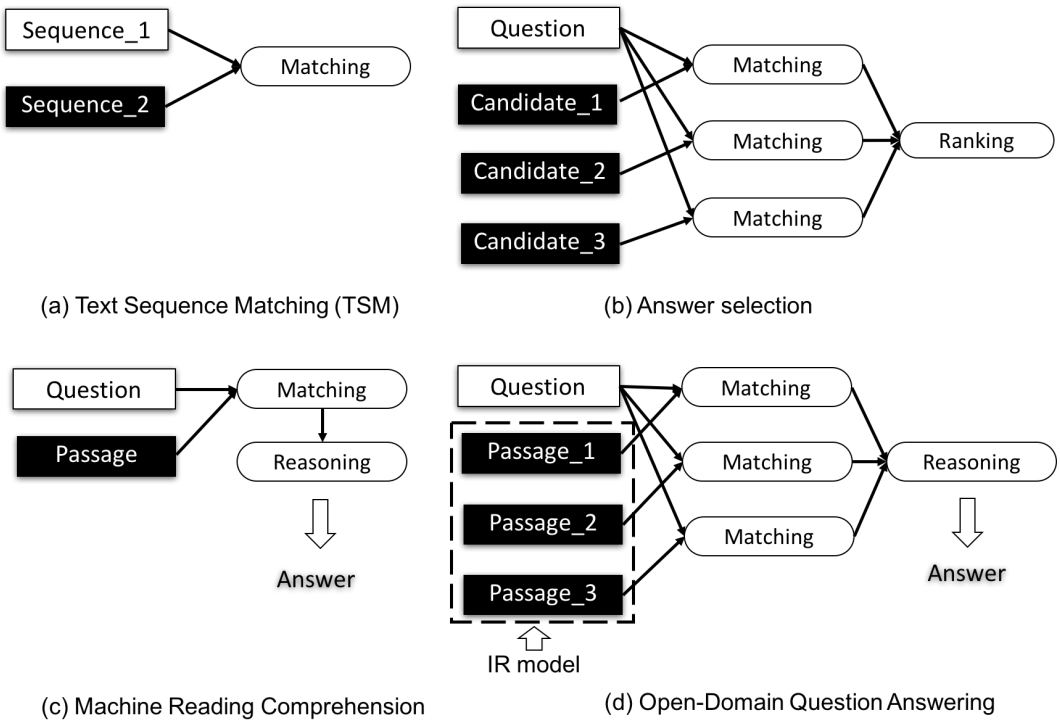

Figure 1.1: An overview of the frameworks for different question answering tasks.

comprehension model on the retrieved passages through distant supervision. However, the limitation of distant supervision is that the model will treat all the passages that contain the answer as golden passages. For example, the question can be "Where's Singapore Management University?", and one IR retrieved passage can be "SMU is in Singapore", while another passage can be "Multiple universities are in Singapore". Although the ground-truth answer "Singapore" appears in both of the passages, the second passage is not closely related to the question. And it shouldn't be used to train our reading comprehension model. In this way, we propose to use a neural ranker to select the passages for training the reading comprehension model (reader), and make use of reinforcement learning to jointly train ranker and reader. The experiment results show the effectiveness of our model on several open-domain QA datasets.

Although different QA tasks as discussed above will require some task-

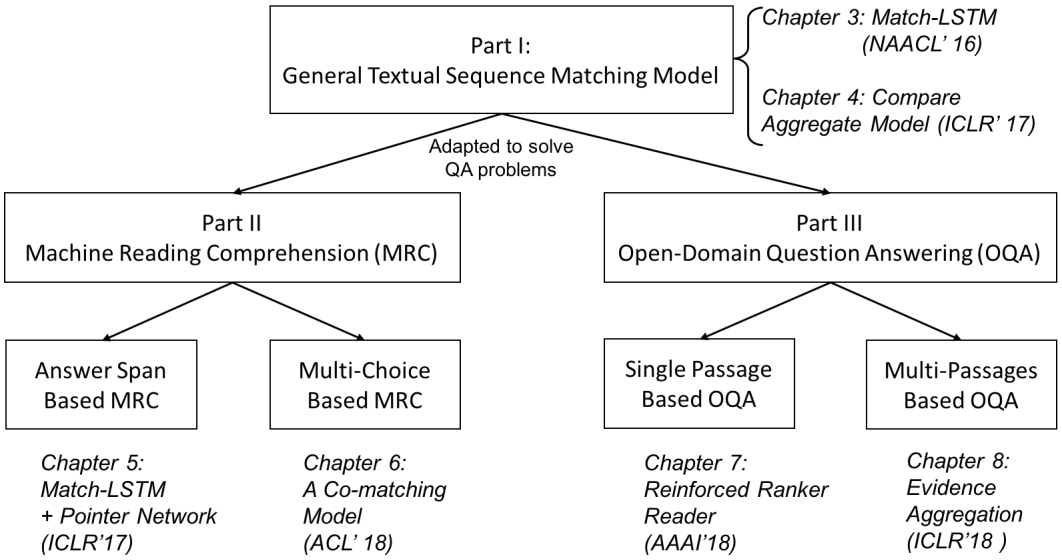

Figure 1.2: An overview of the major components in the thesis.

specific frameworks to solve the problem, they actually rely heavily on a good textual sequence matching model. As shown in Figure 1.1, we have a comparison of the frameworks to solve different tasks. The sequence matching model is to build a matching representation between two sequences, as shown in Figure 1.1 (a). It can be directly applied to the answer selection task by matching every candidate with the question and have a comparison of the matching results for ranking, as shown in Figure 1.1 (b). For the tasks of machine reading comprehension and open-domain question answering, the sequence matching model is always the bottom layer to build the matching representations between question and the passages, as shown in Figure 1.1 (c) and (d). The differences lie on how the reasoning module is constructed to extract the answer based on the matching representations.

Overall, this thesis consists of three parts, as shown in Figure 1.2: (I) textual sequence matching model, and (II) the adaptation of the sequence matching model to machine reading comprehension and (III) the adaptation of the sequence matching model to open-domain question answering tasks. In Part I, we propose two general sequence matching models to solve the answer

selection and textual entailment tasks. In Part II, we propose two models to solve two types of machine reading comprehension tasks (MRC): answer span based and multi-choice based MRC. In Part III, we propose two models to handle noise and diversity issues in the IR retrieved passages from the task of open-domain question answering respectively.

1.2 Thesis Outline and Contributions

In Chapter 2, I will show the related works for the three major parts of this thesis. And then I'll introduce more details of our models in the following chapters.

1.2.1 Textual Sequence Matching

In Chapter 3, I will first address the task of natural language inference, one of key tasks for solving question answering. In this task, we need to identify the relationship between a premise and a hypothesis. The relationship can be "entailment", "contradiction" or "neutral". To solve this problem, models with sequence matching are always applied. In previous work, LSTM based siamese neural networks and LSTM with attention mechanism have been explored on this task. Unlike these models based on matching between the sequence-level representations, we propose a model named match-LSTM [86] which performs the matching in a word-by-word manner. In more details, we will first use LSTM to pre-process both sequences. Then every hidden state of these pre-processing LSTM will integrate not only the word information in the corresponding position but also its context information. Next, we further use these states to compute the attention weights. For each state in one sequence, we compute an attention vector which is the weighted sum of all the hidden states of the other sequence, and each pair of vectors (one hidden state and its attention vector) can represent the matched word pairs. And they will be

the inputs of another LSTM which aggregates all the matched word pairs in order. The final state of this LSTM will be used for the final classification. By making use of this model, we achieved state of the art performance on Stanford Natural Language Inference [7] dataset at the time our paper was published..

In Chapter 4, we further explore a more general and efficient compare-aggregate model [87] for sequence matching. For the compare-aggregate model, we further research on the different word-level matching functions in the "compare" part, which is to build the representation between each word in one sequence and its corresponding attention vector, the weighted sum of all the hidden states of the other sequence. We start from applying the most complex function of Neural Tensor Networks to the simplest function of Euclidean Distance to make the word level comparison across sequences. Then we further add a CNN layer to aggregate the comparison representations. To test the effectiveness of different comparison functions, we further explore our model on another three answer selection datasets besides the natural language inference dataset. According to the experiment results, we find that the element-wise comparison function with the complexity in the middle works the best, and achieved state of the art on four different datasets.

1.2.2 Machine Reading Comprehension

In Chapter 5, I will focus on the task of answer-span based reading comprehension, where a passage is given for each question and the ground-truth answer can be extracted from the given passage. To get the answer positions in the passage, we need to match the passage with question first to locate the potential sub-sequence that could match the question. Then, another reasoning layer could be added to extract the exact answer span based on the previous matching representations. Based on this process, our model [88] uses match-LSTM to match the passage with the question, and further uses Pointer Network to

find the answer. The pointer network is the reasoning module shown in Figure 1.1 (c), and it can point out which word in the given passage can be used to compose the final answer. And we propose two ways to find the answer: 1) word-by-word pointing until the end, so that all the pointed words will compose the answer sequence; 2) start and end positions pointing, and all the words between the start and end positions will compose the answer sequence. Based on the experiment results, we can clearly see that the second way for answer extraction is much better and achieved state of the art on SQuAD [61] and MSMARCO [57] datasets.

In Chapter 6, we try to solve another more challenging reading comprehension task, the English examination from middle/high school. A question related passage is given for this task. Besides, several candidate answers are also given, and we can select the answer from the candidates. That is the major difference between previous answer span based reading comprehension and this task. For our model, instead of building representation between two types of sequence, either question matching passage (span-based reading comprehension) or question matching candidate (answer selection), we need to build sequence matching representations for three types of sequences: passage, question, candidate answers. As the passage is the key component to explain how the ground-truth answer can entail the question, we propose a "co-matching" based model to match the passage with question and candidate answers in word-level at the same time, and make use of another hierarchical structure to integrate the "co-matching" representation. And our model achieved state of the art performance on the RACE dataset [46].

1.2.3 Open-Domain Question Answering

In Chapter 7, we present a novel open-domain QA system called *Reinforced Ranker-Reader* (R^3) consisting of a ranking model (ranker) and a reading com-

prehension model (reader). Our model is to extract the answer from IR retrieved passages, but the problem is that we don't know which passage can entail/answer the question. Even thought some passages containing the ground truth, they should not be used to train our reading comprehension model. In this way, we propose to use a neural ranker to select the passage for training the reader which can extract the exact answer phrase from the selected passage. As the selecting action by ranker is non-differentiable, in order to jointly train the ranker and reader, we make use of the method REINFORCE [96]. And our experiments show that our Reinforced Ranker-Reader can outperform the baseline which ensembles the ranker and reader trained separately under distance supervision. Moreover, our model achieved state-of-the-art performance on three open-domain QA datasets.

In Chapter 8, by observing that some questions require a combination of evidence from across different sources to answer correctly, we propose two methods to address this issue. Suppose the passages that contain the same candidate answer are strongly related, and our goal is to make use these passages for evidence aggregation to answer questions. The easiest way to get the candidate answers is just using the *Reinforced Ranker-Reader* (R^3) in chapter 7, and we could re-rank these candidates by combining the evidence from the passages containing the corresponding candidate answer. We propose two methods, namely, *strength-based* re-ranking and *coverage-based* re-ranking, to solve the problem. For *strength-based* re-ranking, the hypothesis is that the more evidence that can support the candidate, the more likely it would be the ground-truth answer. Here, we treat the passage as supporting evidence for one candidate if the pre-trained model *Reinforced Ranker-Reader* (R^3) can generate the candidate. Then we can rank the candidates by either counting the number of supporting evidence or accumulating the probabilities provided by (R^3). For the *coverage-based* re-ranking, we will concatenate all the passages containing one candidate answer together and treat it as a new sequence,

so that the new sequence will contain all the evidence related to the candidate answer. Then we will directly match the new sequence with the question to check whether the aggregated evidence from different passages can entail the question. The model here is to use textual sequence matching for ranking. According to our experiment results, both of our models can outperform previous best results on three different open-domain QA datasets.

Overall, this thesis will cover my published papers as the first author as follows:

- Learning Natural Language Inference with LSTM [86], In *Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics (NAACL), 2016.*

- A Compare-aggregate Model for Matching Text Sequences [87], In *Proceedings of the International Conference on Learning Representations (ICLR), 2017.*

- Machine Comprehension Using Match-LSTM and Answer Pointer [88], In *Proceedings of the International Conference on Learning Representations (ICLR), 2017.*

- A Co-Matching Model for Multi-choice Reading Comprehension [89], In *Proceedings of the Conference on Association for Computational Linguistics (ACL), 2018.*

- R^3: Reinforced Reader-Ranker for Open-Domain Question Answering [90], In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI), 2018.*

- Evidence Aggregation for Answer Re-Ranking in Open-Domain Question Answering [91], In *Proceedings of the International Conference on Learning Representations (ICLR), 2018*

Chapter 2

Related work

In this chapter, I'll discuss the related work on solving different question answering tasks, especially machine reading comprehension and open-domain question answering, with textual sequence matching models.

2.1 Textual Sequence Matching

Many answer selection tasks are based on the the textual sequence matching model. We will review related works in three types of general structures for matching sequences.

Siamense network: These kinds of models use the same structure, such as RNN or CNN, to build the representations for the sequences separately. Then cosine similarity [26, 99], element-wise operation [71, 55] or neural network-based combination [7] are used for sequence matching.

Attentive network: Soft-attention mechanism [2, 49] has been widely used for sequence matching in machine comprehension [34], text entailment [64] and question answering [72]. Instead of using the final state of RNN to represent a sequence, these studies use weighted sum of all the states for the sequence representation.

Compare-Aggregate network: This kind of framework is to perform

the word level matching [88, 58, 33, 76, 83]. Our works are under this framework. But our structures are different from previous models and our model can be applied on different tasks. Besides, we analyzed different word-level comparison functions separately.

2.2 Machine Reading Comprehension

Machine reading comprehension of text has gained much attention in recent years, and increasingly researchers are building data-driven, end-to-end neural network models for the task. We will first review the recently released datasets and then some end-to-end models on this task.

2.2.1 Datasets

A number of datasets for studying machine reading comprehension were created in Cloze style by removing a single token from a sentence in the original corpus, and the task is to predict the missing word. For example, [34] created questions in Cloze style from CNN and Daily Mail highlights. [36] created the Children's Book Test dataset, which is based on children's stories. [19] released two similar datasets in Chinese, the People Daily dataset and the Children's Fairy Tale dataset.

Instead of creating questions in Cloze style, a number of other datasets rely on human annotators to create real questions. [62] created the well-known MCTest dataset and [75] created the MovieQA dataset. In these datasets, candidate answers are provided for each question. Similar to these two datasets, the SQuAD dataset [61] was also created by human annotators. Different from the previous two, however, the SQuAD dataset does not provide candidate answers, and thus all possible subsequences from the given passage have to be considered as candidate answers.

Besides the datasets above, there are also a few other datasets created

for machine comprehension, such as WikiReading dataset [35] and bAbI dataset [94], but they are quite different from the datasets above in nature.

2.2.2 End-to-end Neural Network Models for Machine Comprehension

There have been a number of studies proposing end-to-end neural network models for machine comprehension. A common approach is to use recurrent neural networks (RNNs) to process the given text and the question in order to predict or generate the answers [34]. Attention mechanism is also widely used on top of RNNs in order to match the question with the given passage [34, 9]. Given that answers often come from the given passage, Pointer Network has been adopted in a few studies in order to copy tokens from the given passage as answers [41, 78]. Compared with existing work, we use match-LSTM to match a question and a given passage, and we use Pointer Network in a different way such that we can generate answers that contain multiple tokens from the given passage.

Memory Networks [95] have also been applied to machine comprehension [70, 44, 36], but its scalability when applied to a large dataset is still an issue. In this thesis, we did not consider memory networks for the SQuAD/MSMARCO datasets.

2.3 Open-domain Question Answering

Open domain question answering dates back to as early as [31] and was popularized with TREC-8 [80]. The task is to answer a question by exploiting resources such as documents [80], webpages [45, 13] or structured knowledge bases [5, 6, 103]. An early consensus since TREC-8 has produced an approach with three major components: question analysis, document retrieval and rank-

ing, and answer extraction. Although question analysis is relatively mature, answer extraction and document ranking still represent significant challenges.

Very recently, IR plus machine reading comprehension (**SR-QA**) showed promise for open-domain QA, especially after datasets created specifically for the multiple-passage RC setting [57, 10, 40, 24, 23]. These datasets deal with the end-to-end open-domain QA setting, where only question-answer pairs provide supervision. Similarly to previous work on open-domain QA, existing deep learning based solutions to the above datasets also rely on a document retrieval module to retrieve a list of passages for RC models to extract answers. Therefore, these approaches suffer from the limitation that the passage ranking scores are determined by n-gram matching (with tf-idf weighting), which is not ideal for QA.

Our ranker module in R^3 could help to alleviate the above problem, and RL is a natural fit to jointly train the ranker and reader since the passages do not have ground-truth labels. Our work is related to the idea of soft or hard attentions (usually with reinforcement learning) for hierarchical or coarse-to-fine decision sequences making in NLP, where the attentions themselves are latent variables. For example, [47] propose to first extract informative text fragments then feed them to text classification and question retrieval models. [15] and [16] proposed coarse-to-fine frameworks with an additional sentence selection step before the original word-level prediction for text summarization and reading comprehension, respectively. To the best of our knowledge, we are the first apply this kind of framework to the open-domain question answering.

From the method-perspective, our work is most close to [16]'s work in terms of the usage of REINFORCE. Our main aim is to deal with the lack of annotation in the passage selection step, which is a necessary intermediate step in open-domain QA. In comparison, [16] has as its main aim to speed up the RC model in the single passage setting. From the motivation-perspective, we are similar to [56]'s work. Both work aim to find passages easy and suitable

for the QA or IE models to extract answers, in order to boost accuracy.

Part I

General Textual Sequence Matching Models

Chapter 3

Learning Natural Language Inference with Match-LSTM

In this chapter, I will introduce "match-LSTM", which is based on sequence-to-sequence word-level matching, for the task of natural language inference.

3.1 Introduction

Natural language inference (NLI) is the problem of determining whether from a premise sentence P one can infer another hypothesis sentence H [50]. NLI is a fundamentally important problem that has applications in many tasks including question answering, semantic search and automatic text summarization. There has been much interest in NLI in the past decade, especially surrounding the PASCAL Recognizing Textual Entailment (RTE) Challenge [20]. Existing solutions to NLI range from shallow approaches based on lexical similarities [29] to advanced methods that consider syntax [53], perform explicit sentence alignment [51] or use formal logic [17].

Recently, [7] released the Stanford Natural Language Inference (SNLI) corpus for the purpose of encouraging more learning-centered approaches to NLI. This corpus contains around 570K sentence pairs with three labels: *entailment*,

contradiction and *neutral*. The size of the corpus makes it now feasible to train deep neural network models, which typically require a large amount of training data. [7] tested a straightforward architecture of deep neural networks for NLI. In their architecture, the premise and the hypothesis are each represented by a sentence embedding vector. The two vectors are then fed into a multi-layer neural network to train a classifier. [7] achieved an accuracy of 77.6% when long short-term memory (LSTM) networks were used to obtain the sentence embeddings.

A more recent work by [65] improved the performance by applying a neural attention model. While their basic architecture is still based on sentence embeddings for the premise and the hypothesis, a key difference is that the embedding of the premise takes into consideration the alignment between the premise and the hypothesis. This so-called *attention-weighted* representation of the premise was shown to help push the accuracy to 83.5% on the SNLI corpus.

A limitation of the aforementioned two models is that they reduce both the premise and the hypothesis to a single embedding vector before matching them; i.e., in the end, they use two embedding vectors to perform sentence-level matching. However, not all word or phrase-level matching results are equally important. For example, the matching between stop words in the two sentences is not likely to contribute much to the final prediction. Also, for a hypothesis to *contradict* a premise, a single word or phrase-level mismatch (e.g., a mismatch of the subjects of the two sentences) may be sufficient and other matching results are less important, but this intuition is hard to be captured if we directly match two sentence embeddings.

In this chapter, we propose a new LSTM-based architecture for learning natural language inference. Different from previous models, our prediction is not based on whole sentence embeddings of the premise and the hypothesis. Instead, we use an LSTM to perform *word-by-word* matching of the hypothesis

with the premise. Our LSTM sequentially processes the hypothesis, and at each position, it tries to match the current word in the hypothesis with an attention-weighted representation of the premise. Matching results that are critical for the final prediction will be "remembered" by the LSTM while less important matching results will be "forgotten." We refer to this architecture a match-LSTM, or mLSTM for short.

Experiments show that our mLSTM model achieves an accuracy of 86.1% on the SNLI corpus, outperforming the state of the art. Furthermore, through further analyses of the learned parameters, we show that the mLSTM architecture can indeed pick up the more important word-level matching results that need to be remembered for the final prediction. In particular, we observe that good word-level matching results are generally "forgotten" but important mismatches, which often indicate a *contradiction* or a *neutral* relationship, tend to be "remembered."

Our code is available online[1].

3.2 Model

In this section, we first review the word-by-word attention model by [65], which is their best performing model. Then we present our mLSTM architecture for natural language inference.

3.2.1 Neural Attention Model

For the natural language inference task, we have two sentences $\mathbf{X}^{\mathrm{s}} = (\mathbf{x}_1^{\mathrm{s}}, \mathbf{x}_2^{\mathrm{s}}, \ldots, \mathbf{x}_M^{\mathrm{s}})$ and $\mathbf{X}^{\mathrm{t}} = (\mathbf{x}_1^{\mathrm{t}}, \mathbf{x}_2^{\mathrm{t}}, \ldots, \mathbf{x}_N^{\mathrm{t}})$, where $\mathbf{X}^{\mathrm{s}}$ is the premise and $\mathbf{X}^{\mathrm{t}}$ is the hypothesis. Here each $\mathbf{x}$ is an embedding vector of the corresponding word. The goal is to predict a label y that indicates the relationship between $\mathbf{X}^{\mathrm{s}}$ and $\mathbf{X}^{\mathrm{t}}$. In this chapter, we assume y is one of *entailment*, *contradiction*

[1] `https://github.com/shuohangwang/SeqMatchSeq`

and *neutral*.

[65] first used two LSTMs to process the premise and the hypothesis, respectively, but initialized the second LSTM (for the hypothesis) with the last cell state of the first LSTM (for the premise). Let us use $\mathbf{h}^{\mathrm{s}}_j$ and $\mathbf{h}^{\mathrm{t}}_k$ to denote the resulting hidden states corresponding to $\mathbf{x}^{\mathrm{s}}_j$ and $\mathbf{x}^{\mathrm{t}}_k$, respectively. The main idea of the word-by-word attention model by [65] is to introduce a series of attention-weighted combinations of the hidden states of the premise, where each combination is for a particular word in the hypothesis. Let us use $\mathbf{a}_k$ to denote such an *attention vector* for word $\mathbf{x}^{\mathrm{t}}_k$ in the hypothesis. Specifically, $\mathbf{a}_k$ is defined as follows[2]:

$$\mathbf{a}_k = \sum_{j=1}^{M} \alpha_{kj} \mathbf{h}^{\mathrm{s}}_j, \tag{3.1}$$

where α_{kj} is an attention weight that encodes the degree to which $\mathbf{x}^{\mathrm{t}}_k$ in the hypothesis is aligned with $\mathbf{x}^{\mathrm{s}}_j$ in the premise. The attention weight α_{kj} is generated in the following way:

$$\alpha_{kj} = \frac{\exp(e_{kj})}{\sum_{j'} \exp(e_{kj'})}, \tag{3.2}$$

where

$$e_{kj} = \mathbf{w}^{\mathrm{e}} \cdot \tanh(\mathbf{W}^{\mathrm{s}} \mathbf{h}^{\mathrm{s}}_j + \mathbf{W}^{\mathrm{t}} \mathbf{h}^{\mathrm{t}}_k + \mathbf{W}^{\mathrm{a}} \mathbf{h}^{\mathrm{a}}_{k-1}). \tag{3.3}$$

Here $\cdot$ is the dot-product between two vectors, the vector $\mathbf{w}^{\mathrm{e}} \in \mathbb{R}^d$ and all matrices $\mathbf{W}^* \in \mathbb{R}^{d \times d}$ contain weights to be learned, and $\mathbf{h}^{\mathrm{a}}_{k-1}$ is another hidden state which we will explain below.

[2]We present the word-by-word attention model by [65] in a different way but the underlying model is the same. Our $\mathbf{h}^{\mathrm{a}}_k$ is their $\mathbf{r}_t$, our $\mathbf{H}^{\mathrm{s}}$ (all of $\mathbf{h}^{\mathrm{s}}_j$) is their $\mathbf{Y}$, our $\mathbf{h}^{\mathrm{t}}_k$ is their $\mathbf{h}_{\mathrm{t}}$, and our α_k is their α_t. Our presentation is close to the one by [2], with our attention vectors $\mathbf{a}$ corresponding to the context vectors $\mathbf{c}$ in their paper.

The attention-weighted premise $\mathbf{a}_k$ essentially tries to model the relevant parts in the premise with respect to $\mathbf{x}^{\mathrm{t}}_k$, i.e., the k^{th} word in the hypothesis. [65] further built an RNN model over $\{\mathbf{a}_k\}_{k=1}^N$ by defining the following hidden states:

$$\mathbf{h}^{\mathrm{a}}_k = \mathbf{a}_k + \tanh(\mathbf{V}^{\mathrm{a}}\mathbf{h}^{\mathrm{a}}_{k-1}), \tag{3.4}$$

where $\mathbf{V}^{\mathrm{a}} \in \mathbb{R}^{d\times d}$ is a weight matrix to be learned. We can see that the last $\mathbf{h}^{\mathrm{a}}_N$ aggregates all the previous $\mathbf{a}_k$ and can be seen as an attention-weighted representation of the whole premise. [65] then used this $\mathbf{h}^{\mathrm{a}}_N$, which represents the whole premise, together with $\mathbf{h}^{\mathrm{t}}_N$, which can be approximately regarded as an aggregated representation of the hypothesis[3], to predict the label y.

3.2.2 Our Model

Although the neural attention model by [65] achieved better results than [7], we see two limitations. First, the model still uses a single vector representation of the premise, namely $\mathbf{h}^{\mathrm{a}}_N$, to match the entire hypothesis. We speculate that if we instead use each of the attention-weighted representations of the premise for matching, i.e., use $\mathbf{a}_k$ at position k to match the hidden state $\mathbf{h}^{\mathrm{t}}_k$ of the hypothesis while we go through the hypothesis, we could achieve better matching results. This can be done using an RNN which at each position takes in both $\mathbf{a}_k$ and $\mathbf{h}^{\mathrm{t}}_k$ as its input and determines how well the overall matching of the two sentences is up to the current position. In the end the RNN will produce a single vector representing the matching of the two entire sentences.

The second limitation is that the model by [65] does not explicitly allow us to place more emphasis on the more important matching results between

[3]Strictly speaking, in the model by [65], $\mathbf{h}^{\mathrm{t}}_N$ encodes both the premise and the hypothesis because the two sentences are chained. But $\mathbf{h}^{\mathrm{t}}_N$ places a higher emphasis on the hypothesis given the nature of RNNs.

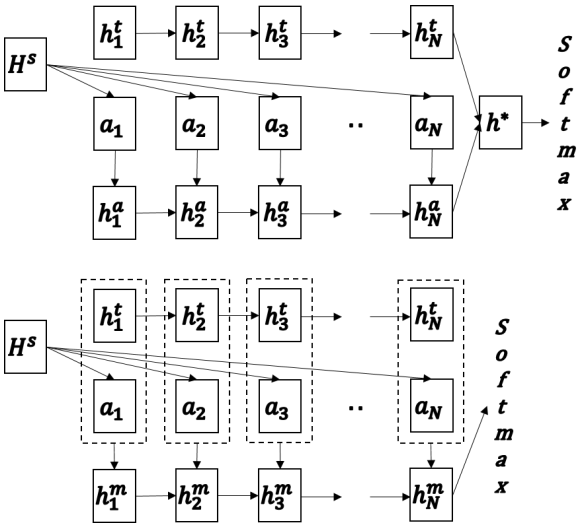

Figure 3.1: The top figure depicts the model by Rocktäschel et al. (2016) and the bottom figure depicts our model. Here $\mathbf{H}^{\mathrm{s}}$ represents all the hidden states $\mathbf{h}_j^{\mathrm{s}}$. Note that in the top model each $\mathbf{h}_k^{\mathrm{a}}$ represents a weighted version of the premise only, while in our model, each $\mathbf{h}_k^{\mathrm{m}}$ represents the matching between the premise and the hypothesis up to position k.

the premise and the hypothesis and down-weight the less critical ones. For example, matching of stop words is presumably less important than matching of content words. Also, some matching results may be particularly critical for making the final prediction and thus should be remembered. For example, consider the premise "*A dog jumping for a Frisbee in the snow.*" and the hypothesis "*A cat washes his face and whiskers with his front paw.*" When we sequentially process the hypothesis, once we see that the subject of the hypothesis *cat* does not match the subject of the premise *dog*, we have a high probability to believe that there is a contradiction. So this mismatch should be remembered.

Based on the two observations above, we propose to use an LSTM to sequentially match the two sentences. At each position the LSTM takes in both $\mathbf{a}_k$ and $\mathbf{h}_k^{\mathrm{t}}$ as its input. Figure 3.1 gives an overview of our model in contrast to the model by [65].

Specifically, our model works as follows. First, similar to [65], we process the premise and the hypothesis using two LSTMs, but we do not feed the last

cell state of the premise to the LSTM of the hypothesis. This is because we do not need the LSTM for the hypothesis to encode any knowledge about the premise but we will match the premise with the hypothesis using the hidden states of the two LSTMs. Again, we use $\mathbf{h}_j^{\mathrm{s}}$ and $\mathbf{h}_k^{\mathrm{t}}$ to represent these hidden states.

Next, we generate the attention vectors $\mathbf{a}_k$ similarly to Eqn (3.1). However, Eqn (3.3) will be replaced by the following equation:

$$e_{kj} = \mathbf{w}^{\mathrm{e}} \cdot \tanh(\mathbf{W}^{\mathrm{s}}\mathbf{h}_j^{\mathrm{s}} + \mathbf{W}^{\mathrm{t}}\mathbf{h}_k^{\mathrm{t}} + \mathbf{W}^{\mathrm{m}}\mathbf{h}_{k-1}^{\mathrm{m}}). \tag{3.5}$$

The only difference here is that we use a hidden state $\mathbf{h}^{\mathrm{m}}$ instead of $\mathbf{h}^{\mathrm{a}}$, and the way we define $\mathbf{h}^{\mathrm{m}}$ is very different from the definition of $\mathbf{h}^{\mathrm{a}}$.

Our $\mathbf{h}_k^{\mathrm{m}}$ is the hidden state at position k generated from our mLSTM. This LSTM models the *matching* between the premise and the hypothesis. Important matching results will be "remembered" by the LSTM while non-essential ones will be "forgotten." We use the concatenation of $\mathbf{a}_k$, which is the attention-weighted version of the premise for the k^{th} word in the hypothesis, and $\mathbf{h}_k^{\mathrm{t}}$, the hidden state for the k^{th} word itself, as input to the mLSTM.

Specifically, let us define

$$\mathbf{m}_k = \begin{bmatrix} \mathbf{a}_k \\ \mathbf{h}_k^{\mathrm{t}} \end{bmatrix}. \tag{3.6}$$

We then build the mLSTM as follows:

$$
\begin{aligned}
\mathbf{i}_k^{\mathrm{m}} &= \sigma(\mathbf{W}^{\mathrm{mi}}\mathbf{m}_k + \mathbf{V}^{\mathrm{mi}}\mathbf{h}_{k-1}^{\mathrm{m}} + \mathbf{b}^{\mathrm{mi}}), \\
\mathbf{f}_k^{\mathrm{m}} &= \sigma(\mathbf{W}^{\mathrm{mf}}\mathbf{m}_k + \mathbf{V}^{\mathrm{mf}}\mathbf{h}_{k-1}^{\mathrm{m}} + \mathbf{b}^{\mathrm{mf}}), \\
\mathbf{o}_k^{\mathrm{m}} &= \sigma(\mathbf{W}^{\mathrm{mo}}\mathbf{m}_k + \mathbf{V}^{\mathrm{mo}}\mathbf{h}_{k-1}^{\mathrm{m}} + \mathbf{b}^{\mathrm{mo}}), \\
\mathbf{c}_k^{\mathrm{m}} &= \mathbf{f}_k^{\mathrm{m}} \odot \mathbf{c}_{k-1}^{\mathrm{m}} + \mathbf{i}_k^{\mathrm{m}} \odot \tanh(\mathbf{W}^{\mathrm{mc}}\mathbf{m}_k + \mathbf{V}^{\mathrm{mc}}\mathbf{h}_{k-1}^{\mathrm{m}} \\
&\quad + \mathbf{b}^{\mathrm{mc}}), \\
\mathbf{h}_k^{\mathrm{m}} &= \mathbf{o}_k^{\mathrm{m}} \odot \tanh(\mathbf{c}_k^{\mathrm{m}}). \qquad (3.7)
\end{aligned}
$$

With this mLSTM, finally we use only $\mathbf{h}_N^{\mathrm{m}}$, the last hidden state, to predict the label y.

3.2.3 Implementation Details

Besides the difference of the LSTM architecture, we also introduce a few other changes from the model by [65]. First, we insert a special word *NULL* to the premise, and we allow words in the hypothesis to be aligned with this *NULL*. This is inspired by common practice in machine translation. Specifically, we introduce a vector $\mathbf{h}_0^{\mathrm{s}}$, which is fixed to be a vector of 0s of dimension d. This $\mathbf{h}_0^{\mathrm{s}}$ represents *NULL* and is used with other $\mathbf{h}_j^{\mathrm{s}}$ to derive the attention vectors $\{\mathbf{a}_k\}_{k=1}^N$.

Second, we use word embeddings trained from GloVe [59] instead of word2vec vectors. The main reason is that GloVe covers more words in the SNLI corpus than word2vec[4].

Third, for words which do not have pre-trained word embeddings, we take the average of the embeddings of all the words (in GloVe) surrounding the unseen word within a window size of 9 (4 on the left and 4 on the right) as an approximation of the embedding of this unseen word. Then we do not update

[4]The SNLI corpus contains 37K unique tokens. Around 12.1K of them cannot be found in word2vec but only around 4.1K of them cannot be found in GloVe.

Model	d	$\|\theta\|_{\text{W+M}}$	$\|\theta\|_{\text{M}}$	Train	Dev	Test
LSTM [[7]]	100	10M	221K	84.4	-	77.6
Classifier [[7]]	-	-	-	99.7	-	78.2
LSTM shared [[65]]	159	3.9M	252K	84.4	83.0	81.4
Word-by-word attention [[65]]	100	3.9M	252K	85.3	83.7	83.5
Word-by-word attention (our implementation)	150	340K	340K	85.5	83.3	82.6
mLSTM	150	544K	544K	91.0	86.2	85.7
mLSTM with bi-LSTM sentence modeling	150	1.4M	1.4M	91.3	86.6	86.0
mLSTM	300	1.9M	1.9M	92.0	**86.9**	**86.1**
mLSTM with word embedding	300	1.3M	1.3M	88.6	85.4	85.3

Table 3.1: Experiment results in terms of accuracy. d is the dimension of the hidden states. $|\theta|_{\text{W+M}}$ is the total number of parameters and $|\theta|_{\text{M}}$ is the number of parameters excluding the word embeddings. Note that the five models in the last section were implemented by us while the other results were taken directly from previous papers. Note also that for the five models in the last section, we do not update word embeddings so $|\theta|_{\text{W+M}}$ is the same as $|\theta|_{\text{M}}$. The three columns on the right are the accuracies of the trained models on the training data, the development data and the test data, respectively.

any word embedding when learning our model. Although this is a very crude approximation, it reduces the number of parameters we need to update, and as it turns out, we can still achieve better performance than [65].

3.3 Experiments

3.3.1 Experiment Settings

Data: We use the SNLI corpus to test the effectiveness of our model. The original data set contains 570,152 sentence pairs, each labeled with one of the following relationships: *entailment*, *contradiction*, *neutral* and –, where – indicates a lack of consensus from the human annotators. We discard the sentence pairs labeled with – and keep the remaining ones for our experiments. In the end, we have 549,367 pairs for training, 9,842 pairs for development and 9,824 pairs for testing. This follows the same data partition used by [7] in their experiments. We perform three-class classification and use accuracy as

our evaluation metric.

Parameters: We use the Adam method [43] with hyperparameters β_1 set to 0.9 and β_2 set to 0.999 for optimization. The initial learning rate is set to be 0.001 with a decay ratio of 0.95 for each iteration. The batch size is set to be 30. We experiment with $d = 150$ and $d = 300$ where d is the dimension of all the hidden states.

Methods for comparison: We mainly want to compare our model with the word-by-word attention model by [65] because this model achieved the state-of-the-art performance on the SNLI corpus. To ensure fair comparison, besides comparing with the accuracy reported by [65], we also re-implemented their model and report the performance of our implementation. We also consider a few variations of our model. Specifically, the following models are implemented and tested in our experiments:

- Word-by-word attention ($d = 150$): This is our implementation of the word-by-word attention model by [65], where we set the dimension of the hidden states to 150. The differences between our implementation and the original implementation by [65] are the following: (1) We also add a *NULL* token to the premise for matching. (2) We do not feed the last cell state of the LSTM for the premise to the LSTM for the hypothesis, to keep it consistent with the implementation of our model. (3) For word representation, we also use the GloVe word embeddings and we do not update the word embeddings. For unseen words, we adopt the same strategy as described in Section 3.2.3.
- mLSTM ($d = 150$): This is our mLSTM model with d set to 150.
- mLSTM with bi-LSTM sentence modeling ($d = 150$): This is the same as the model above except that when we derive the hidden states $\mathbf{h}_j^{\mathrm{s}}$ and $\mathbf{h}_k^{\mathrm{t}}$ of the two sentences, we use bi-LSTMs [30] instead of LSTMs. We implement this model to see whether bi-LSTMs allow us to better align the sentences.

prediction	ground truth		
	N	E	C
N	2628	286	255
E	340	3005	159
C	250	77	2823

Table 3.2: The confusion matrix of the results by mLSTM with $d = 300$. N, E and C correspond to *neutral*, *entailment* and *contradiction*, respectively.

	ID	**sentence**	**label**
Premise		A dog jumping for a Frisbee in the snow.	
Hypothesis	Example 1	An animal is outside in the cold weather, playing with a plastic toy.	*entailment*
	Example 2	A cat washed his face and whiskers with his front paw.	*contradiction*
	Example 3	A pet is enjoying a game of fetch with his owner.	*neutral*

Table 3.3: Three examples of sentence pairs with different relationship labels. The second hypothesis is a contradiction because it mentions a completely different event. The third hypothesis is neutral to the premise because the phrase "with his owner" cannot be inferred from the premise.

- mLSTM ($d = 300$): This is our mLSTM model with d set to 300.
- mLSTM with word embedding ($d = 300$): This is the same as the model above except that we directly use the word embedding vectors $\mathbf{x}^{\mathrm{s}}_j$ and $\mathbf{x}^{\mathrm{t}}_k$ instead of the hidden states $\mathbf{h}^{\mathrm{s}}_j$ and $\mathbf{h}^{\mathrm{t}}_k$ in our model. In this case, each attention vector $\mathbf{a}_k$ is a weighted sum of $\{\mathbf{x}^{\mathrm{s}}_j\}_{j=1}^{M}$. We experiment with this setting because we hypothesize that the effectiveness of our model is largely related to the mLSTM architecture rather than the use of LSTMs to process the original sentences.

3.3.2 Main Results

Table 3.1 compares the performance of the various models we tested together with some previously reported results.

We have the following observations: (1) First of all, we can see that when we set d to 300, our model achieves an accuracy of 86.1% on the test data, which to the best of our knowledge is the highest on this data set. (2) If we

compare our mLSTM model with our implementation of the word-by-word attention model by [65] under the same setting with $d = 150$, we can see that our performance on the test data (85.7%) is higher than that of their model (82.6%). We also tested statistical significance and found the improvement to be statistically significant at the 0.001 level. (3) The performance of mLSTM with bi-LSTM sentence modeling compared with the model with standard LSTM sentence modeling when d is set to 150 shows that using bi-LSTM to process the original sentences helps (86.0% vs. 85.7% on the test data), but the difference is small and the complexity of bi-LSTM is much higher than LSTM. Therefore when we increased d to 300 we did not experiment with bi-LSTM sentence modeling. (4) Interestingly, when we experimented with the mLSTM model using the pre-trained word embeddings instead of LSTM-generated hidden states as initial representations of the premise and the hypothesis, we were able to achieve an accuracy of 85.3% on the test data, which is still better than previously reported state of the art. This suggests that the mLSTM architecture coupled with the attention model works well, regardless of whether or not we use LSTM to process the original sentences.

Because the NLI task is a three-way classification problem, to better understand the errors, we also show the confusion matrix of the results obtained by our mLSTM model with $d = 300$ in Table 3.2. We can see that there is more confusion between *neutral* and *entailment* and between *neutral* and *contradiction* than between *entailment* and *contradiction*. This shows that *neutral* is relatively hard to capture.

3.3.3 Further Analyses

To obtain a better understanding of how our proposed model actually performs the matching between a premise and a hypothesis, we further conduct the following analyses. First, we look at the learned word-by-word alignment weights

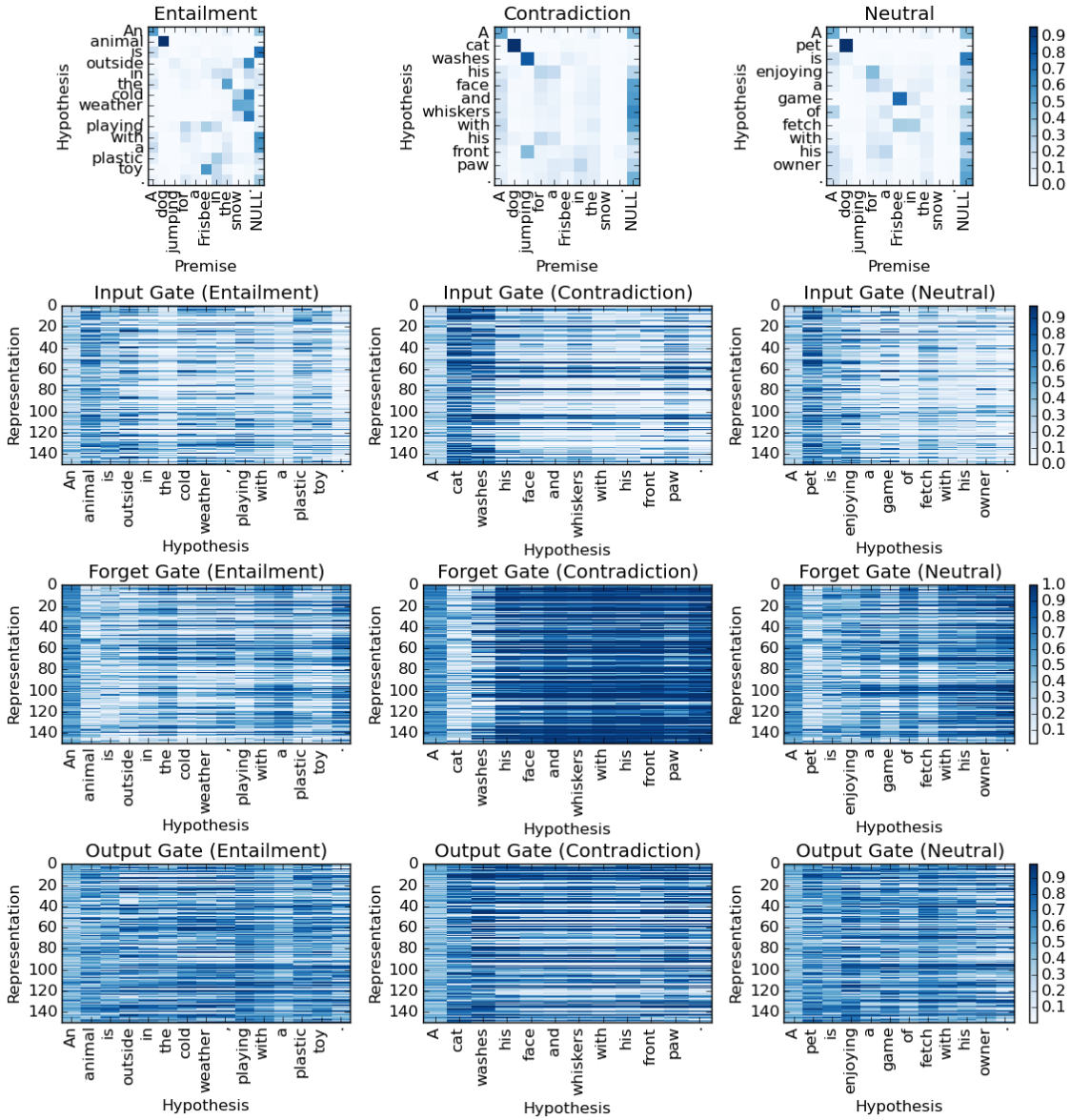

Figure 3.2: The alignment weights and the gate vectors of the three examples.

α_{kj} to check whether the soft alignment makes sense. This is the same as what was done by [65]. We then look at the values of the various gate vectors of the mLSTM. By looking at these values, we aim to check (1) whether the model is able to differentiate between more important and less important word-level matching results, and (2) whether the model forgets certain matching results and remembers certain other ones.

To conduct the analyses, we choose three examples and display the various learned parameter values. These three sentence pairs share the same premise

but have different hypotheses and different relationship labels. They are given in Table 3.3. The values of the alignment weights and the gate vectors are plotted in Figure 3.2.

Besides using the three examples, we will also give some overall statistics of the parameter values to confirm our observations with the three examples.

Word Alignment

First, let us look at the top-most plots of Figure 3.2. These plots show the alignment weights α_{kj} between the hypothesis and the premise, where a darker color corresponds to a larger value of α_{kj}. Recall that α_{kj} is the degree to which the k^{th} word in the hypothesis is aligned with the j^{th} word in the premise. Also recall that the weights α_{kj} are configured such that for the same k all the α_{kj} add up to 1. This means the weights in the same row in these plots add up to 1. From the three plots we can see that the alignment weights generally make sense. For example, in Example 1, "animal" is strongly aligned with "dog" and "toy" aligned with "Frisbee." The phrase "cold weather" is aligned with "snow." In Example 3, we also see that "pet" is strongly aligned with "dog" and "game" aligned with "Frisbee."

In Example 2, "cat" is strongly aligned with "dog" and "washes" is aligned with "jumping." It may appear that these matching results are wrong. However, "dog" is likely the best match for "cat" among all the words in the premise, and as we will show later, this match between "cat" and "dog" is actually a strong indication of a contradiction between the two sentences. The same explanation applies to the match between "washes" and "jumping."

We also observe that some words are aligned with the *NULL* token we inserted. For example, the word "is" in the hypothesis in Example 1 does not correspond to any word in the premise and is therefore aligned with *NULL*. The words "face" and "whiskers" in Example 2 and "owner" in Example 3 are

also aligned with *NULL*. Intuitively, if some important content words in the hypothesis are aligned with *NULL*, it is more likely that the relationship label is either contradiction or neutral.

Values of Gate Vectors

Next, let us look at the values of the learned gate vectors of our mLSTM for the three examples. We show these values under the setting where d is set to 150. Each row of these plots corresponds to one of the 150 dimensions. Again, a darker color indicates a higher value.

An input gate controls whether the input at the current position should be used in deriving the final hidden state of the current position. From the three plots of the input gates, we can observe that generally for stop words such as prepositions and articles the input gates have lower values, suggesting that the matching of these words is less important. On the other hand, content words such as nouns and verbs tend to have higher values of the input gates, which also makes sense because these words are generally more important for determining the final relationship label.

To further verify the observation above, we compute the average input gate values for stop words and the other content words. We find that the former has an average value of 0.287 with a standard deviation of 0.084 while the latter has an average value of 0.347 with a standard deviation of 0.116. This shows that indeed generally stop words have lower input gate values. Interestingly, we also find that some stop words may have higher input gate values if they are critical for the classification task. For example, the negation word "not" has an average input gate value of 0.444 with a standard deviation of 0.104.

Overall, the values of the input gates confirm that the mLSTM helps differentiate the more important word-level matching results from the less important ones.

Next, let us look at the forget gates. Recall that a forget gate controls the importance of the *previous* cell state in deriving the final hidden state of the current position. Higher values of a forget gate indicate that we need to remember the previous cell state and pass it on whereas lower values indicate that we should probably forget the previous cell. From the three plots of the forget gates, we can see that overall the colors are the lightest for Example 1, which is an *entailment*. This suggests that when the hypothesis is an entailment of the premise, the *m*LSTM tends to forget the previous matching results. On the other hand, for Example 2 and Example 3, which are *contradiction* and *neutral*, we see generally darker colors. In particular, in Example 2, we can see that the colors are consistently dark starting from the word "his" in the hypothesis until the end. We believe the explanation is that after the *m*LSTM processes the first three words of the hypothesis, "A cat washes," it sees that the matching between "cat" and "dog" and between "washes" and "jumping" is a strong indication of a contradiction, and therefore these matching results need to be remembered until the end of the *m*LSTM for the final prediction.

We have also checked the forget gates of the other sentence pairs in the test data by computing the average forget gate values and the standard deviations for *entailment*, *neutral* and *contradiction*, respectively. We find that the values are 0.446 ± 0.123, 0.507 ± 0.148 and 0.536 ± 0.170, respectively. For *contradiction* and *neutral*, the forget gates start to have higher values from certain positions of the hypotheses.

Based on the observations above, we hypothesize that the way the *m*LSTM works is as follows. It remembers important mismatches, which are useful for predicting the *contradiction* or the *neutral* relationship, and forgets good matching results. At the end of the *m*LSTM, if no important mismatch is remembered, the final classifier will likely predict *entailment* by default. Otherwise, depending on the kind of mismatch remembered, the classifier will predict either *contradiction* or *neutral*.

For the output gates, we are not able to draw any important conclusion except that the output gates seem to be positively correlated with the input gates but they tend to be darker than the input gates.

3.4 Conclusions

In this chapter, we proposed a special LSTM architecture for the task of natural language inference. Based on a recent work by [65], we first used neural attention models to derive attention-weighted vector representations of the premise. We then designed a match-LSTM that processes the hypothesis word by word while trying to match the hypothesis with the premise. The last hidden state of this mLSTM can be used for predicting the relationship between the premise and the hypothesis. Experiments on the SNLI corpus showed that the mLSTM model outperformed the state-of-the-art performance reported so far on this data set. Moreover, closer analyses on the gate vectors revealed that our mLSTM indeed remembers and passes on important matching results, which are typically mismatches that indicate a *contradiction* or a *neutral* relationship between the premise and the hypothesis.

With the large number of parameters to learn, an inevitable limitation of our model is that a large training data set is needed to learn good model parameters. Indeed some preliminary experiments applying our mLSTM to the SICK corpus [52], a smaller textual entailment benchmark data set, did not give very good results. We believe that this is because our model learns everything from scratch except using the pre-trained word embeddings. A future direction would be to incorporate other resources such as the paraphrase database [28] into the learning process.

Chapter 4

A Compare-Aggregate Model for Textual Sequence Matching

In this chapter, I will introduce a more general sequence matching model and will focus on exploring a better way to build word-level matching representations.

4.1 Introduction

Many natural language processing problems involve matching two or more sequences to make a decision. For example, in textual entailment, one needs to determine whether a hypothesis sentence can be inferred from a premise sentence [7]. In machine comprehension, given a passage, a question needs to be matched against it in order to find the correct answer [62, 75]. Table 4.1 gives two example sequence matching problems. In the first example, a passage, a question and four candidate answers are given. We can see that to get the correct answer, we need to match the question against the passage and identify the last sentence to be the answer-bearing sentence. In the second example, given a question and a set of candidate answers, we need to find the answer that best matches the question. Because of the fundamental impor-

Plot: ... Aragorn is crowned King of Gondor and taking Arwen as his queen before all present at his coronation bowing before Frodo and the other Hobbits . The Hobbits return to **the Shire** where Sam marries Rosie Cotton	**Question**: can i have auto insurance without a car
Qustion: Where does Sam marry Rosie?	**Ground-truth answer**: yes, it be possible have auto insurance without own a vehicle. you will purchase what be call a name ...
Candidate answers: 0) Grey Havens. 1) Gondor. **2) The Shire**. 3) Erebor 4) Mordor.	**Another candidate answer**: insurance not be a tax or merely a legal obligation because auto insurance follow a car...

Table 4.1: The example on the left is a machine comprehension problem from MovieQA, where the correct answer here is **The Shire**. The example on the right is an answer selection problem from InsuranceQA.

tance of comparing two sequences of text to judge their semantic similarity or relatedness, sequence matching has been well studied in natural language processing.

With recent advances of neural network models in natural language processing, a standard practice for sequence modeling now is to encode a sequence of text as an embedding vector using models such as RNN and CNN. To match two sequences, a straightforward approach is to encode each sequence as a vector and then to combine the two vectors to make a decision [7, 26]. However, it has been found that using a single vector to encode an entire sequence is not sufficient to capture all the important information from the sequence, and therefore advanced techniques such as attention mechanisms and memory networks have been applied to sequence matching problems [34, 36, 64].

A common trait of a number of these recent studies on sequence matching problems is the use of a "compare-aggregate" framework [86, 33, 58]. In such a framework, comparison of two sequences is not done by comparing two vectors each representing an entire sequence. Instead, these models first compare vector representations of smaller units such as words from these sequences and then aggregate these comparison results to make the final decision. For example, the match-LSTM model proposed by [86] for textual entailment first

compares each word in the hypothesis with an attention-weighted version of the premise. The comparison results are then aggregated through an LSTM. [33] proposed a pairwise word interaction model that first takes each pair of words from two sequences and applies a comparison unit on the two words. It then combines the results of these word interactions using a similarity focus layer followed by a multi-layer CNN. [58] proposed a decomposable attention model for textual entailment, in which words from each sequence are compared with an attention-weighted version of the other sequence to produce a series of comparison vectors. The comparison vectors are then aggregated and fed into a feed forward network for final classification.

Although these studies have shown the effectiveness of such a "compare-aggregate" framework for sequence matching, there are at least two limitations with these previous studies: (1) Each of the models proposed in these studies is tested on one or two tasks only, but we hypothesize that this general framework is effective on many sequence matching problems. There has not been any study that empirically verifies this. (2) More importantly, these studies did not pay much attention to the comparison function that is used to compare two small textual units. Usually a standard feedforward network is used [38, 86] to combine two vectors representing two units that need to be compared, e.g., two words. However, based on the nature of these sequence matching problems, we essentially need to measure how semantically similar the two sequences are. Presumably, this property of these sequence matching problems should guide us in choosing more appropriate comparison functions. Indeed [33] used cosine similarity, Euclidean distance and dot product to define the comparison function, which seem to be better justifiable. But they did not systematically evaluate these similarity or distance functions or compare them with a standard feedforward network.

In this chapter, we argue that the general "compare-aggregate" framework is effective for a wide range of sequence matching problems. We present a

model that follows this general framework and test it on four different datasets, namely, MovieQA, InsuranceQA, WikiQA and SNLI. The first three datasets are for Question Answering, but the setups of the tasks are quite different. The last dataset is for textual entailment. More importantly, we systematically present and test six different comparison functions. We find that overall a comparison function based on element-wise subtraction and multiplication works the best on the four datasets.

The contributions of this work are twofold: (1) Using four different datasets, we show that our model following the "compare-aggregate" framework is very effective when compared with the state-of-the-art performance on these datasets. (2) We conduct systematic evaluation of different comparison functions and show that a comparison function based on element-wise operations, which is not widely used for word-level matching, works the best across the different datasets. We believe that these findings will be useful for future research on sequence matching problems. We have also made our code available online.[1]

4.2 Method

In this section, we propose a general model following the "compare-aggregate" framework for matching two sequences. This general model can be applied to different tasks. We focus our discussion on six different comparison functions that can be plugged into this general "compare-aggregate" model. In particular, we hypothesize that two comparison functions based on element-wise operations, SUB and MULT, are good middle ground between highly flexible functions using standard neural network models and highly restrictive functions based on cosine similarity and/or Euclidean distance. As we will show in the experiment section, these comparison functions based on element-wise

[1] `https://github.com/shuohangwang/SeqMatchSeq`

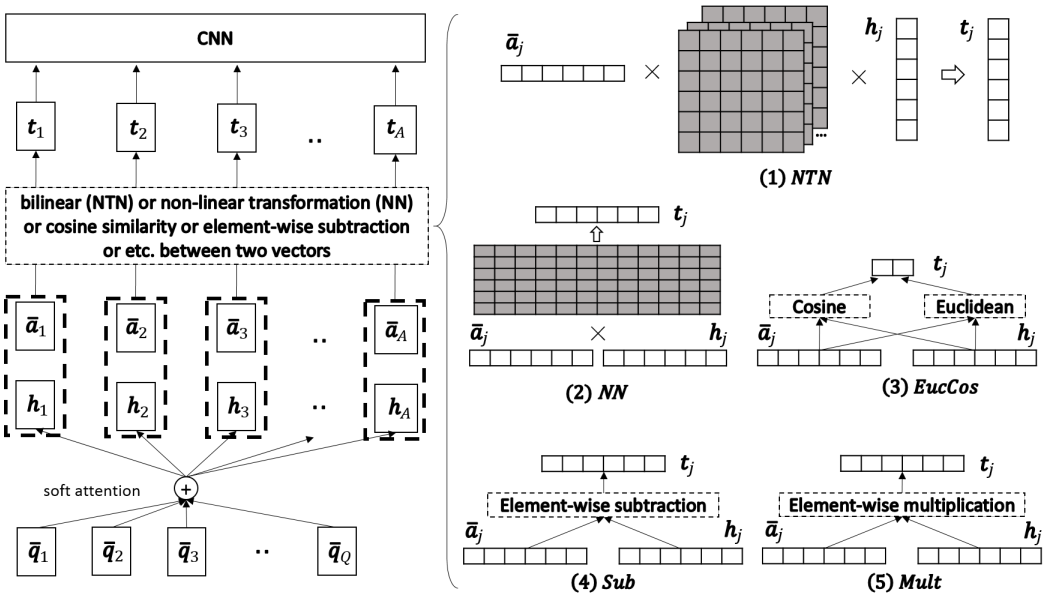

Figure 4.1: The left hand side is an overview of the model. The right hand side shows the details about the different comparison functions. The rectangles in dark represent parameters to be learned. $\times$ represents matrix multiplication.

operations can indeed perform very well on a number of sequence matching problems.

4.2.1 Problem Definition and Model Overview

The general setup of the sequence matching problem we consider is the following. We assume there are two sequences to be matched. We use two matrices $\mathbf{Q} \in \mathbb{R}^{d \times Q}$ and $\mathbf{A} \in \mathbb{R}^{d \times A}$ to represent the word embeddings of the two sequences, where Q and A are the lengths of the two sequences, respectively, and d is the dimensionality of the word embeddings. In other words, each column vector of $\mathbf{Q}$ or $\mathbf{A}$ is an embedding vector representing a single word. Given a pair of $\mathbf{Q}$ and $\mathbf{A}$, the goal is to predict a label y. For example, in textual entailment, $\mathbf{Q}$ may represent a premise and $\mathbf{A}$ a hypothesis, and y indicates whether $\mathbf{Q}$ entails $\mathbf{A}$ or contradicts $\mathbf{A}$. In question answering, $\mathbf{Q}$ may be a question and $\mathbf{A}$ a candidate answer, and y indicates whether $\mathbf{A}$ is the correct answer to $\mathbf{Q}$.

We treat the problem as a supervised learning task. We assume that a set

of training examples in the form of $(\mathbf{Q}, \mathbf{A}, y)$ is given and we aim to learn a model that maps any pair of $(\mathbf{Q}, \mathbf{A})$ to a y.

An overview of our model is shown in Figure 4.1. The model can be divided into the following four layers:

1. **Preprocessing:** We use a preprocessing layer (not shown in the figure) to process $\mathbf{Q}$ and $\mathbf{A}$ to obtain two new matrices $\overline{\mathbf{Q}} \in \mathbb{R}^{l \times Q}$ and $\overline{\mathbf{A}} \in \mathbb{R}^{l \times A}$. The purpose here is to use some gate values to control the importance of different words in making the predictions on the sequence pair. For example, $\overline{\mathbf{q}}_i \in \mathbb{R}^l$, which is the i^{th} column vector of $\overline{\mathbf{Q}}$, encodes the i^{th} word in $\mathbf{Q}$.

2. **Attention:** We apply a standard attention mechanism on $\overline{\mathbf{Q}}$ and $\overline{\mathbf{A}}$ to obtain attention weights over the column vectors in $\overline{\mathbf{Q}}$ for each column vector in $\overline{\mathbf{A}}$. With these attention weights, for each column vector $\overline{\mathbf{a}}_j$ in $\overline{\mathbf{A}}$, we obtain a corresponding vector $\mathbf{h}_j$, which is an attention-weighted sum of the column vectors of $\overline{\mathbf{Q}}$.

3. **Comparison:** We use a comparison function f to combine each pair of $\overline{\mathbf{a}}_j$ and $\mathbf{h}_j$ into a vector $\mathbf{t}_j$.

4. **Aggregation:** We use a CNN layer to aggregate the sequence of vectors $\mathbf{t}_j$ for the final classification.

Although this model follows more or less the same framework as the model proposed by [58], our work has some notable differences. First, we will pay much attention to the comparison function f and compare a number of options, including some uncommon ones based on element-wise operations. Second, we apply our model to four different datasets representing four different tasks to evaluate its general effectiveness for sequence matching problems. There are also some other differences from the work by [58]. For example, we use a CNN

layer instead of summation and concatenation for aggregation. Our attention mechanism is one-directional instead of two-directional.

In the rest of this section we will present the model in detail. We will focus mostly on the comparison functions we consider.

4.2.2 Preprocessing and Attention

Inspired by the use of gates in LSTM and GRU, we preprocess $\mathbf{Q}$ and $\mathbf{A}$ with the following formulas:

$$\begin{aligned} \overline{\mathbf{Q}} &= \sigma(\mathbf{W}^{\mathrm{i}}\mathbf{Q} + \mathbf{b}^{\mathrm{i}} \otimes \mathbf{e}_Q) \odot \tanh(\mathbf{W}^{\mathrm{u}}\mathbf{Q} + \mathbf{b}^{\mathrm{u}} \otimes \mathbf{e}_Q), \\ \overline{\mathbf{A}} &= \sigma(\mathbf{W}^{\mathrm{i}}\mathbf{A} + \mathbf{b}^{\mathrm{i}} \otimes \mathbf{e}_A) \odot \tanh(\mathbf{W}^{\mathrm{u}}\mathbf{A} + \mathbf{b}^{\mathrm{u}} \otimes \mathbf{e}_A), \end{aligned} \tag{4.1}$$

where $\odot$ is element-wise multiplication, and $\mathbf{W}^{\mathrm{i}}, \mathbf{W}^{\mathrm{u}} \in \mathbb{R}^{l \times d}$ and $\mathbf{b}^{\mathrm{i}}, \mathbf{b}^{\mathrm{u}} \in \mathbb{R}^{l}$ are parameters to be learned. The outer product $(\cdot \otimes \mathbf{e}_X)$ produces a matrix or row vector by repeating the vector or scalar on the left for X times. Here $\sigma(\mathbf{W}^{\mathrm{i}}\mathbf{Q} + \mathbf{b}^{\mathrm{i}} \otimes \mathbf{e}_Q)$ and $\sigma(\mathbf{W}^{\mathrm{i}}\mathbf{A} + \mathbf{b}^{\mathrm{i}} \otimes \mathbf{e}_A)$ act as gate values to control the degree to which the original values of $\mathbf{Q}$ and $\mathbf{A}$ are preserved in $\overline{\mathbf{Q}}$ and $\overline{\mathbf{A}}$. For example, for stop words, their gate values would likely be low for tasks where stop words make little difference to the final predictions.

In this preprocessing step, the word order does not matter. Although a better way would be to use RNN such as LSTM and GRU to chain up the words such that we can capture some contextual information, this could be computationally expensive for long sequences. In our experiments, we only incorporated LSTM into the formulas above for the SNLI task.

The general attention [49] layer is built on top of the resulting $\overline{\mathbf{Q}}$ and $\overline{\mathbf{A}}$ as follows:

$$\begin{aligned} \mathbf{G} &= \mathrm{softmax}\left((\mathbf{W}^{\mathrm{g}}\overline{\mathbf{Q}} + \mathbf{b}^{\mathrm{g}} \otimes \mathbf{e}_Q)^{\mathrm{T}}\overline{\mathbf{A}}\right), \\ \mathbf{H} &= \overline{\mathbf{Q}}\mathbf{G}, \end{aligned} \tag{4.2}$$

where $\mathbf{W}^{\mathrm{g}} \in \mathbb{R}^{l \times l}$ and $\mathbf{b}^{\mathrm{g}} \in \mathbb{R}^{l}$ are parameters to be learned, $\mathbf{G} \in \mathbb{R}^{Q \times A}$ is the attention weight matrix, and $\mathbf{H} \in \mathbb{R}^{l \times A}$ are the attention-weighted vectors. Specifically, $\mathbf{h}_j$, which is the j^{th} column vector of $\mathbf{H}$, is a weighted sum of the column vectors of $\overline{\mathbf{Q}}$ and represents the part of $\mathbf{Q}$ that best matches the j^{th} word in $\mathbf{A}$. Next we will combine $\mathbf{h}_j$ and $\overline{\mathbf{a}}_j$ using a comparison function.

4.2.3 Comparison

The goal of the comparison layer is to match each $\overline{\mathbf{a}}_j$, which represents the j^{th} word and its context in $\mathbf{A}$, with $\mathbf{h}_j$, which represents a weighted version of $\mathbf{Q}$ that best matches $\overline{\mathbf{a}}_j$. Let f denote a comparison function that transforms $\overline{\mathbf{a}}_j$ and $\mathbf{h}_j$ into a vector $\mathbf{t}_j$ to represent the comparison result.

A natural choice of f is a standard neural network layer that consists of a linear transformation followed by a non-linear activation function. For example, we can consider the following choice:

$$\textsc{NeuralNet (NN)}: \qquad \mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = \mathrm{ReLU}(\mathbf{W} \begin{bmatrix} \overline{\mathbf{a}}_j \\ \mathbf{h}_j \end{bmatrix} + \mathbf{b}), \quad (4.3)$$

where matrix $\mathbf{W} \in \mathbb{R}^{l \times 2l}$ and vector $\mathbf{b} \in \mathbb{R}^{l}$ are parameters to be learned.

Alternatively, another natural choice is a neural tensor network [68] as follows:

$$\textsc{NeuralTensorNet (NTN)}: \mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = \mathrm{ReLU}(\overline{\mathbf{a}}_j^{\mathrm{T}} \mathbf{T}^{[1\ldots l]} \mathbf{h}_j + \mathbf{b}), \quad (4.4)$$

where tensor $\mathbf{T}^{[1\ldots l]} \in \mathbb{R}^{l \times l \times l}$ and vector $\mathbf{b} \in \mathbb{R}^{l}$ are parameters to be learned.

However, we note that for many sequence matching problems, we intend to measure the semantic similarity or relatedness of the two sequences. So at the word level, we also intend to check how similar or related $\overline{\mathbf{a}}_j$ is to $\mathbf{h}_j$. For this reason, a more natural choice used in some previous work [] is Euclidean

distance or cosine similarity between $\overline{\mathbf{a}}_j$ and $\mathbf{h}_j$. We therefore consider the following definition of f:

$$\text{EUCLIDEAN+COSINE (EUCCOS):} \qquad \mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = \begin{bmatrix} \|\overline{\mathbf{a}}_j - \mathbf{h}_j\|_2 \\ \cos(\overline{\mathbf{a}}_j, \mathbf{h}_j) \end{bmatrix} \tag{4.5}$$

Note that with EUCCOS, the resulting vector $\mathbf{t}_j$ is only a 2-dimensional vector. Although EUCCOS is a well-justified comparison function, we suspect that it may lose some useful information from the original vectors $\overline{\mathbf{a}}_j$ and $\mathbf{h}_j$. On the other hand, NN and NTN are too general and thus do not capture the intuition that we care mostly about the similarity between $\overline{\mathbf{a}}_j$ and $\mathbf{h}_j$.

To use something that is a good compromise between the two extreme cases, we consider the following two new comparison functions, which operate on the two vectors in an element-wise manner. These functions have been used previously by [55].

$$\text{SUBTRACTION (SUB):} \qquad \mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = (\overline{\mathbf{a}}_j - \mathbf{h}_j) \odot (\overline{\mathbf{a}}_j - \mathbf{h}_j) \tag{4.6}$$

$$\text{MULTIPLICATION (MULT):} \qquad \mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = \overline{\mathbf{a}}_j \odot \mathbf{h}_j. \tag{4.7}$$

Note that the operator $\odot$ is element-wise multiplication. For both comparison functions, the resulting vector $\mathbf{t}_j$ has the same dimensionality as $\overline{\mathbf{a}}_j$ and $\mathbf{h}_j$.

We can see that SUB is closely related to Euclidean distance in that Euclidean distance is the sum of all the entries of the vector $\mathbf{t}_j$ produced by SUB. But by not summing up these entries, SUB preserves some information about the different dimensions of the original two vectors. Similarly, MULT is closely related to cosine similarity but preserves some information about the original two vectors.

Finally, we consider combining SUB and MULT followed by an NN layer as

follows:

$$\textsc{SubMult+NN}:\mathbf{t}_j = f(\overline{\mathbf{a}}_j, \mathbf{h}_j) = \text{ReLU}(\mathbf{W}\begin{bmatrix}(\overline{\mathbf{a}}_j - \mathbf{h}_j) \odot (\overline{\mathbf{a}}_j - \mathbf{h}_j) \\ \overline{\mathbf{a}}_j \odot \mathbf{h}_j\end{bmatrix} + \mathbf{b}) \quad (4.8)$$

In summary, we consider six different comparison functions: NN, NTN, EucCos, Sub, Mult and SubMult+NN. Among these functions, the last three (Sub, Mult and SubMult+NN) have not been widely used in previous work for word-level matching.

4.2.4 Aggregation

After we apply the comparison function to each pair of $\overline{\mathbf{a}}_j$ and $\mathbf{h}_j$ to obtain a series of vectors $\mathbf{t}_j$, finally we aggregate these vectors using a one-layer CNN [42]:

$$\mathbf{r} = \text{MaxPooling}\left(\text{CNN}\left([\mathbf{t}_1, \ldots, \mathbf{t}_A]\right)\right). \quad (4.9)$$

$\mathbf{r} \in \mathbb{R}^{nl}$ is then used for the final classification, where n is the number of windows in CNN.

4.3 Experiments

	MovieQA			InsuranceQA			WikiQA			SNLI		
	train	dev	test	train	dev	test	train	dev	test	train	dev	test
#Q	9848	1958	3138	13K	1K	1.8K*2	873	126	243	549K	9842	9824
#C	5	5	5	50	500	500	10	9	10	-	-	-
#w in P	873	866	914	-	-	-	-	-	-	-	-	-
#w in Q	10.6	10.6	10.8	7.2	7.2	7.2	6.5	6.5	6.4	14	15.2	15.2
#w in A	5.9	5.6	5.5	92.1	92.1	92.1	25.5	24.7	25.1	8.3	8.4	8.3

Table 4.2: The statistics of different datasets. Q:question/hypothesis, C:candidate answers for each question, A:answer/hypothesis, P:plot, w:word (average).

In this section, we evaluate our model on four different datasets representing different tasks. The first three datasets are question answering tasks while

Models	MovieQA		InsuranceQA			WikiQA		SNLI	
	dev	test	dev	test1	test2	MAP	MRR	train	test
Cosine Word2Vec	46.4	45.63	-	-	-	-	-	-	-
Cosine TFIDF	47.6	**47.36**	-	-	-	-	-	-	-
SSCB TFIDF	**48.5**	-	-	-	-	-	-	-	-
IR model	-	-	52.7	55.1	50.8	-	-	-	-
CNN with GESD	-	-	65.4	65.3	61.0	-	-	-	-
Attentive LSTM	-	-	68.9	69.0	64.8	-	-	-	-
IARNN-Occam	-	-	69.1	68.9	**65.1**	**0.7341**	**0.7418**	-	-
IARNN-Gate	-	-	**70.0**	**70.1**	62.8	0.7258	0.7394	-	-
CNN-Cnt	-	-	-	-	-	0.6520	0.6652	-	-
ABCNN	-	-	-	-	-	0.6921	0.7108	-	-
CubeCNN	-	-	-	-	-	0.7090	0.7234	-	-
W-by-W Attention	-	-	-	-	-	-	-	85.3	83.5
match-LSTM	-	-	-	-	-	-	-	92.0	86.1
LSTMN	-	-	-	-	-	-	-	88.5	86.3
Decomp Attention	-	-	-	-	-	-	-	90.5	86.8
EBIM+TreeLSTM	-	-	-	-	-	-	-	93.0	**88.3**
NN	31.6	-	76.8	74.9	72.4	0.7102	0.7224	89.3	86.3
NTN	31.6	-	75.6	75.0	72.5	0.7349	0.7456	91.6	86.3
EucCos	71.9	-	70.6	70.2	67.9	0.6740	0.6882	87.1	84.0
Sub	64.9	-	70.0	71.3	68.2	0.7019	0.7151	89.8	**86.8**
Mult	66.4	-	76.0	75.2	**73.4**	**0.7433**	**0.7545**	89.7	85.8
SubMult+NN	**72.1**	**72.9**	**77.0**	**75.6**	72.3	0.7332	0.7477	89.4	**86.8**

Table 4.3: Experiment Results

the last one is on textual entailment. The statistics of the four datasets are shown in Table 4.2. We will fist introduce the task settings and the way we customize the "compare-aggregate" structure to each task. Then we will show the baselines for the different datasets. Finally, we discuss the experiment results shown in Table 4.3 and the ablation study shown in Table 4.4.

4.3.1 Task-specific Model Structures

In all these tasks, we use matrix $\mathbf{Q} \in \mathbb{R}^{d \times Q}$ to represent the question or premise and matrix $\mathbf{A}_k \in \mathbb{R}^{d \times A_k}$ $(k \in [1, K])$ to represent the k^{th} answer or the hypothesis. For the machine comprehension task **MovieQA** [75], there is also a matrix $\mathbf{P} \in \mathbb{R}^{d \times P}$ that represents the plot of a movie. Here Q is the length of the question or premise, A_k the length of the k^{th} answer, and P the length of

Models	MovieQA		InsuranceQA			WikiQA		SNLI	
	dev	test	dev	test1	test2	MAP	MRR	train	test
SubMult+NN (no preprocess)	72.0	-	72.8	73.8	70.7	0.6996	0.7156	89.6	82.8
SubMult+NN (no attention)	60.4	-	69.4	70.4	67.8	0.7164	0.7238	89.0	84.4

Table 4.4: Ablation Experiment Results. "no preprocess": remove the preprocessing layer by directly using word embeddings $\mathbf{Q}$ and $\mathbf{A}$ to replace $\overline{\mathbf{Q}}$ and $\overline{\mathbf{A}}$ in Eqn. 4.1; "no attention": remove the attention layer by using mean pooling of $\overline{\mathbf{Q}}$ to replace all the vectors of $\mathbf{H}$ in Eqn. 5.2.

the plot.

For the **SNLI** [7] dataset, the task is text entailment, which identifies the relationship (entailment, contradiction or neutral) between a premise sentence and a hypothesis sentence. Here $K = 1$, and there are exactly two sequences to match. The actual model structure is what we have described before.

For the **InsuranceQA** [26] dataset, the task is an answer selection task which needs to select the correct answer for a question from a candidate pool. For the **WikiQA** [99] datasets, we need to rank the candidate answers according to a question. For both tasks, there are K candidate answers for each question. Let us use $\mathbf{r}_k$ to represent the resulting vector produced by Eqn. 4.9 for the k^{th} answer. In order to select one of the K answers, we first define $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_K]$. We then compute the probability of the k^{th} answer to be the correct one as follows:

$$p(k|\mathbf{R}) \quad = \quad \text{softmax}(\mathbf{w}^{\text{T}} \tanh(\mathbf{W}^{\text{s}}\mathbf{R} + \mathbf{b}^{\text{s}} \otimes \mathbf{e}_K) + b \otimes \mathbf{e}_K), \qquad (4.10)$$

where $\mathbf{W}^{\text{s}} \in \mathbb{R}^{l \times nl}$, $\mathbf{w} \in \mathbb{R}^l$, $\mathbf{b}^{\text{s}} \in \mathbb{R}^l$, $b \in \mathbb{R}$ are parameters to be learned.

For the machine comprehension task **MovieQA**, each question is related to Plot Synopses written by fans after watching the movie and each question has five candidate answers. So for each candidate answer there are three sequences to be matched: the plot $\mathbf{P}$, the question $\mathbf{Q}$ and the answer $\mathbf{A}_k$. For each k, we first match $\mathbf{Q}$ and $\mathbf{P}$ and refer to the matching result at position j as $\mathbf{t}^{\text{q}}_j$, as

generated by one of the comparison functions f. Similarly, we also match $\mathbf{A}_k$ with $\mathbf{P}$ and refer to the matching result at position j as $\mathbf{t}^{\mathrm{a}}_{k,j}$. We then define

$$\mathbf{t}_{k,j} = \begin{bmatrix} \mathbf{t}^{\mathrm{q}}_j \\ \mathbf{t}^{\mathrm{a}}_{k,j} \end{bmatrix},$$

and

$$\mathbf{r}_k = \mathrm{CNN}([\mathbf{t}_{k,1}, \ldots, \mathbf{t}_{k,P}]).$$

To select an answer from the K candidate answers, again we use Eqn. 4.10 to compute the probabilities.

The implementation details of the modes are as follows. The word embeddings are initialized from GloVe [59]. During training, they are not updated. The word embeddings not found in GloVe are initialized with zero.

The dimensionality l of the hidden layers is set to be 150. We use ADAMAX [43] with the coefficients $\beta_1 = 0.9$ and $\beta_2 = 0.999$ to optimize the model. We do not use L2-regularization. The main parameter we tuned is the dropout on the embedding layer. For WikiQA, which is a relatively small dataset, we also tune the learning rate and the batch size. For the others, we set the batch size to be 30 and the learning rate 0.002.

4.3.2 Baselines

Here, we will introduce the baselines for each dataset. We did not re-implement these models but simply took the reported performance for the purpose of comparison.

SNLI: • **W-by-W Attention**: The model by [64], who first introduced attention mechanism into text entailment. • **match-LSTM**: The model by [86], which concatenates the matched words as the inputs of an LSTM.

• **LSTMN**: Long short-term memory-networks proposed by [14]. • **Decomp Attention**: Another "compare-aggregate" model proposed by [58]. • **EBIM+TreeLSTM**: The state-of-the-art model proposed by [11] on the SNLI dataset.

InsuranceQA: • **IR model**: This model by [4] learns the concept information to help rank the candidates. • **CNN with GESD**: This model by [26] uses Euclidean distance and dot product between sequence representations built through convolutional neural networks to select the answer. • **Attentive LSTM**: [72] used soft-attention mechanism to select the most important information from the candidates according to the representation of the questions. • **IARNN-Occam**: This model by [84] adds regularization on the attention weights. • **IARNN-Gate**: This model by [84] uses the representation of the question to build the GRU gates for each candidate answer.

WikiQA: • **IARNN-Occam** and **IARNN-Gate** as introduced before. • **CNN-Cnt**: This model by [99] combines sentence representations built by a convolutional neural network with logistic regression. • **ABCNN**: This model is Attention-Based Convolutional Neural Network proposed by [102]. • **CubeCNN** proposed by [33] builds a CNN on all pairs of word similarity.

MovieQA: All the baselines we consider come from [75]'s work: • **Cosine Word2Vec**: A sliding window is used to select the answer according to the similarities computed through Word2Vec between the sentences in plot and the question/answer. • **Cosine TFIDF**: This model is similar to the previous method but uses bag-of-word with tf-idf scores to compute similarity. • **SSCB TFIDF**: Instead of using the sliding window method, a convolutional neural network is built on the sentence level similarities.

4.3.3 Analysis of Results

We use accuracy as the evaluation metric for the datasets MovieQA, InsuranceQA and SNLI, as there is only one correct answer or one label for each instance. For WikiQA, there may be multiple correct answers, so evaluation metrics we use are Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR).

We observe the following from the results. (1) Overall, we can find that our general "compare-aggregate" structure achieves the best performance on **MovieQA**, **InsuranceQA**, **WikiQA** datasets and very competitive performance on the **SNLI** dataset. Especially for the **InsuranceQA** dataset, with any comparison function we use, our model can outperform all the previous models. (2) The comparison method SubMult+NN is the best in general. (3) Some simple comparison functions can achieve better performance than the neural networks or neural tensor network comparison functions. For example, the simplest comparison function EucCos achieves nearly the best performance in the **MovieQA** dataset, and the element-wise comparison functions, which do not need parameters can achieve the best performance on the **WikiQA** dataset. (4) We find the preprocessing layer and the attention layer for word selection to be important in the "compare-aggregate" structure through the experiments of removing these two layers separately. We also see that for sequence matching with big difference in length, such as the **MovieQA** and **InsuranceQA** tasks, the attention layer plays a more important role. For sequence matching with smaller difference in length, such as the **WikiQA** and **SNLI** tasks, the pre-processing layer plays a more important role. (5) For the **MovieQA**, **InsuranceQA** and **WikiQA** tasks, our preprocessing layer is order-insensitive so that it will not take the context information into consideration during the comparison, but our model can still outperform the previous work with order-sensitive preprocessing layer. With this finding, we believe

the word-by-word comparison part plays a very important role in these tasks. We will further explore the preprocessing layer in the future.

4.3.4 Further Analyses

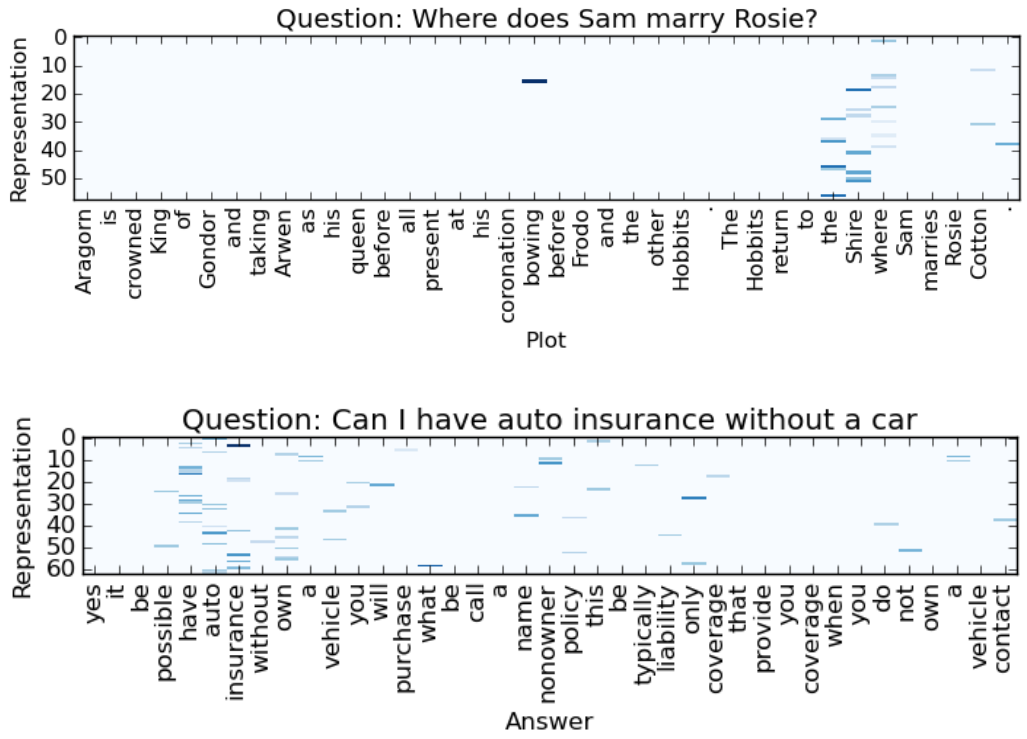

Figure 4.2: An visualization of the largest value of each dimension in the convolutional layer of CNN. The top figure is an example from the dataset **MovieQA** with CNN window size 5. The bottom figure is an example from the dataset **InsuranceQA** with CNN window size 3. Due to the sparsity of the representation, we show only the dimensions with larger values. The dimensionality of the raw representations is 150.

To further explain how our model works, we visualize the max values in each dimension of the convolutional layer. We use two examples shown in Table 4.1 from MovieQA and InsuranceQA datasets respectively. In the top of Figure 4.2, we can see that the plot words that also appear in either the question or the answer will draw more attention by the CNN. We hypothesize that if the nearby words in the plot can match both the words in question and the words in one answer, then this answer is more likely to be the correct

one. Similarly, the bottom one of Figure 4.2 also shows that the CNN will focus more on the matched word representations. If the words in one answer continuously match the words in the question, this answer is more likely to be the correct one.

4.4 Conclusions

In this chapter, we systematically analyzed the effectiveness of a "compare-aggregate" model on four different datasets representing different tasks. Moreover, we compared and tested different kinds of word-level comparison functions and found that some element-wise comparison functions can outperform the others. According to our experiment results, many different tasks can share the same "compare-aggregate" structure. In the future work, we would like to test its effectiveness on multi-task learning.

Part II

Machine Reading Comprehension with Textual Sequence Matching

Chapter 5

Machine Comprehension Using Match-LSTM and Answer Pointer

In this chapter, we will introduce our model for addressing task of machine reading comprehension, and will also show the effectiveness of textual sequence matching in the whole structure.

5.1 Introduction

Machine comprehension of text is one of the ultimate goals of natural language processing. While the ability of a machine to understand text can be assessed in many different ways, in recent years, several benchmark datasets have been created to focus on answering questions as a way to evaluate machine comprehension [62, 34, 36, 94, 61, 57]. In this setup, typically the machine is first presented with a piece of text such as a news article or a story. The machine is then expected to answer one or multiple questions related to the text.

In most of the benchmark datasets, a question can be treated as a multiple choice question, whose correct answer is to be chosen from a set of provided

In 1870, Tesla moved to Karlovac, **to attend school at the Higher Real Gymnasium**, where he was profoundly influenced by a math teacher **Martin Sekulić**. The classes were held in **German**, as it was a school within the Austro-Hungarian Military Frontier. Tesla was able to perform integral calculus in his head, which prompted his teachers to believe that he was cheating. He finished a four-year term in three years, graduating in 1873.	
1. In what language were the classes given?	**German**
2. Who was Tesla's main influence in Karlovac?	**Martin Sekulić**
3. Why did Tesla go to Karlovac?	**attend school at the Higher Real Gymnasium**

Table 5.1: A paragraph from Wikipedia and three associated questions together with their answers, taken from the SQuAD dataset. The tokens in bold in the paragraph are our predicted answers while the texts next to the questions are the ground truth answers.

candidate answers [62, 36]. Presumably, questions with more given candidate answers are more challenging. The Stanford Question Answering Dataset (SQuAD) introduced recently by [61] contains such more challenging questions whose correct answers can be any sequence of tokens from the given text. Moreover, unlike some other datasets whose questions and answers were created automatically in Cloze style [34, 36], the questions and answers in SQuAD were created by humans through crowdsourcing, which makes the dataset more realistic. Another real dataset, the Human-Generated MAchine Reading COmprehension dataset (MSMARCO) [57], provided a query together with several related documents collected from Bing Index. The answer to the query is generated by human and the answer words can not only come from the given text.

Given these advantages of the SQuAD and MSMARCO datasets, in this chapter, we focus on these new datasets to study machine comprehension of text. A sample piece of text and three of its associated questions from SQuAD are shown in Table 5.1. Traditional solutions to this kind of question answering tasks rely on NLP pipelines that involve multiple steps of linguistic analyses and feature engineering, including syntactic parsing, named entity recognition, question classification, semantic parsing, etc. Recently, with the

advances of applying neural network models in NLP, there has been much interest in building end-to-end neural architectures for various NLP tasks, including several pieces of work on machine comprehension [34, 36, 101, 41, 19]. However, given the properties of previous machine comprehension datasets, existing end-to-end neural architectures for the task either rely on the candidate answers [36, 101] or assume that the answer is a single token [34, 41, 19], which make these methods unsuitable for the SQuAD/MSMARCO dataset. In this chapter, we propose a new end-to-end neural architecture to address the machine comprehension problem as defined in the SQuAD/MSMARCO dataset. And for the MSMARCO dataset, we will only make use of the words in the given text to generate the answer.

Specifically, observing that in the SQuAD/MSMARCO dataset many questions could be entailed from some sentences in the original text, we adopt a match-LSTM model that we developed earlier for textual entailment [86] as one layer of our model. We build a bi-directional match-LSTM on the given passage with attentions on the question for each word so that each position in the paragraph will have a hidden representation reflecting its relation to the question. Then we further adopt the Pointer Net (Ptr-Net) model developed by [79] to select the words in these positions based on the hidden representations built by match-LSTM as an answer. We propose two ways to apply the Ptr-Net model for our task: a **sequence model** which selects the answer word by word, and a **boundary model** which only selects the start and end points of the answer span. Experiments on the SQuAD dataset show that our two models both outperform the best performance reported by [61]. Moreover, using an ensemble of several of our models, we can achieve very competitive performance on SQuAD. For the MSMARCO dataset, a real query based problem, our boundary model outperforms our sequence model with a big margin. It also outperforms the golden passage baseline.

Our contributions can be summarized as follows: (1) We propose two new

end-to-end neural network models for machine comprehension, which combine match-LSTM and Ptr-Net to handle the special properties of the SQuAD dataset. To the best of our knowledge, we are the first to propose the boundary model which is more suitable to the SQuAD/MSMARCO tasks. And we are the first to integrate the attention-based word pair matching into machine comprehension tasks. (2) We have achieved the performance of an exact match score of 71.3% and an F1 score of 80.8% on the unseen SQuAD test dataset, which is much better than the feature-engineered solution [61]. Our performance is also close to the state of the art on SQuAD, which is 74.8% in terms of exact match and 82.2% in terms of F1 collected from the SQuAD Leaderboard [1]. Besides, our boundary model achieves the state-of-art performance on the MSMARCO dataset with BLEU-1/2/3/4 40.7/33.9/30.6/28.7 and Rouge-L 37.3 [2]. (3) Our further visualization of the models reveals some useful insights of the attention mechanism for reasoning the questions. And we also show that the boundary model can overcome the early stop prediction problem in the sequence model. Besides, we also made our code available online [3].

5.2 Method

In this section, we first briefly review match-LSTM and Pointer Net. These two pieces of existing work lay the foundation of our method. We then present our end-to-end neural architecture for machine comprehension.

5.2.1 Match-LSTM

In a recent work on learning natural language inference, we proposed a match-LSTM model for predicting textual entailment [86]. In textual entailment, two

[1] https://rajpurkar.github.io/SQuAD-explorer/
[2] http://www.msmarco.org/leaders.aspx
[3] https://github.com/shuohangwang/SeqMatchSeq

sentences are given where one is a premise and the other is a hypothesis. To predict whether the premise entails the hypothesis, the match-LSTM model goes through the tokens of the hypothesis sequentially. At each position of the hypothesis, attention mechanism is used to obtain a weighted vector representation of the premise. This weighted premise is then to be combined with a vector representation of the current token of the hypothesis and fed into an LSTM, which we call the match-LSTM. The match-LSTM essentially sequentially aggregates the matching of the attention-weighted premise to each token of the hypothesis and uses the aggregated matching result to make a final prediction.

5.2.2 Pointer Net

[79] proposed a Pointer Network (Ptr-Net) model to solve a special kind of problems where we want to generate an output sequence whose tokens must come from the input sequence. Instead of picking an output token from a fixed vocabulary, Ptr-Net uses attention mechanism as a pointer to select a position from the input sequence as an output symbol. The pointer mechanism has inspired some recent work on language processing [32, 41]. Here we adopt Ptr-Net in order to construct answers using tokens from the input text.

5.2.3 Our Method

Formally, the problem we are trying to solve can be formulated as follows. We are given a piece of text, which we refer to as a passage, and a question related to the passage. The passage is represented by matrix $\mathbf{P} \in \mathbb{R}^{d \times P}$, where P is the length (number of tokens) of the passage and d is the dimensionality of word embeddings. Similarly, the question is represented by matrix $\mathbf{Q} \in \mathbb{R}^{d \times Q}$ where Q is the length of the question. Our goal is to identify a subsequence from the passage as the answer to the question.

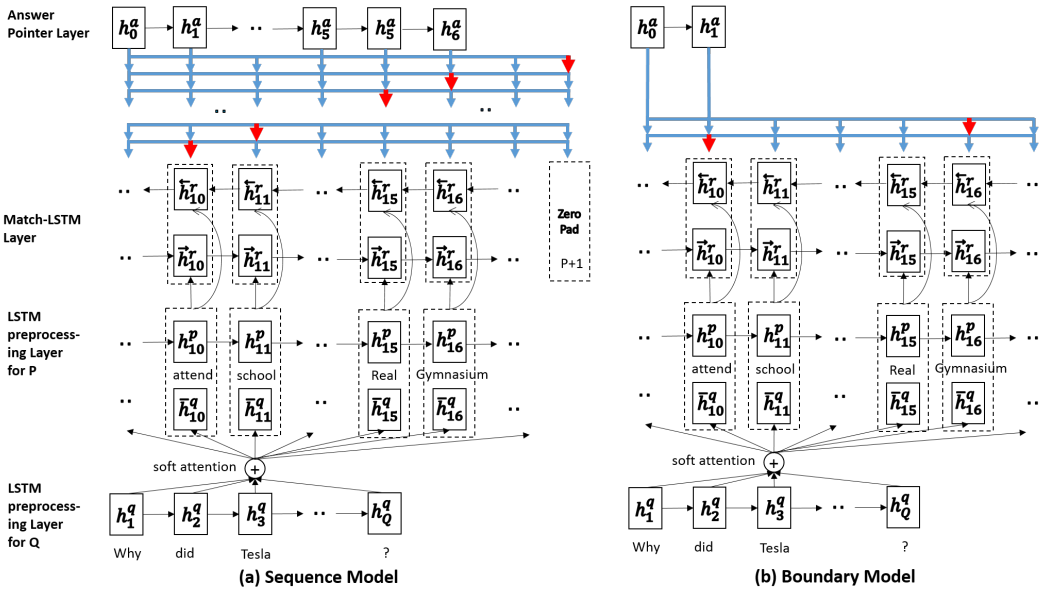

Figure 5.1: An overview of our two models. Both models consist of an LSTM preprocessing layer, a match-LSTM layer and an Answer Pointer layer. For each match-LSTM in a particular direction, $\bar{h}_i^{\mathrm{q}}$, which is defined as $\mathbf{H}^{\mathrm{q}}\alpha_i^{\intercal}$, is computed using the α in the corresponding direction, as described in Eqn. (5.2)

As pointed out earlier, since the output tokens are from the input, we would like to adopt the Pointer Net for this problem. A straightforward way of applying Ptr-Net here is to treat an answer as a sequence of tokens from the input passage but ignore the fact that these tokens are consecutive in the original passage, because Ptr-Net does not make the consecutivity assumption. Specifically, we represent the answer as a sequence of integers $\mathbf{a} = (a_1, a_2, \ldots)$, where each a_i is an integer between 1 and P, indicating a certain position in the passage.

Alternatively, if we want to ensure consecutivity, that is, if we want to ensure that we indeed select a subsequence from the passage as an answer, we can use the Ptr-Net to predict only the start and the end of an answer. In this case, the Ptr-Net only needs to select two tokens from the input passage, and all the tokens between these two tokens in the passage are treated as the answer. Specifically, we can represent the answer to be predicted as two integers $\mathbf{a} = (a_{\mathrm{s}}, a_{\mathrm{e}})$, where a_{s} an a_{e} are integers between 1 and P.

We refer to the first setting above as a *sequence* model and the second

setting above as a *boundary* model. For either model, we assume that a set of training examples in the form of triplets $\{(\mathbf{P}_n, \mathbf{Q}_n, \mathbf{a}_n)\}_{n=1}^{N}$ are given.

An overview of the two neural network models are shown in Figure 5.1. Both models consist of three layers: (1) An LSTM preprocessing layer that preprocesses the passage and the question using LSTMs. (2) A match-LSTM layer that tries to match the passage against the question. (3) An Answer Pointer (Ans-Ptr) layer that uses Ptr-Net to select a set of tokens from the passage as the answer. The difference between the two models only lies in the third layer.

LSTM Preprocessing Layer

The purpose for the LSTM preprocessing layer is to incorporate contextual information into the representation of each token in the passage and the question. We use a standard one-directional LSTM [37] to process the passage [4] and the question separately, as shown below:

$$\mathbf{H}^{\mathrm{p}} = \overrightarrow{LSTM}(\mathbf{P}), \quad \mathbf{H}^{\mathrm{q}} = \overrightarrow{LSTM}(\mathbf{Q}). \tag{5.1}$$

The resulting matrices $\mathbf{H}^{\mathrm{p}} \in \mathbb{R}^{l \times P}$ and $\mathbf{H}^{\mathrm{q}} \in \mathbb{R}^{l \times Q}$ are hidden representations of the passage and the question, where l is the dimensionality of the hidden vectors. In other words, the i^{th} column vector $\mathbf{h}_i^{\mathrm{p}}$ (or $\mathbf{h}_i^{\mathrm{q}}$) in $\mathbf{H}^{\mathrm{p}}$ (or $\mathbf{H}^{\mathrm{q}}$) represents the i^{th} token in the passage (or the question) together with some contextual information from the left.

Match-LSTM Layer

We apply the match-LSTM model [86] proposed for textual entailment to our machine comprehension problem by treating the question as a premise and the passage as a hypothesis. The match-LSTM sequentially goes through the passage. At position i of the passage, it first uses the standard word-by-word

[4] For the MSMARCO dataset, $\mathbf{P}$ is actually consisted of several unrelated documents. The previous state of pre-processing LSTM and match-LSTM to compute the first state of each document is set to zero.

attention mechanism to obtain attention weight vector $\overrightarrow{\alpha}_i \in \mathbb{R}^{1\times Q}$ as follows:

$$\begin{aligned}\overrightarrow{\mathbf{G}}_i &= \tanh(\mathbf{W}^{\mathrm{q}}\mathbf{H}^{\mathrm{q}} + (\mathbf{W}^{\mathrm{p}}\mathbf{h}_i^{\mathrm{p}} + \mathbf{W}^{\mathrm{r}}\overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}} + \mathbf{b}^{\mathrm{p}}) \otimes \mathbf{e}_Q),\\ \overrightarrow{\alpha}_i &= \mathrm{softmax}(\mathbf{w}^{\intercal}\overrightarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q),\end{aligned} \tag{5.2}$$

where $\mathbf{W}^{\mathrm{q}}, \mathbf{W}^{\mathrm{p}}, \mathbf{W}^{\mathrm{r}} \in \mathbb{R}^{l\times l}$, $\mathbf{b}^{\mathrm{p}}, \mathbf{w} \in \mathbb{R}^{l\times 1}$ and $b \in \mathbb{R}$ are parameters to be learned, $\overrightarrow{\mathbf{G}}_i \in \mathbb{R}^{l\times Q}$ is the intermediate result, $\overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}} \in \mathbb{R}^{l\times 1}$ is the hidden vector of the one-directional match-LSTM (to be explained below) at position $i-1$, and the outer product $(\cdot \otimes \mathbf{e}_Q)$ produces a matrix or row vector by repeating the vector or scalar on the left for Q times.

Essentially, the resulting attention weight $\overrightarrow{\alpha}_{i,j}$ above indicates the degree of matching between the i^{th} token in the passage with the j^{th} token in the question. Next, we use the attention weight vector $\overrightarrow{\alpha}_i$ to obtain a weighted version of the question and combine it with the current token of the passage to form a vector $\overrightarrow{\mathbf{z}}_i$:

$$\overrightarrow{\mathbf{z}}_i = \begin{bmatrix}\mathbf{h}_i^{\mathrm{p}}\\ \mathbf{H}^{\mathrm{q}}\overrightarrow{\alpha}_i^{\intercal}\end{bmatrix}, \tag{5.3}$$

where $\mathbf{H}^{\mathrm{q}} \in \mathbb{R}^{l\times Q}$, $\overrightarrow{\alpha}_i \in \mathbb{R}^{1\times Q}$ and $\mathbf{h}_i^{\mathrm{p}} \in \mathbb{R}^{l\times 1}$. This vector $\overrightarrow{\mathbf{z}}_i$ is fed into a standard one-directional LSTM to form our so-called match-LSTM:

$$\overrightarrow{\mathbf{h}}_i^{\mathrm{r}} = \overrightarrow{LSTM}(\overrightarrow{\mathbf{z}}_i, \overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}}), \tag{5.4}$$

where $\overrightarrow{\mathbf{h}}_i^{\mathrm{r}} \in \mathbb{R}^{l\times 1}$.

We further build a similar match-LSTM in the reverse direction. The purpose is to obtain a representation that encodes the contexts from both directions for each token in the passage.

Let $\overrightarrow{\mathbf{H}}^{\mathrm{r}} \in \mathbb{R}^{l\times P}$ represent the hidden states $[\overrightarrow{\mathbf{h}}_1^{\mathrm{r}}, \overrightarrow{\mathbf{h}}_2^{\mathrm{r}}, \ldots, \overrightarrow{\mathbf{h}}_P^{\mathrm{r}}]$ and $\overleftarrow{\mathbf{H}}^{\mathrm{r}} \in$

$\mathbb{R}^{l\times P}$ represent $[\overleftarrow{\mathbf{h}}^{\mathrm{r}}_1, \overleftarrow{\mathbf{h}}^{\mathrm{r}}_2, \ldots, \overleftarrow{\mathbf{h}}^{\mathrm{r}}_P]$, the hidden states of match-LSTM in the reverse direction. We define $\mathbf{H}^{\mathrm{r}} \in \mathbb{R}^{2l\times P}$ as the concatenation of the two:

$$\mathbf{H}^{\mathrm{r}} = \begin{bmatrix} \overrightarrow{\mathbf{H}}^{\mathrm{r}} \\ \overleftarrow{\mathbf{H}}^{\mathrm{r}} \end{bmatrix}. \tag{5.5}$$

Answer Pointer Layer

The top layer, the Answer Pointer (Ans-Ptr) layer, is motivated by the Pointer Net introduced by [79]. This layer uses the sequence $\mathbf{H}^{\mathrm{r}}$ as input. Recall that we have two different models: The *sequence* model produces a sequence of answer tokens but these tokens may not be consecutive in the original passage. The *boundary* model produces only the start token and the end token of the answer, and then all the tokens between these two in the original passage are considered to be the answer. We now explain the two models separately.

The Sequence Model: Recall that in the sequence model, the answer is represented by a sequence of integers $\mathbf{a} = (a_1, a_2, \ldots)$ indicating the positions of the selected tokens in the original passage. The Ans-Ptr layer models the generation of these integers in a sequential manner. Because the length of an answer is not fixed, in order to stop generating answer tokens at certain point, we allow each a_k to take up an integer value between 1 and $P+1$, where $P+1$ is a special value indicating the end of the answer. Once a_k is set to be $P+1$, the generation of the answer stops.

In order to generate the k^{th} answer token indicated by a_k, first, the attention mechanism is used again to obtain an attention weight vector $\beta_k \in \mathbb{R}^{1\times(P+1)}$, where $\beta_{k,j}$ $(1 \le j \le P+1)$ is the probability of selecting the j^{th} token from the passage as the k^{th} token in the answer, and $\beta_{k,(P+1)}$ is the probability of

stopping the answer generation at position k. β_k is modeled as follows:

$$\mathbf{F}_k = \tanh(\mathbf{V}\widetilde{\mathbf{H}}^{\mathrm{r}} + (\mathbf{W}^{\mathrm{a}}\mathbf{h}^{\mathrm{a}}_{k-1} + \mathbf{b}^a) \otimes \mathbf{e}_{(P+1)}), \tag{5.6}$$

$$\beta_k = \mathrm{softmax}(\mathbf{v}^\intercal \mathbf{F}_k + c \otimes \mathbf{e}_{(P+1)}), \tag{5.7}$$

where $\widetilde{\mathbf{H}}^{\mathrm{r}} \in \mathbb{R}^{2l\times(P+1)}$ is the concatenation of $\mathbf{H}^{\mathrm{r}}$ with a zero vector, defined as $\widetilde{\mathbf{H}}^{\mathrm{r}} = [\mathbf{H}^{\mathrm{r}}; \mathbf{0}]$, $\mathbf{V} \in \mathbb{R}^{l\times 2l}, \mathbf{W}^{\mathrm{a}} \in \mathbb{R}^{l\times l}$, $\mathbf{b}^a, \mathbf{v} \in \mathbb{R}^{l\times 1}$ and $c \in \mathbb{R}$ are parameters to be learned, $\mathbf{F}_k \in \mathbb{R}^{l\times(P+1)}$ is the intermediate result, $(\cdot \otimes \mathbf{e}_{(P+1)})$ follows the same definition as before, and $\mathbf{h}^{\mathrm{a}}_{k-1} \in \mathbb{R}^{l\times 1}$ is the hidden vector at position $k-1$ of an answer LSTM as defined below:

$$\mathbf{h}^{\mathrm{a}}_k = \overrightarrow{LSTM}(\widetilde{\mathbf{H}}^{\mathrm{r}}\beta_k^\intercal, \mathbf{h}^{\mathrm{a}}_{k-1}). \tag{5.8}$$

We can then model the probability of generating the answer sequence as

$$p(\mathbf{a}|\mathbf{H}^{\mathrm{r}}) = \prod_k p(a_k|a_1, a_2, \ldots, a_{k-1}, \mathbf{H}^{\mathrm{r}}), \tag{5.9}$$

and

$$p(a_k = j|a_1, a_2, \ldots, a_{k-1}, \mathbf{H}^{\mathrm{r}}) = \beta_{k,j}. \tag{5.10}$$

To train the model, we minimize the following loss function based on the training examples:

$$-\sum_{n=1}^{N} \log p(\mathbf{a}_n|\mathbf{P}_n, \mathbf{Q}_n). \tag{5.11}$$

The Boundary Model: The boundary model works in a way very similar to the sequence model above, except that instead of predicting a sequence of indices $a_1, a_2, \ldots$, we only need to predict two indices a_{s} and a_{e}. So the main difference from the sequence model above is that in the boundary model we

do not need to add the zero padding to $\mathbf{H}^{\mathrm{r}}$, and the probability of generating an answer is simply modeled as

$$p(\mathbf{a}|\mathbf{H}^{\mathrm{r}}) \quad = \quad p(a_{\mathrm{s}}|\mathbf{H}^{\mathrm{r}})p(a_{\mathrm{e}}|a_{\mathrm{s}}, \mathbf{H}^{\mathrm{r}}). \tag{5.12}$$

As this boundary model could point to a span covering too many tokens without any restriction, we try to manually limit the length of the predicted span and then search the span with the highest probability computed by $p(\mathbf{a}_s) \times p(\mathbf{a}_e|\mathbf{a}_s)$ as the answer.

5.3 Experiments

In this section, we present our experiment results and perform some analyses to better understand how our models works.

5.3.1 Data

We use the Stanford Question Answering Dataset (SQuAD) v1.1 and the human-generated Microsoft MAchine Reading COmprehension (MSMARCO) dataset v1.1 to conduct our experiments.

Passages in SQuAD come from 536 articles in Wikipedia covering a wide range of topics. Each passage is a single paragraph from a Wikipedia article, and each passage has around 5 questions associated with it. In total, there are 23,215 passages and 107,785 questions. The data has been split into a training set (with 87,599 question-answer pairs), a development set (with 10,570 question-answer pairs) and a hidden test set.

For the MSMARCO dataset, the questions are user queries issued to the Bing search engine, the context passages are real Web documents and the answers are human-generated. We select the span that has the highest F1 score with the gold standard answer for training and only predict the span in

the passages during evaluation. The data has been split into a training set (82326 pairs), a development set (10047 pairs) and a test set (9650 pairs).

5.3.2 Experiment Settings

We first tokenize all the passages, questions and answers. We use word embeddings from GloVe [59] to initialize the model. Words not found in GloVe are initialized as zero vectors. The word embeddings are not updated during the training of the model.

The dimensionality l of the hidden layers is set to be 150. We use ADAMAX [43] with the coefficients $\beta_1 = 0.9$ and $\beta_2 = 0.999$ to optimize the model. Each update is computed through a minibatch of 30 instances. We do not use L2-regularization.

For the SQuAD dataset, the performance is measured by two metrics: percentage of exact match with the ground truth answers and word-level F1 score when comparing the tokens in the predicted answers with the tokens in the ground truth answers. Note that in the development set and the test set each question has around three ground truth answers. F1 scores with the best matching answers are used to compute the average F1 score. For the MS-MARCO dataset, the metrics in the official tool of MSMARCO evaluation are BLEU-1/2/3/4 and Rouge-L, which are widely used in many domains.

5.3.3 Results

The SQuAD and MSMARCO results of our models as well as the results of the baselines [61, 104] are shown in Table 5.2. For the "LSTM with Ans-Ptr" models, they are the experiments with the ablation of attention mechanism in match-LSTM. Specifically, we use the final representation of the question to replace the weighted sum of the question representations. For the MSMARCO dataset, as the context for each question is consisted of around 10 documents,

	SQuAD				MSMARCO
	Exact Match		F1		BLEU1/2/3/4 / Rouge-L
	Dev	Test	Dev	Test	Dev & Test
Human	80.3	77.0	90.5	86.8	- & 46/-/-/- / 47
Golden Passage	-	-	-	-	19.6/18.8/18.1/17.5/32.3 & -
LR [61]	40.0	40.4	51.0	51.0	-
DCR [104]	62.5	62.5	71.2	71.0	-
LSTM with Ans-Ptr (Sequence)	37.7	-	48.5	-	10.3/7.2 /5.6 /4.6 /21.6 & -
LSTM with Ans-Ptr (Boundary)	45.2	-	55.3	-	32.0/25.3/22.2/20.4/32.3 & -
mLSTM with Ans-Ptr (Sequence)	54.4	-	68.2	-	12.5/9.2 /7.5 /6.5 /22.5 & -
mLSTM with Ans-Ptr (Boundary)	63.0	-	72.7	-	32.9/26.4/23.2/21.6/33.8 & -
Our best boundary model	**67.0**	**66.9**	**77.2**	**77.1**	**40.1/33.3/30.1/28.2/37.2** & **40.7/33.9/30.6/28.7/37.3**
mLSTM with Ans-Ptr (Boundary+en)	67.6	67.9	76.8	77.0	-
Our best boundary model (en)	**71.3**	**72.6**	**80.0**	**80.8**	-

Table 5.2: Experiment Results on SQuAD and MSMARCO datasets. Here "LSTM with Ans-Ptr" removes the attention mechanism in match-LSTM (mLSTM) by using the final state of the LSTM for the question to replace the weighted sum of all the states. Our best boundary model is the further tuned model and its ablation study is shown in Table 5.4. "en" refers to ensemble method.

the "Golden Passage" is to directly use the human labeled document which could answer the question as the prediction.

From the results in Table 5.2, we can see that the boundary model could clearly outperform the sequence model in a big margin on both datasets. We hypothesis that the sequence model is more likely to stop word generation earlier, and the boundary model can somehow overcome this problem. We have a statistical analysis on the answers generated by our sequence and boundary models shown in Table 5.3. We can see that the length of the answers generated by the sequence model is much shorter than the ground truth. Especially for the MSMARCO task where the answers are usually much longer, the sequence model could only generate 7 words on average, while the ground truth answers are 16 on average and the boundary model could generate nearly the same number of words with the ground truth. Several answers generated by our models are shown in Appendix A. From Table 5.2, we can also see that the performance gets poorer by removing the attention mechanism in match-

	SQuAD #w in A/Q/P	MSMARCO #w in A/Q/P
raw	3.1/11/141	16.3 / 6 / 667
seq	2.4/-/-	6.7 / - / -
bou	3.0/-/-	15.7 / - / -

Table 5.3: Statistical analysis on the development datasets. #w: number of words on average; P: passage; Q: question; A: answer; raw: raw data from the development dataset; seq/bou: the answers generated by the sequence/boundary models with match-LSTM.

	SQuAD EM & F1	MSMARCO BLEU1/2/3/4 & Rouge-L
Best model	67.0 & 77.2	40.1/33.3/30.1/28.2 & 37.2
-bi-Ans-Ptr	66.5 & 76.8	39.9/32.8/29.6/27.9 & 36.7
-deep	65.9 & 75.8	39.6/32.6/29.4/27.4 & 35.9
-elem	65.2 & 75.4	38.1/31.4/28.3/26.5 & 35.5
-pre-LSTM	64.0 & 72.9	39.6/32.8/29.8/27.7 & 36.3

Table 5.4: Ablation study for our best boundary model on the development datasets. Our best model is a further tuned boundary model by considering "bi-Ans-Ptr" which adds bi-directional answer pointer, "deep" which adds another two-layer bi-directional LSTMs between the match-LSTM and the Answer Pointer layers, and "elem" which adds element-wise comparison ,($\mathbf{h}_i^{\mathrm{p}} - \mathbf{H}^{\mathrm{q}}\alpha_i^{\intercal}$) and ($\mathbf{h}_i^{\mathrm{p}} \odot \mathbf{H}^{\mathrm{q}}\alpha_i^{\intercal}$), into Eqn 5.3. "-pre-LSTM" refers to removing the pre-processing layer.

LSTM, while for the MSMARCO dataset, the attention mechanism effects less, with no more than 2 percent reduction in BLEU and Rouge-L scores by attention mechanism ablation.

Based on the effectiveness of boundary pointer and match-LSTM, we conduct further exploration of the boundary model by adding element-wise comparison ($\mathbf{h}_i^{\mathrm{p}} - \mathbf{H}^{\mathrm{q}}\alpha_i^{\intercal}$) and ($\mathbf{h}_i^{\mathrm{p}} \odot \mathbf{H}^{\mathrm{q}}\alpha_i^{\intercal}$) into Eqn 5.3 in match-LSTM layer, adding 2 more bi-directional LSTM layers between match-LSTM and Ans-Ptr layers, and adding bi-directional Ans-Ptr. We show the ablation study of this further tuned model in Table 5.4. We can see that adding element-wise matching could make the biggest improvement for our boundary model. We also try to remove the phrase-level representation by removing the pre-process LSTM and using the word-level representations as the inputs of match-LSTM. Interestingly, we find the phrase-level representation effects little on the MSMARCO task.

Overall, we can see that both of our match-LSTM models have clearly outperformed the logistic regression model by [61], which relies on carefully designed features. The improvement of our models over the logistic regression model shows that our end-to-end neural network models without much feature

engineering are very effective on these tasks and datasets. Our boundary model also outperformed the DCR model [104], which maximizes the probability of the gold standard span from all the candidate spans through a neural network structure.

5.3.4 Further Analyses

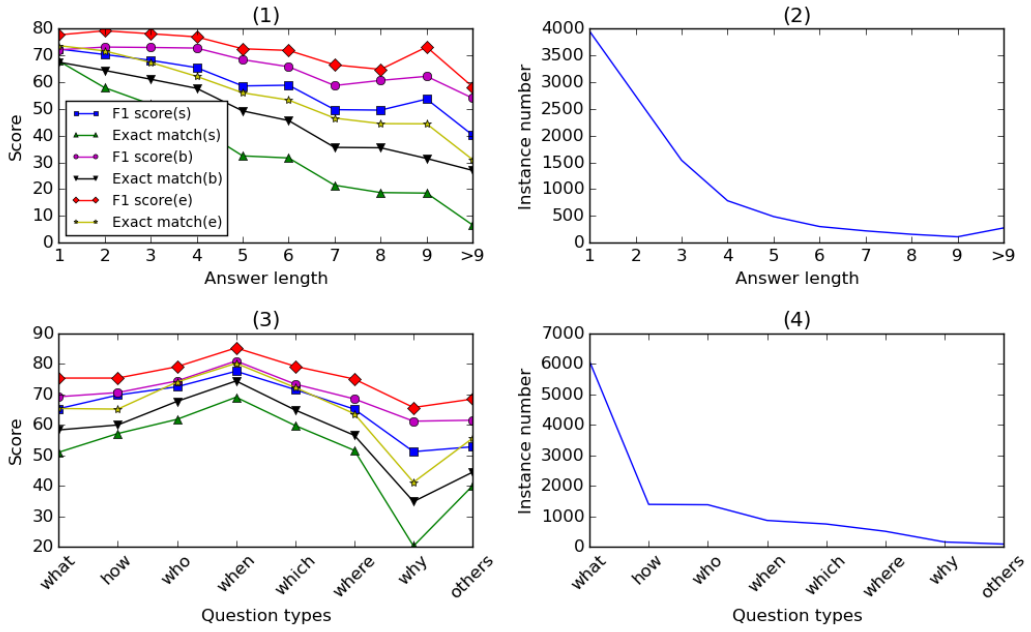

Figure 5.2: Performance breakdown by answer lengths and question types on SQuAD development dataset. Top: Plot (1) shows the performance of our two models (where s refers to the sequence model , b refers to the boundary model, and e refers to the ensemble boundary model) over answers with different lengths. Plot (2) shows the numbers of answers with different lengths. Bottom: Plot (3) shows the performance our the two models on different types of questions. Plot (4) shows the numbers of different types of questions.

To better understand the strengths and weaknesses of our models, we perform some further analyses of the results below.

First, we suspect that longer answers are harder to predict. To verify this hypothesis, we analysed the performance in terms of both exact match and F1 score with respect to the answer length on the development set, as shown in Figure 5.2. For example, for questions whose answers contain more than 9 tokens, the F1 score of the boundary model drops to around 55% and the

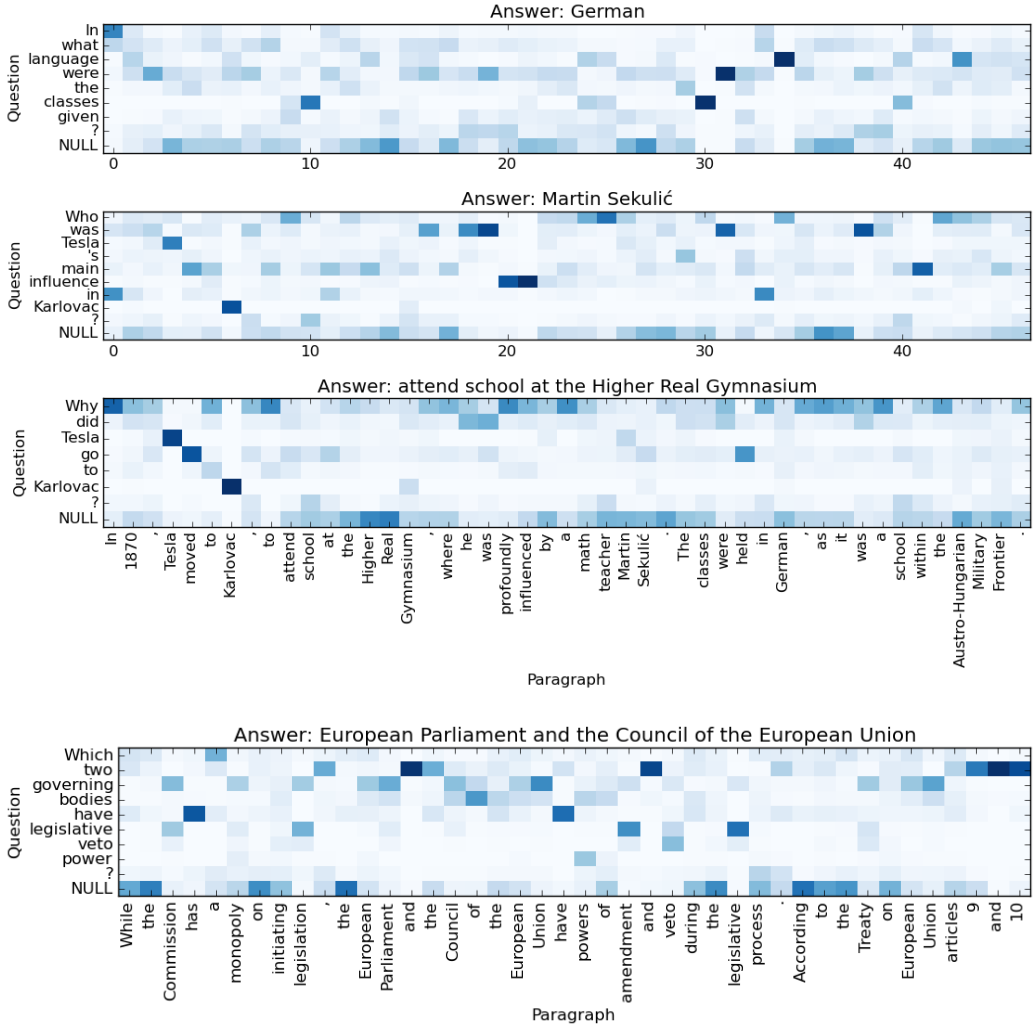

Figure 5.3: Visualization of the attention weights α for four questions. The first three questions share the same paragraph. The title is the answer predicted by our model.

exact match score drops to only around 30%, compared to the F1 score and exact match score of close to 72% and 67%, respectively, for questions with single-token answers. And that supports our hypothesis.

Next, we analyze the performance of our models on different groups of questions, as shown in Figure 5.2. We use a crude way to split the questions into different groups based on a set of question words we have defined, including "what," "how," "who," "when," "which," "where," and "why." These different question words roughly refer to questions with different types of answers. For example, "when" questions look for temporal expressions as answers, whereas

"where" questions look for locations as answers. According to the performance on the development dataset, our models work the best for "when" questions. This may be because in this dataset temporal expressions are relatively easier to recognize. Other groups of questions whose answers are noun phrases, such as "what" questions, "which" questions and "where" questions, also get relatively better results. On the other hand, "why" questions are the hardest to answer. This is not surprising because the answers to "why" questions can be very diverse, and they are not restricted to any certain type of phrases.

Finally, we would like to check whether the attention mechanism used in the match-LSTM layer is effective in helping the model locate the answer. We show the attention weights α in Figure 5.3. In the figure the darker the color is the higher the weight is. We can see that some words have been well aligned based on the attention weights. For example, the word "German" in the passage is aligned well to the word "language" in the first question, and the model successfully predicts "German" as the answer to the question. For the question word "who" in the second question, the word "teacher" actually receives relatively higher attention weight, and the model has predicted the phrase "Martin Sekulic" after that as the answer, which is correct. For the third question that starts with "why", the attention weights are more evenly distributed and it is not clear which words have been aligned to "why". For the last question, we can see that the word knowledge needed for generating the answer can also be detected by match-LSTM. For example, the words "European", "Parliament", "Council", "European" and "Union" have higher attention weights on "governing" in the question. Even though our models can solve this type of questions, they are still not able to solve the questions that need multi-sentences reasoning. More answers generated by our models for the questions related to different kinds of reasoning are shown in Appendix B.

5.4 Conclusions

In this chapter, We developed two models for the machine comprehension problem defined in the Stanford Question Answering (SQuAD) and A Human-Generated MAchine Reading COmprehension (MSMARCO) datasets, both making use of match-LSTM and Pointer Network. Experiments on the SQuAD and MSMARCO datasets showed that our second model, the boundary model, could achieve a performance close to the state-of-the-art performance on the SQuAD dataset and achieved the state-of-the-art on the MSMARCO dataset. We also show the boundary model could overcome the early stop prediction problem of the sequence model.

In the future, we plan to look further into the different types of questions and focus on those questions which currently have low performance, such as the "why' questions and multi-sentences related questions. We also plan to test how our models could be applied to other machine comprehension datasets.

Chapter 6

Multi-choice Reading Comprehension Using A Co-Matching Model

In this chapter, we will show how to make use of sequence matching model to solve the problem of Multi-choice Reading Comprehension, where question related passage and several candidate answers are given for each question, rather than extracting an answer span from the passage as previous chapter.

6.1 Introduction

Enabling machines to understand natural language text is arguably the ultimate goal of natural language processing, and the task of machine reading comprehension is an intermediate step towards this ultimate goal [62, 34, 36, 61, 57]. Recently, [46] released a new multi-choice machine comprehension dataset called RACE that was extracted from middle and high school English examinations in China. Figure 6.1 shows an example passage and two related questions from RACE. The key difference between RACE and previously released machine comprehension datasets (*e.g.*, the CNN/Daily Mail dataset [34]

and SQuAD [61]) is that the answers in RACE often cannot be directly extracted from the given passages, as illustrated by the two example questions (Q1 & Q2) in Figure 6.1. Thus, answering these questions is more challenging and requires more inferences.

Previous approaches to machine comprehension are usually based on pairwise sequence matching, where either the passage is matched against the sequence that concatenates both the question and a candidate answer [102], or the passage is matched against the question alone followed by a second step of selecting an answer using the matching result of the first step [46, 105]. However, these approaches may not be suitable for multi-choice reading comprehension since questions and answers are often equally important. Matching the passage only against the question may not be meaningful and may lead to loss of information from the original passage, as we can see from the first example question in Figure 6.1. On the other hand, concatenating the question and the answer into a single sequence for matching may not work, either, due to the loss of interaction information between a question and an answer. As illustrated by Q2 in Figure 6.1, the model may need to recognize what "he" and "it" in candidate answer (c) refer to in the question, in order to select (c) as the correct answer. This observation of the RACE dataset shows that we face a new challenge of matching sequence triplets (*i.e.*, passage, question and answer) instead of pairwise matching.

In this chapter, we propose a new model to match a question-answer pair to a given passage. Our *co-matching* approach explicitly treats the question and the candidate answer as two sequences and jointly matches them to the given passage. Specifically, for each position in the passage, we compute two attention-weighted vectors, where one is from the question and the other from the candidate answer. Then, two matching representations are constructed: the first one matches the passage with the question while the second one matches the passage with the candidate answer. These two newly

Passage: *My father wasn't a king, he was a taxi driver, but I am a prince-Prince Renato II, of the country Pontinha , an island fort on Funchal harbour. In 1903, the king of Portugal sold the land to a wealthy British family, the Blandys, who make Madeira wine. Fourteen years ago the family decided to sell it for just EUR25,000, but nobody wanted to buy it either. I met Blandy at a party and he asked if I'd like to buy the island. Of course I said yes, but I had no money-I was just an art teacher. I tried to find some business partners, who all thought I was crazy. So I sold some of my possessions, put my savings together and bought it. Of course, my family and my friends-all thought I was mad ... If l want to have a national flag, it could be blue today, red tomorrow. ... My family sometimes drops by, and other people come every day because the country is free for tourists to visit ...*	
Q1: *Which statement of the following is true?*	Q2: *How did the author get the island?*
a. *The author made his living by driving.*	a. *It was a present from Blandy.*
b. *The author's wife supported to buy the island.*	b. *The king sold it to him.*
c. *Blue and red are the main colors of his national flag.*	**c. *He bought it from Blandy.***
d. *People can travel around the island free of charge.*	d. *He inherited from his father.*

Table 6.1: An example passage and two related multi-choice questions. The ground-truth answers are in **bold**.

constructed matching representations together form a *co-matching state*. Intuitively, it encodes the locational information of the question and the candidate answer matched to a specific context of the passage. Finally, we apply a hierarchical LSTM [74] over the sequence of co-matching states at different positions of the passage. Information is aggregated from word-level to sentence-level and then from sentence-level to document-level. In this way, our model can better deal with the questions that require evidence scattered in different sentences in the passage. Our model improves the state-of-the-art model by 3 percentage on the RACE dataset. Our code will be released under `https://github.com/shuohangwang/comatch`.

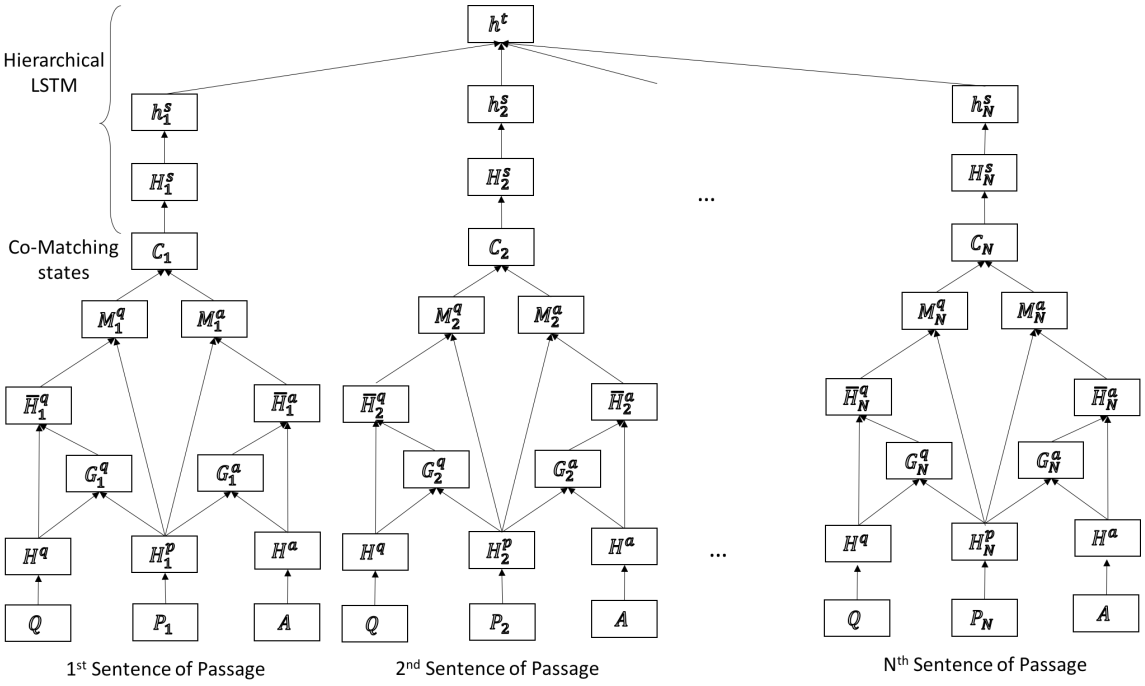

Figure 6.1: An overview of the model that builds a matching representation for a triplet $\{\mathbf{P}, \mathbf{Q}, \mathbf{A}\}$ (*i.e.*, passage, question and candidate answer).

6.2 Model

For the task of multi-choice reading comprehension, the machine is given a passage, a question and a set of candidate answers. The goal is to select the correct answer from the candidates. Let us use $\mathbf{P} \in \mathbb{R}^{d \times P}$, $\mathbf{Q} \in \mathbb{R}^{d \times Q}$ and $\mathbf{A} \in \mathbb{R}^{d \times A}$ to represent the passage, the question and a candidate answer, respectively, where each word in each sequence is represented by an embedding vector. d is the dimensionality of the embeddings, and P, Q, and A are the lengths of these sequences.

Overall our model works as follows. For each candidate answer, our model constructs a vector that represents the matching of $\mathbf{P}$ with both $\mathbf{Q}$ and $\mathbf{A}$. The vectors of all candidate answers are then used for answer selection. Because we simultaneously match $\mathbf{P}$ with $\mathbf{Q}$ and $\mathbf{A}$, we call this a *co-matching* model. In Section 6.2.1 we introduce the word-level co-matching mechanism. Then in Section 6.2.2 we introduce a hierarchical aggregation process. Finally in Section 6.2.3 we present the objective function. An overview of our co-matching

model is shown in Figure 6.1.

6.2.1 Co-matching

The co-matching part of our model aims to match the passage with the question and the candidate answer at the word-level. Inspired by some previous work [86, 77], we first use bi-directional LSTMs [37] to pre-process the sequences as follows:

$$\begin{aligned} \mathbf{H}^{\mathrm{p}} = \text{Bi-LSTM}(\mathbf{P}), \mathbf{H}^{\mathrm{q}} = \text{Bi-LSTM}(\mathbf{Q}), \\ \mathbf{H}^{\mathrm{a}} = \text{Bi-LSTM}(\mathbf{A}), \end{aligned} \tag{6.1}$$

where $\mathbf{H}^{\mathrm{p}} \in \mathbb{R}^{l \times P}$, $\mathbf{H}^{\mathrm{q}} \in \mathbb{R}^{l \times Q}$ and $\mathbf{H}^{\mathrm{a}} \in \mathbb{R}^{l \times A}$ are the sequences of hidden states generated by the bi-directional LSTMs. We then make use of the attention mechanism to match each state in the passage to an aggregated representation of the question and the candidate answer. The attention vectors are computed as follows:

$$\begin{aligned} \mathbf{G}^{\mathrm{q}} &= \text{SoftMax}\left((\mathbf{W}^{\mathrm{g}}\mathbf{H}^{\mathrm{q}} + \mathbf{b}^{\mathrm{g}} \otimes \mathbf{e}_Q)^{\mathrm{T}}\mathbf{H}^{\mathrm{p}}\right), \\ \mathbf{G}^{\mathrm{a}} &= \text{SoftMax}\left((\mathbf{W}^{\mathrm{g}}\mathbf{H}^{\mathrm{a}} + \mathbf{b}^{\mathrm{g}} \otimes \mathbf{e}_Q)^{\mathrm{T}}\mathbf{H}^{\mathrm{p}}\right), \\ \overline{\mathbf{H}}^{\mathrm{q}} &= \mathbf{H}^{\mathrm{q}}\mathbf{G}^{\mathrm{q}}, \\ \overline{\mathbf{H}}^{\mathrm{a}} &= \mathbf{H}^{\mathrm{a}}\mathbf{G}^{\mathrm{a}}, \end{aligned} \tag{6.2}$$

where $\mathbf{W}^{\mathrm{g}} \in \mathbb{R}^{l \times l}$ and $\mathbf{b}^{\mathrm{g}} \in \mathbb{R}^{l}$ are the parameters to learn. $e_Q \in \mathbb{R}^{\mathrm{Q}}$ is a vector of all 1s and it is used to repeat the bias vector into the matrix. $\mathbf{G}^{\mathrm{q}} \in \mathbb{R}^{Q \times P}$ and $\mathbf{G}^{\mathrm{a}} \in \mathbb{R}^{A \times P}$ are the attention weights assigned to the different hidden states in the question and the candidate answer sequences, respectively. $\overline{\mathbf{H}}^{\mathrm{q}} \in \mathbb{R}^{l \times P}$ is the weighted sum of all the question hidden states and it represents how the question can be aligned to each hidden state in the passage. So is $\overline{\mathbf{H}}^{\mathrm{a}} \in \mathbb{R}^{l \times P}$. Finally we can co-match the passage states with the question and the candidate

answer as follows:

$$\begin{aligned}
\mathbf{M}^{\mathrm{q}} &= \mathrm{ReLU}\left(\mathbf{W}^{\mathrm{m}}\begin{bmatrix}\overline{\mathbf{H}}^{\mathrm{q}} \ominus \mathbf{H}^{\mathrm{p}} \\ \overline{\mathbf{H}}^{\mathrm{q}} \otimes \mathbf{H}^{\mathrm{p}}\end{bmatrix} + \mathbf{b}^{\mathrm{m}}\right), \\
\mathbf{M}^{\mathrm{a}} &= \mathrm{ReLU}\left(\mathbf{W}^{\mathrm{m}}\begin{bmatrix}\overline{\mathbf{H}}^{\mathrm{a}} \ominus \mathbf{H}^{\mathrm{p}} \\ \overline{\mathbf{H}}^{\mathrm{a}} \otimes \mathbf{H}^{\mathrm{p}}\end{bmatrix} + \mathbf{b}^{\mathrm{m}}\right), \\
\mathbf{C} &= \begin{bmatrix}\mathbf{M}^{\mathrm{q}} \\ \mathbf{M}^{\mathrm{a}}\end{bmatrix}, \qquad (6.3)
\end{aligned}$$

where $\mathbf{W}^{\mathrm{g}} \in \mathbb{R}^{l \times 2l}$ and $\mathbf{b}^{\mathrm{g}} \in \mathbb{R}^{l}$ are the parameters to learn. $\begin{bmatrix}\cdot \\ \cdot\end{bmatrix}$ is the column-wise concatenation of two matrices, and $\cdot \ominus \cdot$ and $\cdot \otimes \cdot$ are the element-wise subtraction and multiplication between two matrices, which are used to build better matching representations [71, 87]. $\mathbf{M}^{\mathrm{q}} \in \mathbb{R}^{l \times P}$ represents the matching between the hidden states of the passage and the corresponding attention-weighted representations of the question. Similarly, we match the passage with the candidate answer and represent the matching results using $\mathbf{M}^{\mathrm{a}} \in \mathbb{R}^{l \times P}$. Finally $C \in \mathbb{R}^{2l \times P}$ is the concatenation of $\mathbf{M}^{\mathrm{q}} \in \mathbb{R}^{l \times P}$ and $\mathbf{M}^{\mathrm{a}} \in \mathbb{R}^{l \times P}$ and represents how each passage state can be matched with the question and the candidate answer. We refer to $\mathbf{c} \in \mathbb{R}^{2l}$, which is a single column of $\mathbf{C}$, as a *co-matching state* that concurrently matches a passage state with both the question and the candidate answer.

6.2.2 Hierarchical Aggregation

In order to capture the sentence structure of the passage, we further modify the model presented earlier and build a hierarchical LSTM [74] on top of the co-matching states. Specifically, we first split the passage into sentences and we use $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_N$ to represent these sentences, where N is the number of sentences in the passage. For each triplet $\{\mathbf{P}_n, \mathbf{Q}, \mathbf{A}\}, n \in [1, N]$, we can

get the co-matching states $\mathbf{C}_n$ through Eqn. (6.1-6.3). Then we build a bi-directional LSTM followed by max pooling on top of the co-matching states of each sentence as follows:

$$\mathbf{h}_n^{\mathrm{s}} = \mathrm{MaxPooling}\left(\text{Bi-LSTM}\left(\mathbf{C}_n\right)\right), \tag{6.4}$$

where the function MaxPooling$(\cdot)$ is the row-wise max pooling operation. $\mathbf{h}_n^{\mathrm{s}} \in \mathbb{R}^l, n \in [1, N]$ is the sentence-level aggregation of the co-matching states. All these representations will be further integrated by another Bi-LSTM to get the final triplet matching representation.

$$\begin{aligned} \mathbf{H}^{\mathrm{s}} &= [\mathbf{h}_1^s; \mathbf{h}_2^s; \ldots; \mathbf{h}_N^s], \\ \mathbf{h}^{\mathrm{t}} &= \mathrm{MaxPooling}\left(\text{Bi-LSTM}\left(\mathbf{H}^{\mathrm{s}}\right)\right), \end{aligned} \tag{6.5}$$

where $\mathbf{H}^{\mathrm{s}} \in \mathbb{R}^{l \times N}$ is the concatenation of all the sentence-level representations and it is the input of a higher level LSTM. $\mathbf{h}^{\mathrm{t}} \in \mathbb{R}^l$ is the final output of the matching between the sequences of the passage, the question and the candidate answer.

6.2.3 Objective function

For each candidate answer $\mathbf{A}_i$, we can build its matching representation $\mathbf{h}_i^t \in \mathbb{R}^l$ with the question and the passage through Eqn. (6.5). Our loss function is computed as follows:

$$L(\mathbf{A}_i|\mathbf{P}, \mathbf{Q}) = -\log \frac{\exp(\mathbf{w}^T \mathbf{h}_i^t)}{\sum_{j=1}^{4} \exp(\mathbf{w}^T \mathbf{h}_j^t)}, \tag{6.6}$$

where $\mathbf{w} \in \mathbb{R}^l$ is a parameter to learn.

	RACE-M	RACE-H	RACE
Random	24.6	25.0	24.9
Sliding Window	37.3	30.4	32.2
Stanford AR	44.2	43.0	43.3
GA	43.7	44.2	44.1
ElimiNet	-	-	44.7
HAF	45.3	47.9	47.2
MUSIC	51.5	45.7	47.4
Hier-Co-Matching	**55.8***	**48.2***	**50.4***
- Hier-Aggregation	54.2	46.2	48.5
- Co-Matching	50.7	45.6	46.4
Turkers	85.1	69.4	73.3
Ceiling	95.4	94.2	94.5

Table 6.2: Experiment Results. * means it's significant to the models ablating either the hierarchical aggregation or co-matching state.

6.3 Experiment

To evaluate the effectiveness of our hierarchical co-matching model, we use the RACE dataset [46], which consists of two subsets: RACE-M comes from middle school examinations while RACE-H comes from high school examinations. RACE is the combination of the two.

We compare our model with a number of baseline models. We also compare with two variants of our model for an ablation study.

Comparison with Baselines We compare our model with the following baselines:

• **Sliding Window** based method [62] computes the matching score based on the sum of the tf-idf values of the matched words between the question-answer pair and each sub-passage with a fixed a window size.

• **Stanford Attentive Reader (AR)** [9] first builds a question-related passage representation through attention mechanism and then compares it with each candidate answer representation to get the answer probabilities.

• **GA** [22] uses gated attention mechanism with multiple hops to extract the question-related information of the passage and compares it with candidate

answers.

• **ElimiNet** [69] tries to first eliminate the most irrelevant choices and then select the best answer.

• **HAF** [105] considers not only the matching between the three sequences, namely, passage, question and candidate answer, but also the matching between the candidate answers.

• **MUSIC** [98] integrates different sequence matching strategies into the model and also adds a unit of multi-step reasoning for selecting the answer.

Besides, we also report the following two results as reference points: **Turkers** is the performance of Amazon Turkers on a randomly sampled subset of the RACE test set. **Ceiling** is the percentage of the unambiguous questions with a correct answer in a subset of the test set.

The performance of our model together with the baselines are shown in Table 6.2. We can see that our proposed complete model, **Hier-Co-Matching**, achieved the best performance among all the public results. Still, there is a huge gap between the best machine reading performance and the human performance, showing the great potential for further research.

Ablation Study Moreover, we conduct an ablation study of our model architecture. In this study, we are mainly interested in the contribution of each component introduced in this work to our final results. We studied two key factors: (1) the co-matching module and (2) the hierarchical aggregation approach. We observed a 4 percentage performance decrease by replacing the co-matching module with a single matching state (*i.e.*, only $\mathbf{M}^a$ in Eqn (5.3)) by directly concatenating the question with each candidate answer [102]. We also observe about 2 percentage decrease when we treat the passage as a plain sequence, and run a two-layer LSTM (to ensure the numbers of parameters are comparable) over the whole passage instead of the hierarchical LSTM.

Question Type Analysis We also conducted an analysis on what types of questions our model can handle better. We find that our model obtains similar performance on the "wh" questions such as "why," "what," "when" and "where" questions, on which the performance is usually around 50%. We also check statement-justification questions with the keyword "true" (*e.g.*, "Which of the following statements is true"), negation questions with the keyword "not" (*e.g.*, "which of the following is not true"), and summarization questions with the keyword "title" (*e.g.*, "what is the best title for the passage?"), and their performance is 51%, 52% and 48%, respectively. We can see that the performance of our model on different types of questions in the RACE dataset is quite similar. However, our model is only based on word-level matching and may not have the ability of reasoning. In order to answer questions that require summarization, inference or reasoning, we still need to further explore the dataset and improve the model. Finally, we further compared our model to the baseline, which concatenates the question with each candidate answer, and our model can achieve better performance on different types of questions. For example, on the subset of the questions with pronouns, our model can achieve better accuracy of 49.8% than 47.9%. Similarly, on statement-justification questions with the keyword "true", our model could achieve better accuracy of 51% than 47%.

6.4 Conclusions

In this chapter, we proposed a co-matching model for multi-choice reading comprehension. The model consists of a co-matching component and a hierarchical aggregation component. We showed that our model could achieve state-of-the-art performance on the RACE dataset. In the future, we will adapt the idea of co-matching and hierarchical aggregation to the standard open-domain QA setting for answer candidate reranking [91]. We will also further study how to

explicitly model inference and reasoning on the RACE dataset.

Part III

Open-domain Question Answering with Textual Sequence Matching Model

Chapter 7

R^3: Reinforced Ranker-Reader for Open-Domain Question Answering

From this chapter, we will go into another question answering task: open-domain QA, where the passages are not pre-selected any more and needs information retrieval model to first search some passages for answer extraction. As the answer extraction part is still related to the model for reading comprehension task, our models for open-domain QA also rely on the sequence matching model.

7.1 Introduction

Open-domain question answering (QA) is a key challenge in natural language processing. A successful open-domain QA system must be able to effectively retrieve and comprehend one or more knowledge sources to infer a correct answer. Knowledge sources can be knowledge bases [5, 103] or structured or unstructured text passages [27, 3].

Recent deep learning-based research has focused on open-domain QA

Q:	What is the largest island in the Philippines?
A:	**Luzon**
P1	Mindanao is the second largest and easternmost island in the Philippines.
P2	As an island, **Luzon** is the Philippine's largest at 104,688 square kilometers, and is also the world's 17th largest island.
P3	Manila, located on east central **Luzon** Island, is the national capital and largest city.

Table 7.1: An open-domain QA training example. Q: question, A: answer, P: passages retrieved by an IR model and ordered by IR score.

based on large text corpora such as Wikipedia, applying information retrieval (IR) to select passages and reading comprehension (RC) to extract answer phrases [10, 23]. These methods, which we call *Search-and-Reading QA* (SR-QA), are simple yet powerful for open-domain QA. Dividing the pipeline into IR and RC stages leverages an enormous body of research in both IR and RC, including recent successes in RC via neural network techniques [88, 93, 97, 92].

The main difference between training SR-QA and standard RC models is in the passages used for training. In standard RC model training, passages are manually selected to guarantee that ground-truth answers are contained and annotated within the passage [61].[1] By contrast, in SR-QA approaches [10, 23], the model is given only QA-pairs and uses an IR component to retrieve passages similar to the question from a large corpus. Depending on the quality of the IR component, retrieved passages may not contain or entail the correct answer, making RC training more difficult. Table 1 shows an example which illustrates the difficulty. This ordering was produced by an off-the-shelf IR engine using the BM25 algorithm. The correct answer is contained in passage P2. The top passage (P1), despite being ranked highest by the IR engine, is ineffective for answering the question, since it fails to capture the semantic distinction between "largest" and "second largest". Passage P3 contains the answer text ("Luzon") but does not semantically entail the correct answer ("Luzon is the

[1]This forms a closed-domain QA by our adopted definition where the domain consists of the given passage only.

largest island in the Philippines"). Training on passages such as P1 and P3 can degrade performance of the RC component.[2]

In this chapter we propose a new approach which explicitly separates the tasks of predicting the likelihood that a passage provides the answer, and reading those passages to extract correct answers. Specifically we propose an end-to-end framework consisting of two components: a *Ranker* and a *Reader* (i.e. RC model). The Ranker selects the passage most likely to entail the answer and passes it to the Reader, which reads and extracts from that passage. The Reader is trained using SGD/backprop to maximize the likelihood of the span containing the correct answer (if one exists). The Ranker is trained using REINFORCE [96] with a reward determined by how well the Reader extracts answers from the top-ranked passages. This optimizes the Ranker with an objective determined by end-performance on answer prediction, which provides a strong signal to distinguish passages lexically similar to but semantically different from the question.

We discuss the Ranker-Reader model in detail below but briefly, the Ranker and Reader are implemented as variants of Match-LSTM models [86]. These models were originally designed for solving the text entailment problem. For this task, different non-linear layers are added for selecting the passages or predicting the start and end positions of the answer in the passage.

We evaluate our model on five different datasets and achieve state-of-the-art results on four of the them. Our results also show the merits of employing a separate REINFORCE-trained ranking component over several challenging fully supervised baselines.

[2]Passage ranking models for non-factoid QA [85, 99] are able to learn to rank these passages; but these models are trained using human annotated answer labels, which are not available here.

7.2 Framework

Problem Definition We assume that we have available a factoid question $\mathbf{q}$ to be answered and a set of passages which may contain the ground-truth answer $\mathbf{a}^g$. Those passages[3] are the top N retrieved from a corpus by an IR model supplied with the question, for N a hyper-parameter. During training we are given only the $(\mathbf{q}, \mathbf{a}^g)$ pairs, together with an IR model with index built on an open-domain corpus.

Framework Overview An overview of the Ranker-Reader model is shown in Figure 7.1. It shows two key components: a **Ranker**, which selects passages from which an answer can be extracted, and a **Reader** which extracts answers from supplied passages. Both the Ranker and Reader compare the question to each of the passages to generate passage representations based on how well they match the question. The Ranker uses these "matched" representations to select a single passage which is most likely to contain the answer. The selected passage is then processed by the Reader to extract an answer sequence. We train the reader using SGD/backprop and produce a reward to train the Ranker via REINFORCE.

7.3 R^3: Reinforced Ranker-Reader

In this section, we first review the Match-LSTM [86] which provides input for both the Reader and Ranker. We then detail the Reader and Ranker components, and the procedure for joint training, including the objective function used for RL training.

[3]In this chapter we use sentence-level index thus each passage is an individual sentence. See the experimental setting.

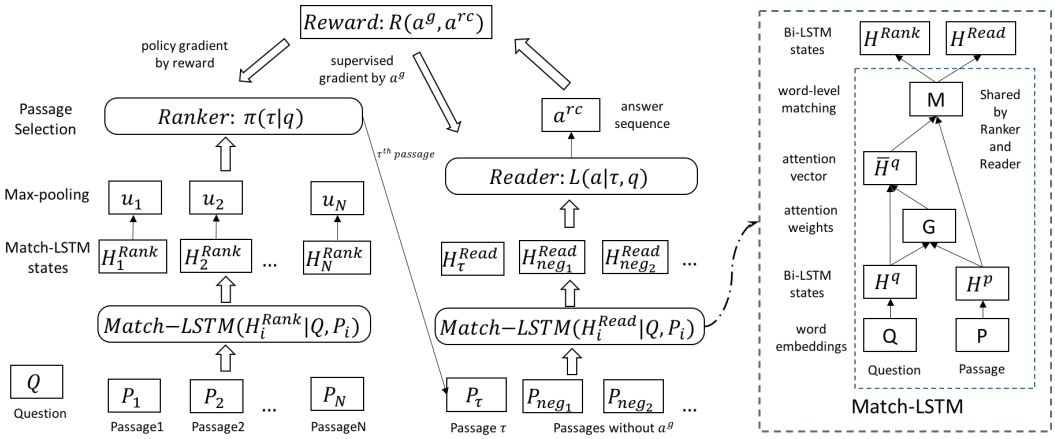

Figure 7.1: Overview of training our model, comprising a Ranker and a Reader based on Match-LSTM as shown on the right side. The Ranker selects a passage τ and the Reader predicts the start and end positions of the answer in τ. The reward for the Ranker depends on similarity of the extracted answer with the ground-truth answer $\mathbf{a}^g$. To accelerate Reader convergence, we also sample several negative passages without ground-truth answer.

Passage Representation Using Match-LSTM To effectively rank and read passages they must be matched to the question. This comparison is performed with a Match-LSTM, a state-of-the-art model for text entailment, shown on the right in Figure 7.1. Match-LSTMs use an attention mechanism to compute word similarities between the passage and question sequences. These are first encoded as matrices $\mathbf{Q}$ and $\mathbf{P}$, respectively, by a Bidirectional LSTM (BiLSTM) with hidden dimension l. With Q words in question $\mathbf{Q}$ and P words in passage $\mathbf{P}$ we can write:

$$\mathbf{H}^{\mathrm{p}} = \mathrm{BiLSTM}(\mathbf{P}), \quad \mathbf{H}^{\mathrm{q}} = \mathrm{BiLSTM}(\mathbf{Q}), \tag{7.1}$$

where $\mathbf{H}^{\mathrm{p}} \in \mathbb{R}^{l \times P}$ and $\mathbf{H}^{\mathrm{q}} \in \mathbb{R}^{l \times Q}$ are the hidden states for the passage and the question. In order to improve computational efficiency without degrading performance, we simplify the attention mechanism of the original Match-LSTM by computing the attention weights $\mathbf{G}$ as follows:

$$\mathbf{G} \quad = \quad \mathrm{SoftMax}\left((\mathbf{W}^{\mathrm{g}}\mathbf{H}^{\mathrm{q}} + \mathbf{b}^{\mathrm{g}} \otimes \mathbf{e}_Q)^{\mathrm{T}}\mathbf{H}^{\mathrm{p}}\right)$$

where $\mathbf{W}^{\mathrm{g}} \in \mathbb{R}^{l \times l}$ and $\mathbf{b}^{\mathrm{g}} \in \mathbb{R}^{l}$ are the learnable parameters. The outer product $(\cdot \otimes \mathbf{e}_Q)$ repeats the column vector $\mathbf{b}^{\mathrm{g}}$ Q times to form an $l \times l$ matrix. The i-th column of $\mathbf{G} \in \mathbb{R}^{Q \times P}$ represents the normalized attention weights over all the question words for the i-th word in passage.

We can use this attention matrix G to form representations of the question for each word in passage:

$$\overline{\mathbf{H}}^{\mathrm{q}} = \mathbf{H}^{\mathrm{q}}\mathbf{G} \tag{7.2}$$

Next, we produce the word matching representations $\mathbf{M} \in \mathbb{R}^{2l \times P}$ using $\mathbf{H}^{\mathrm{p}}$ and $\overline{\mathbf{H}}^{\mathrm{q}}$ as follows:

$$\mathbf{M} \quad = \quad \mathrm{ReLU}\left(\mathbf{W}^{\mathrm{m}}\begin{bmatrix}\mathbf{H}^{\mathrm{p}} \\ \overline{\mathbf{H}}^{\mathrm{q}} \\ \mathbf{H}^{\mathrm{p}} \bigodot \overline{\mathbf{H}}^{\mathrm{q}} \\ \mathbf{H}^{\mathrm{p}} - \overline{\mathbf{H}}^{\mathrm{q}}\end{bmatrix}\right), \tag{7.3}$$

where $\mathbf{W}^{\mathrm{m}} \in \mathbb{R}^{2l \times 4l}$ are learnable parameters; $\begin{bmatrix}\cdot \\ \cdot\end{bmatrix}$ is the column concatenation of matrices; Element-wise operations $(\cdot \bigodot \cdot)$ and $(\cdot - \cdot)$ are also used to represent word-level matching [87, 12].

Finally, we aggregate the word matching representations through another bi-directional LSTM:

$$\mathbf{H}^{\mathrm{m}} = \mathrm{BiLSTM}(\mathbf{M}), \tag{7.4}$$

where $\mathbf{H}^{\mathrm{m}} \in \mathbb{R}^{l \times P}$ is the sequence matching representation between a passage and a question.

To produce the input for the Ranker and Reader described next, we apply

Match-LSTMs to the question and each of the passages. To reduce model complexity, the Ranker and Reader share the same $\mathbf{M}$ but have separate parameters for the aggregation stage shown in Eqn.(7.4), resulting different $\mathbf{H}^{\mathrm{m}}$, denoted as $\mathbf{H}^{\mathrm{Rank}}$ and $\mathbf{H}^{\mathrm{Read}}$ respectively.

Ranker Our Ranker selects passages for reading by the Reader. We train the Ranker using reinforcement learning, to output a policy or probability distribution over passages. First, we create a fixed-size vector representation for each passage from the matching representations $\mathbf{H}_{\mathrm{i}}^{\mathrm{Rank}}$, $\mathrm{i} \in [1, N]$, using a standard max pooling operation. The result $\mathbf{u}_{\mathrm{i}}$ is a representation of the i-th passage. We then concatenate the individual passage representations and apply a non-linear transformation followed by a normalization to compute the passage probabilities γ. Specifically:

$$\begin{aligned} \mathbf{u}_i &= \mathrm{MaxPooling}(\mathbf{H}_{\mathrm{i}}^{\mathrm{Rank}}), \\ \mathbf{C} &= \mathrm{Tanh}\left(\mathbf{W}^{\mathrm{c}}[\mathbf{u}_1; \mathbf{u}_2; ...; \mathbf{u}_N] + \mathbf{b}^{\mathrm{c}} \otimes \mathbf{e}_N\right), \\ \gamma &= \mathrm{Softmax}(\mathbf{w}^{c}\mathbf{C}), \end{aligned} \tag{7.5}$$

where $\mathbf{W}^{\mathrm{c}} \in \mathbb{R}^{l \times l}$ and $\mathbf{b}^{\mathrm{c}}, \mathbf{w}^{\mathrm{c}} \in \mathbb{R}^{l}$ are the parameters to optimize; $\mathbf{u}_i \in \mathbb{R}^{l}$ represents how the i-th passage matches the question; $\mathbf{C} \in \mathbb{R}^{l \times N}$ is a non-linear transformation of passage representations; and $\gamma \in \mathbb{R}^{N}$ is a vector of the predicted probabilities that each passage entails the answer.

The action policy is then defined as follows:

$$\pi(\tau|\mathbf{q}; \theta^r) = \gamma_\tau \tag{7.6}$$

where γ_τ is the probability of selecting passage τ, computed in Eqn.(7.5); θ^r represents parameters to learn. In the rest of the chapter we denote the policy $\pi(\tau|q) = \pi(\tau|q; \theta^r)$ for simplicity. In this way, the action is to sample a passage according to its policy $\pi(\tau|q)$ as the input of Reader.

Reader Our Reader extracts an answer span from the passage τ selected by the Ranker. As in previous work [88, 97, 66, 92], the Reader is used to predict the start and end positions of the answer phrase in the passage.

First we process the output of Match-LSTMs on all the passages to produce the probability of the start position of the answer span β^{s}:

$$\begin{aligned} \mathbf{F}^{\mathrm{s}} &= \mathrm{Tanh}\left(\mathbf{W}^{\mathrm{s}}[\mathbf{H}_{\tau}^{\mathrm{Read}}; \mathbf{H}_{\mathrm{neg}_1}^{\mathrm{Read}}; ...; \mathbf{H}_{\mathrm{neg}_n}^{\mathrm{Read}}] + \mathbf{b}^{\mathrm{s}} \otimes \mathbf{e}_V\right), \\ \beta^{\mathrm{s}} &= \mathrm{Softmax}\left(\mathbf{w}^{\mathrm{s}}\mathbf{F}^{\mathrm{s}}\right), \end{aligned} \tag{7.7}$$

where neg_n is the id of a sampled passage not containing ground-truth answer during training; V is the total number of words in these passages; $\mathbf{e}_V$ is thus a V-dimension vector with ones; $[\cdot;\cdot]$ is the column concatenation operation; $\mathbf{W}^{\mathrm{s}} \in \mathbb{R}^{l \times l}$ and $\mathbf{b}^{\mathrm{s}}, \mathbf{w}^{\mathrm{s}} \in \mathbb{R}^{l}$ are the parameters to optimize; $\beta^{\mathrm{s}} \in \mathbb{R}^{V}$ is the probability of the start point of the span.

We similarly compute the probability of the ending position, $\beta^{\mathrm{e}} \in \mathbb{R}^{V}$, using separate parameters $\mathbf{W}^{\mathrm{e}}, \mathbf{b}^{\mathrm{e}}$ and $\mathbf{w}^{\mathrm{e}}$. The loss function can then be expressed as follows:

$$L(\mathbf{a}^{g}|\tau, \mathbf{q}) = -\log(\beta^{\mathrm{s}}_{a^{s}_{\tau}}) - \log(\beta^{\mathrm{e}}_{a^{e}_{\tau}}), \tag{7.8}$$

where $\mathbf{a}^{g}$ is the ground-truth answer; τ is sampled according to Eqn.(7.6), and during training, we keep sampling until passage τ contains $\mathbf{a}^{g}$; $\beta^{\mathrm{s}}_{a^{s}_{\tau}}$ and $\beta^{\mathrm{e}}_{a^{e}_{\tau}}$ represent the probability of the start and end positions of $\mathbf{a}^{g}$ in passage τ.

Training We adopt joint training of Ranker and Reader as shown in Algorithm 1. Since the Ranker makes a hard selection of the passage, it is trained using the REINFORCE algorithm. The Reader is trained using standard SGD/backprop.

Our training objective is to minimize the following loss function

[4]Baseline method SR^2, described in Experimental Settings.

[5]For computational efficency, we sample 10 passages during training, and make sure there are at least 2 negative passages and as many positive passages as possible.

Algorithm 1: Reinforced Ranker-Reader (R^3)

1: **Input:** $\mathbf{a}^g$, $\mathbf{q}$, passages from IR
2: **Output:** Θ
3: **Initialize:** $\Theta \leftarrow$ pre-trained Θ with a baseline method[4]
4: **for** each $\mathbf{q}$ in dataset **do**
5: For question $\mathbf{q}$, sample K passages from the top N passages retrieved by IR model for training. [5]
6: Randomly sample a positive passage $\tau \sim \pi(\tau|\mathbf{q})$
7: Extract the answer $\mathbf{a}^{rc}$ through RC model
8: Get reward r according to $R(\mathbf{a}^g, \mathbf{a}^{rc}|\tau)$.
9: Updating Ranker (ranking model) through policy gradient $r\frac{\partial}{\partial\Theta}\log(\pi(\tau|\mathbf{q}))$
10: Updating Reader (RC model) through supervised gradient $\frac{\partial}{\partial\Theta}L(\mathbf{a}^g|\tau,\mathbf{q})$
11: **end for**

$$J(\Theta) = -\mathbb{E}_{\tau\sim\pi(\tau|\mathbf{q})}\left[L(\mathbf{a}^g|\tau,\mathbf{q})\right], \tag{7.9}$$

where L is the loss of the Reader defined in Eqn. (7.8); $\pi(\tau|\mathbf{q})$ is the action policy defined in Eqn.(7.6); and Θ are parameters to be learned. During training, action sampling is limited solely to passages containing the ground-truth answer, to guarantee Reader updating (line 10 in Algorithm 1) based on the sampled passages with supervised gradients. The gradient of $J(\Theta)$ with respect to Θ is:

$$\begin{aligned}
\nabla_\Theta J(\Theta) &= -\nabla_\Theta \sum_\tau \pi(\tau|\mathbf{q})L(\mathbf{a}^g|\tau,\mathbf{q}) \\
&= -\sum_\tau \left(L(\mathbf{a}^g|\tau,\mathbf{q})\nabla_\Theta\pi(\tau|\mathbf{q}) + \pi(\tau|\mathbf{q})\nabla_\Theta L(\mathbf{a}^g|\tau,\mathbf{q})\right) \\
&= -\mathbb{E}_{\tau\sim\pi(\tau|\mathbf{q})}\left[L(\mathbf{a}^g|\tau,\mathbf{q})\nabla_\Theta\log(\pi(\tau|\mathbf{q}))\right. \\
&\qquad \left. + \nabla_\Theta L(\mathbf{a}^g|\tau,\mathbf{q})\right] \\
&\approx -\mathbb{E}_{\tau\sim\pi(\tau|\mathbf{q})}\left[R(\mathbf{a}^g,\mathbf{a}^{rc}|\tau)\nabla_\Theta\log(\pi(\tau|\mathbf{q}))\right. \\
&\qquad \left. + \nabla_\Theta L(\mathbf{a}^g|\tau,\mathbf{q})\right]
\end{aligned} \tag{7.10}$$

So in training, we first sample a passage τ according to the policy $\pi(\tau|\mathbf{q})$. Then the Reader updates its parameters given the passage τ using standard Backprop and the ranker updates its parameters via policy gradient using

$L(a|\tau, \mathbf{q})$ as rewards. However, $L(a|\tau, \mathbf{q})$ is not bounded and introduces a large variance in gradients (similar to what was reported in [54]). To address this, we replace $L(a|\tau, \mathbf{q})$ with a bounded reward $R(\mathbf{a}^g, \mathbf{a}^{rc}|\tau)$, which captures how well the answer extracted by the Reader matches the ground-truth answer. Specifically:

$$R(\mathbf{a}^g, \mathbf{a}^{rc}|\tau) = \begin{cases} 2, & \text{if } \mathbf{a}^g == \mathbf{a}^{rc} \\ f1(\mathbf{a}^g, \mathbf{a}^{rc}), & \text{else if } \mathbf{a}^g \cap \mathbf{a}^{rc} != \emptyset \\ -1, & \text{else} \end{cases} \tag{7.11}$$

where $\mathbf{a}^g$ is the ground-truth answer; $\mathbf{a}^{rc}$ is the answer extracted by Reader; $f1(\cdot, \cdot) \in [0, 1]$ computes word-level F1 score between two sequences. F1 is used as reward when $\mathbf{a}^g$ and $\mathbf{a}^{rc}$ share some words but do not exactly match. We give a larger reward of 2 for exact match, and -1 reward for no overlap.

Prediction During testing, we combine the Ranker and Reader for answer extraction as follows:

$$\Pr(\mathbf{a}, \tau) = \Pr(\mathbf{a}|\tau) \Pr(\tau) = e^{-L(\mathbf{a}|\tau, \mathbf{q})} \pi(\tau|\mathbf{q}), \tag{7.12}$$

where $\Pr(\mathbf{a}, \tau)$ is the probability of extracting the answer $\mathbf{a}$ from passage τ. We select the answer with the largest $\Pr(\mathbf{a}, \tau)$ as the final prediction.

7.4 Experimental Settings

To evaluate our model we have chosen five challenging datasets under the open-domain QA setting and three public baseline models.

7.4.1 Datasets

We experiment with five different datasets whose statistics are shown in Table 7.2.

Quasar-T is a dataset for SR-QA, with question-answer pairs from various internet sources. Each question is compared to 100 sentence-level candidate passages, retrieved by their IR model from the ClueWeb09 data source, to extract the answer.

The other four datasets we consider are: **SQuAD**, the Stanford QA dataset, from which we take only the question-answer pairs and discard the passages to form an open-domain QA setting (denoted as $\mathbf{SQuAD_{OPEN}}$); **WikiMovies** which contains movie-related questions from the OMDb and MovieLens databases and where the questions can be answered using Wikipedia pages; **CuratedTREC**, based on TREC [81] and designed for open-domain QA; and **WebQuestion** which is designed for knowledge-base QA with answers restricted to Freebase entities. For these four datasets under the open-domain QA setting, no candidate passages are provided so we build a similar sentence-level Search Index based on English Wikipedia, following [10]'s work. To provide a small yet sufficient search space for our model, we employ a traditional IR method to retrieve relevant passages from the whole of Wikipedia. We use the 2016-12-21 dump of English Wikipedia as our sole knowledge source, and build an inverted index with Lucene[6]. We then take each input question as a query to search for top-200 articles, rank them with BM25, and split them into sentences. The sentences are then ranked by TF-IDF and the top-200 sentences for each question retained.

7.4.2 Baselines

We consider three public baseline models[7]: GA [22, 23], a gated-attention reader for text comprehension; BiDAF [66], a reader with bidirectional attention flow for machine comprehension; and DrQA [10], a document reader for question answering. We also compare our model R^3 with two internal baselines:

[6] `https://lucene.apache.org/`

[7] We only compare to the results from the public papers.

	#q(train)	#q(test)	#p(train)	#p(test)
Quasar-T	28496	3000	14.8 / 100	1.9 / 50
$\text{SQuAD}_{\text{OPEN}}$	82271	10570	35.1 / 200	2.3 / 50
WikiMovies	93935	9,952	68.5 / 200	1.8 / 50
CuratedTREC	1204	694	14.6 / 200	4.8 / 50
WebQuestion	3272	2,032	57.2 / 200	4.1 / 50

Table 7.2: Statistics of the datasets. #q represents the number of questions. For the training dataset, we ignore the questions without any answer in all the retrieved passages. In the special case that there's only one answer for the question, during training, we combine the question with the answer as the query to improve IR recall. Otherwise we use only the question. #p represents the number of passages and 14.8 / 100 means there are 14.8 passages containing the answer on average out of the 100 passages. We use top50 passages retrieved by the IR model for testing.

Single Reader (SR) This model is trained in the same way as [10] and [23]. We find all the answer spans that exactly match the ground-truth answers from the retrieved passages and train the Reader using the objective of Eqn.(7.8). Here τ is randomly sampled from $[1, N]$ instead of using Eqn.(7.6).

Simple Ranker-Reader ($\mathbf{SR}^2$) This Ranker-Reader model is trained by combining the two different objective functions for the Single Reader and the Ranker models together. In order to train the Ranker, we treat all the passages that contain the ground-truth answer as positive cases and use the following for the Ranker loss:

$$\sum_{n=1}^{N} y_n \left(\log(y_n) - \log(\gamma_n)\right), \tag{7.13}$$

which is the KL divergence between γ computed through Eqn.(7.5) and a probability vector $\mathbf{y}$, where $y_i = 1/N_p$ when the passage i contains the ground-truth answer, and $y_i = 0/N_p$ otherwise. N_p is the total number of passages which contain the ground-truth answer in the top-N passage list.

	Quasar-T		$\text{SQuAD}_{\text{OPEN}}$		WikiMovies		CuratedTREC		WebQtn	
	EM	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**
GA [22]	26.4	26.4	-	-	-	-	-	-	-	-
BiDAF [66]	25.9	28.5	-	-	-	-	-	-	-	-
DrQA [10]	-	-	28.4	-	34.3	-	25.7	-	**19.5**	-
SR	$31.5^{.2}$	$38.5^{.2}$	$26.9^{.2}$	$35.4^{.2}$	$37.7^{.1}$	$38.8^{.1}$	$27.4^{.4}$	$33.6^{.6}$	$15.2^{.3}$	$22.0^{.2}$
SR^2	$31.9^{.2}$	$38.8^{.2}$	$27.2^{.2}$	$35.8^{.2}$	$38.1^{.1}$	$33.4^{.6}$	$27.7^{.5}$	$39.3^{.1}$	$15.6^{.4}$	$22.5^{.3}$
R^3	$\mathbf{34.2}^{.3}$	$\mathbf{40.9}^{.3}$	$\mathbf{29.1}^{.2}$	$\mathbf{37.5}^{.2}$	$\mathbf{38.8}^{.1}$	$\mathbf{39.9}^{.1}$	$\mathbf{28.4}^{.6}$	$\mathbf{34.3}^{.6}$	$17.1^{.3}$	$\mathbf{24.6}^{.3}$
DrQA-MTL	-	-	29.8	-	36.5	-	25.4	-	20.7	-
YodaQA	-	-	-	-	-	-	31.3	-	39.8	-

Table 7.3: Open-domain question answering results. SR: Single Reader; SR^2: Simple Ranker-Reader; R^3: Reinforced Ranker-Reader; WebQtn: WebQuestions. The results show the average of 5 runs, with standard error in the superscript. The CuratedTREC and WebQuestions models are initialized by training on $\text{SQuAD}_{\text{OPEN}}$ first. On the bottom, YodaQA [3] and DrQA-MTL [10] use additional resources (usage of KB for the former, and multiple training datasets for the latter), so are not a true apple-to-apple comparison to the other methods. EM: Exact Match.

7.4.3 Implementation Details

In order to increase the likelihood that question-related context will be contained in the retrieved passages for the training dataset, if the answer is unique, we combine the question with the answer to form the query for information retrieval. For the testing dataset, we use only the question as a query and collect the top 50 passages for answer extraction.

During training, our R^3 model is first initialized by pre-training the model using the Simple Ranker-Reader (R^2), to encourage convergence. As discussed earlier, the pre-processing and matching layers, Eqn.(7.1-7.3), are shared by both Ranker and Reader. The number of LSTM layers in Eqn.(7.4) is set to 3 for the Reader and 1 for the Ranker.

Our model is optimized using Adamax [43]. We use fixed GloVe [59] word embeddings. We set l to 300, batch size to 30, learning rate to 0.002 and tune the dropout probability.

7.5 Results and Analysis

In this section, we will show the performance of different models on five QA datasets and offer further analysis.

7.5.1 Overall Results

Our results are shown in Table 7.3. We use F1 score and Exact Match (EM) evaluation metrics[8]. We first observe that on Quasar-T, the Single Reader can exceed state-of-the-art performance. Moreover, unlike DrQA, our models are all trained using distant supervision and, without pre-training on the original SQuAD dataset[9], our Single Reader model still achieves better performance on the WikiMovie and CuratedTREC datasets.

Next we observe that the Reinforced Ranker-Reader (R^3) achieves the best performance on the Quasar-T, WikiMovies, and CuratedTREC datests and achieves significantly better performance than our internal baseline model Simple Ranker-Reader (SR^2) on all datasets except CuratedTREC. These results demonstrate the effectiveness of using RL to jointly train the Ranker and Reader both as compared to competing approaches and the non-RL Ranker-Reader baseline.

7.5.2 Further Analysis

In this subsection, we first present an analysis of the improvement of both Ranker and Reader trained with our method, and then discuss ideas for further improvement.

Quantitative Analysis First, we examine whether our RL approach could

[8]Evaluation tooling is from SQuAD [61].

[9]The performance of our Single Reader model on the original SQuAD dev set is F1 77.0, EM 67.6 which is close to the BiDAF model, F1 77.3, EM 67.7 and DrQA model, F1 78.8, EM 69.5.

	F1	EM
Single Reader (SR)	38.3	31.4
SR + Ranker (from SR^2)	38.9	31.8
SR + Ranker (from R^3)	**40.0**	**33.1**
SR^2	38.7	31.9
R^3	40.8	34.1

Table 7.4: Effects of rankers from SR^2 and R^3 (on Quasar-T test dataset). Here we use the same single reader model (SR) as the reader, combined with two different rankers. The performance of the two runs of SR^2 and R^3 (that provide the rankers) is listed at bottom.

	TOP-k	F1	EM
Single Reader (SR)	1	38.3	31.4
Single Reader (SR)	3	51.7	43.7
Single Reader (SR)	5	58.7	49.2
SR + Ranker (from R^3)	1	**40.0**	**33.1**

Table 7.5: Potential improvement on QA performance by improving the ranker. The performance is based on the Quasar-T test dataset. The **TOP-3/5** performance is used to evaluate the further potential improvement by improving rankers (see the "Potential Improvement" section).

help the Ranker overcome the absence of any ground-truth ranking score. To control everything but the change in Ranker, we conduct two experiments combining the same Single Reader with two different Rankers trained from SR^2 and R^3, respectively. Table 7.4 shows the results on the Quasar-T test dataset. Note that the Single Reader combined with the Ranker trained from R^3 model achieves an EM 1.3 higher performance than combined with the Ranker from SR^2 which treats all passages containing ground-truth answer as positive cases. That means our proposed Ranker is better than the Ranker normally trained in the distant supervision setting.

We also find that the performance of R^3 can still achieve an EM 1.0 higher than the Single Reader combined with the Ranker from R^3 through Table 7.4. In this setting, the Ranker is the same, while the Reader is trained differently. We infer from this that our proposed methods R^3 can not only improve the Ranker but also the Reader.

Q	Apart from man what is New Zealand 's only native mammals
A	**bats**

	Reinforced Ranker-Reader (R^3)	Simple Ranker-Reader (SR^2)
P1	New Zealand has no native land mammals apart from some rare **bats** .	New Zealand 's native species were **sitting ducks** !
P2	New Zealand 's native species were sitting **ducks** !	1080 is a commonly used pesticide since it is very effective on mammals and New Zealand has no native land mammals apart from two species of **bat** .
P3	-LSB- edit -RSB- Fauna **Bats** were the only mammals of New Zealand until the arrival of humans .	Previously it had been thought that **bats** were the only terrestrial mammals native to New Zealand .

Table 7.6: An example of the answers extracted by the R^3 and SR^2 methods, given the question. The words in bold are the extracted answers. The passages are ranked by the highest score (Ranker+Reader) of the answer span in each passage.

	TOP-1	TOP-3	TOP-5
IR	19.7	36.3	44.3
Ranker from SR^2	28.8	46.4	54.9
Ranker from R^3	40.3	51.3	54.5

Table 7.7: The performance of Rankers (recall of the top-k ranked passages) on the Quasar-T test dataset. This evaluation is simply based on whether the ground-truth appears in the TOP-N passages. IR directly uses the ranking score from raw dataset.

Potential Improvement We offer a statistical analysis to approximate the upper bound achievable by only improving the ranking models. This is evaluated by computing the QA performance with the best passage among the top-k ranked passages. Specifically, for each question, we extract one answer from each of the top-50 passages retrieved from the IR system, and take the top-k answers with the highest scores according to Eqn.(7.12) from these. Based on the k answer candidates, we compute the **TOP-k** F1/EM by evaluating on the answer with highest F1/EM score for each question. This is equivalent to having an *oracle ranker* that assigns a $+\infty$ score to the passage (from the

passages providing top-k candidates) yielding the best answer candidate.

Table 7.5 shows a clear gap between TOP-3/5 and TOP-1 QA performances (over 12-20%). According to our evaluation approach of TOP-k F1/EM and since the same SR model is used, this gap is solely due to the oracle ranker. Although our model is far from the oracle performance, it still provides a useful upper bound for improvement.

Ranker Performance Analysis Next we show the intermediate performance of our method on the ranking step. Since we do not have the ground-truth for the ranking task, we evaluate on pseudo labels: a passage is considered positive if it contains the ground-truth answer. Then a ranker's top-k output is considered accurate if any of the k passages contain the answer (i.e. *top-k recall*). Note that this way of evaluation on top-1 is consistent with the training objective of the ranker in SR^2.

From the results in Table 7.7, the Ranker from R^3 performs significantly better than the one from SR^2 on top-1 and top-3 performance, despite the fact that it is not directly trained to optimize this pseudo accuracy. Given the evaluation bias that favors the SR^2, this indicates that our R^3 model could make Ranker training easier, compared to training on the objective in Eqn.7.13 with pseudo labels.

Starting from top-5, the Ranker from R^3 gives slightly lower recall. This is because the two Rankers have a similar ability to rank the potentially useful passages in the top-5, but the evaluation bias benefits the SR^2 Ranker. Overall, our R^3 could successfully rank the potentially more useful passages to the highest positions (top 1-3), improving the overall QA performance.

An example in Table 7.6 illustrates the importance of ranking. The passages on the left are from the R^3 Ranker and the ones on the right from the SR^2 Ranker. If SR^2 ranked P2 or P3 higher, it could also have extracted the right answer. In general, if passages that can entail the answer are ranked more

accurately, both models could be improved.

7.6 Conclusion

We have proposed and evaluated R^3, a new open-domain QA framework which combines IR with a deep learning based Ranker and Reader. First the IR model retrieves the top-N passages conditioned on the question. Then the Ranker and Reader are trained jointly using reinforcement learning to directly optimize the expectation of extracting the ground-truth answer from the retrieved passages. Our framework achieves the best performance on several QA datasets.

Chapter 8

Evidence Aggregation for Answer Re-Ranking in Open-Domain Question Answering

For the model in the previous chapter, it can only make use a single passage for answer extraction. In this chapter, we will combine the evidence across different passages to answer the question. Our model will further make use of the combined evidence to re-rank the candidates generated by Reinforced Ranker-Reader. And the textual sequence matching model is the key component of our re-ranker.

8.1 Introduction

Open-domain question answering (QA) aims to answer questions from a broad range of domains by effectively marshalling evidence from large open-domain knowledge sources. Such resources can be Wikipedia [10], the whole web [27], structured knowledge bases [5, 103] or combinations of the above [3]. Recent

work on open-domain QA has focused on using unstructured text retrieved from the web to build machine comprehension models [10, 23, 90]. These studies adopt a two-step process: an information retrieval (IR) model to coarsely select passages relevant to a question, followed by a reading comprehension (RC) model [88, 66, 10] to infer an answer from the passages. These studies have made progress in bringing together evidence from large data sources, but they predict an answer to the question with only a single retrieved passage at a time. However, answer accuracy can often be improved by using multiple passages. In some cases, the answer can only be determined by combining multiple passages.

In this chapter, we propose a method to improve open-domain QA by explicitly aggregating evidence from across multiple passages. Our method is inspired by two notable observations from previous open-domain QA results analysis:

- First, compared with incorrect answers, the correct answer is often suggested by more passages repeatedly. For example, in Figure 8.1(a), the correct answer "*danny boy*" has more passages providing evidence relevant to the question compared to the incorrect one. This observation can be seen as multiple passages collaboratively enhancing the evidence for the correct answer.

- Second, sometimes the question covers multiple answer aspects, which spreads over multiple passages. In order to infer the correct answer, one has to find ways to aggregate those multiple passages in an effective yet sensible way to try to cover all aspects. In Figure 8.1(b), for example, the correct answer "*Galileo Galilei*" at the bottom has passages P1, "*Galileo was a physicist ...*" and P2, "*Galileo discovered the first 4 moons of Jupiter*", mentioning two pieces of evidence to match the question. In this case, the aggregation of these two pieces of evidence can help entail the ground-

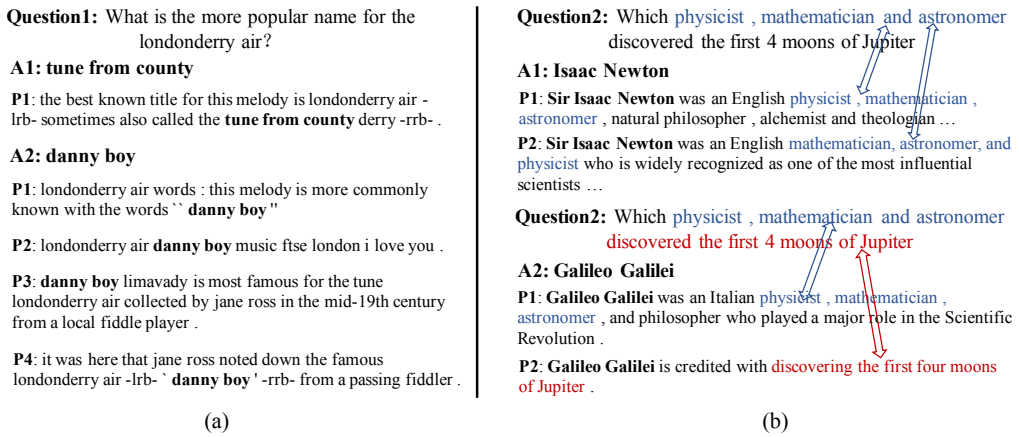

Figure 8.1: Two examples of questions and candidate answers. (a) A question benefiting from the repetition of evidence. Correct answer A2 has multiple passages that could support A2 as answer. The wrong answer A1 has only a single supporting passage. (b) A question benefiting from the union of multiple pieces of evidence to support the answer. The correct answer A2 has evidence passages that can match both the first half and the second half of the question. The wrong answer A1 has evidence passages covering only the first half.

truth answer "*Galileo Galilei*". In comparison, the incorrect answer "*Isaac Newton*" has passages providing partial evidence on only "*physicist, mathematician and astronomer*". This observation illustrates the way in which multiple passages may provide ***complementary*** evidence to better infer the correct answer to a question.

To provide more accurate answers for open-domain QA, we hope to make better use of multiple passages for the same question by aggregating both the strengthened and the complementary evidence from all the passages. We formulate the above evidence aggregation as an answer re-ranking problem. Re-ranking has been commonly used in NLP problems, such as in parsing and translation, in order to make use of high-order or global features that are too expensive for decoding algorithms [18, 67, 39, 25]. Here we apply the idea of re-ranking; for each answer candidate, we efficiently incorporate global information from multiple pieces of textual evidence without significantly increasing the complexity of the prediction of the RC model. Specifically, we first collect the top-K candidate answers based on their probabilities computed by a stan-

dard RC/QA system, and then we use two proposed re-rankers to re-score the answer candidates by aggregating each candidate's evidence in different ways. The re-rankers are:

- A ***strength-based re-ranker***, which ranks the answer candidates according to how often their evidence occurs in different passages. The re-ranker is based on the first observation if an answer candidate has multiple pieces of evidence, and each passage containing some evidence tends to predict the answer with a relatively high score (although it may not be the top score), then the candidate is more likely to be correct. The passage count of each candidate, and the aggregated probabilities for the candidate, reflect how strong its evidence is, and thus in turn suggest how likely the candidate is the corrected answer.

- A ***coverage-based re-ranker***, which aims to rank an answer candidate higher if the union of all its contexts in different passages could cover more aspects included in the question. To achieve this, for each answer we concatenate all the passages that contain the answer together. The result is a new context that aggregates all the evidence necessary to entail the answer for the question. We then treat the new context as one sequence to represent the answer, and build an attention-based match-LSTM model [88] between the sequence and the question to measure how well the new aggregated context could entail the question.

Overall, our contributions are as follows: 1) We propose a re-ranking-based framework to make use of the evidence from multiple passages in open-domain QA, and two re-rankers, namely, a strength-based re-ranker and a coverage-based re-ranker, to perform evidence aggregation in existing open-domain QA datasets. We find the second re-ranker performs better than the first one on two of the three public datasets. 2) Our proposed approach leads to the state-of-the-art results on three different datasets (**Quasar-T** [23], **SearchQA** [24] and

TriviaQA [40]) and outperforms previous state of the art by large margins. In particular, we achieved up to 8% improvement on F1 on both Quasar-T and SearchQA compared to the previous best results.

8.2 Method

Given a question $\mathbf{q}$, we are trying to find the correct answer $\mathbf{a}^g$ to $\mathbf{q}$ using information retrieved from the web. Our method proceeds in two phases. First, we run an IR model (with the help of a search engine such as google or bing) to find the top-N web passages $\mathbf{p}_1$, $\mathbf{p}_2$, ..., $\mathbf{p}_N$ most related to the question. Then a reading comprehension (RC) model is used to extract the answer from these passages. This setting is different from standard reading comprehension tasks (e.g. [61]), where a single fixed passage is given, from which the answer is to be extracted. When developing a reading comprehension system, we can use the specific positions of the answer sequence in the given passage for training. By contrast, in the open-domain setting, the RC models are usually trained under distant supervision [10, 23, 40]. Specifically, since the training data does not have labels indicating the positions of the answer spans in the passages, during the training stage, the RC model will match all passages that contain the ground-truth answer with the question one by one. In this chapter we apply an existing RC model called R^3 [90] to extract these candidate answers.

After the candidate answers are extracted, we aggregate evidence from multiple passages by re-ranking the answer candidates. Given a question $\mathbf{q}$, suppose we have a baseline open-domain QA system that can generate the top-K answer candidates $\mathbf{a}_1, \ldots, \mathbf{a}_K$, each being a text span in some passage $\mathbf{p}_i$. The goal of the re-ranker is to rank this list of candidates so that the top-ranked candidates are more likely to be the correct answer $\mathbf{a}^g$. With access to these additional features, the re-ranking step has the potential to prioritize answers not easily discoverable by the base system alone. We investigate two

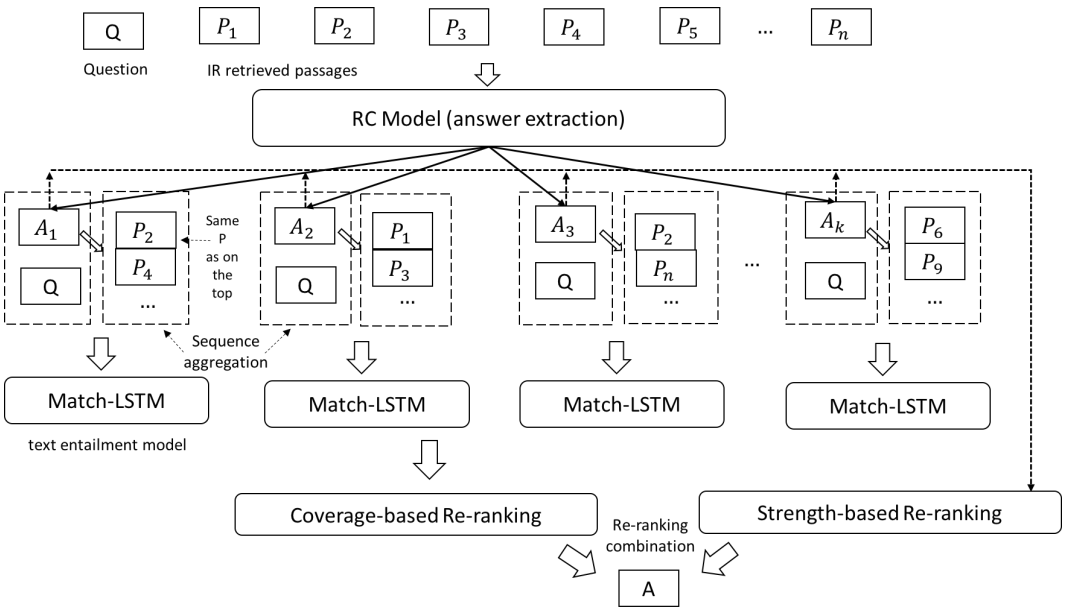

Figure 8.2: An overview of the full re-ranker. It consists of strength-based and coverage-based re-ranking.

re-ranking strategies based on *evidence strength* and *evidence coverage*. An overview of our method is shown in Figure 8.2.

8.2.1 Evidence Aggregation for Strength-based Re-ranker

In open-domain QA, unlike the standard RC setting, we have more passages retrieved by the IR model and the ground-truth answer may appear in different passages, which means different answer spans may correspond to the same answer. To exploit this property, we provide two features to further re-rank the top-K answers generated by the RC model.

Measuring Strength by Count This method is based on the hypothesis that the more passages that entail a particular answer, the stronger the evidence for that answer and the higher it should be ranked. To implement this we count the number of occurrences of each answer in the top-K answer spans generated by the baseline QA model and return the answer with the highest

count.

Measuring Strength by Probability Since we can get the probability of each answer span in a passage based on the RC model, we can also sum up the probabilities of the answer spans that are referring to the same answer. In this method, the answer with the highest probability is the final prediction [1]. In the re-ranking scenario, it is not necessary to exhaustively consider all the probabilities of all the spans in the passages, as there may be a large number of different answer spans and most of them are irrelevant to the ground-truth answer.

Remark: Note that neither of the above methods require any training. Both just take the candidate predictions from the baseline QA system and perform counting or probability calculations. At test time, the time complexity of strength-based re-ranking is negligible.

8.2.2 Evidence Aggregation for Coverage-based Re-ranker

Consider Figure 8.1 where the two answer candidates both have evidence matching the first half of the question. Note that only the correct answer has evidence that could also match the second half. In this case, the strength-based re-ranker will treat both answer candidates the same due to the equal amount of supporting evidence, while the second answer has complementary evidence satisfying all aspects of the question. To handle this case, we propose a *coverage-based re-ranker* that ranks the answer candidates according to how well the union of their evidence from different passages covers the question.

In order to take the union of evidence into consideration, we first concatenate the passages containing the answer into a single "pseudo passage" then

[1] This is an extension of the Attention Sum method in [41] from single-token answers to phrase answers.

measure how well this passage entails the answer for the question. As in the examples shown in Figure 8.1(b), we hope the textual entailment model will reflect (i) how each aspect of the question is matched by the union of multiple passages; and (ii) whether all the aspects of the question can be matched by the union of multiple passages. In our implementation an "aspect" of the question is a hidden state of a bi-directional LSTM [37]. The match-LSTM [86] model is one way to achieve the above effect in entailment. Therefore we build our coverage-based re-ranker on top of the concatenated pseudo passages using the match-LSTM. The detailed method is described below.

Passage Aggregation We consider the top-K answers, $\mathbf{a}_1, \ldots, \mathbf{a}_K$, provided by the baseline QA system. For each answer $\mathbf{a}_k, k \in [1, K]$, we concatenate all the passages that contain $\mathbf{a}_k$, $\{\mathbf{p}_n | \mathbf{a}_k \in \mathbf{p}_n, n \in [1, N]\}$, to form ***the union passage*** $\hat{\mathbf{p}}_k$. Our further model is to identify which union passage, e.g., $\hat{\mathbf{p}}_k$, could better entail its answer, e.g., $\mathbf{a}_k$, for the question.

Measuring Aspect(Word)-Level Matching As discussed earlier, the first mission of the coverage-based re-ranker is to measure how each aspect of the question is matched by the union of multiple passages. We achieve this with word-by-word attention followed by a comparison module.

First, we write the answer candidate $\mathbf{a}$, question $\mathbf{q}$ and the union passage $\hat{\mathbf{p}}$ of $\mathbf{a}$ as matrices $\mathbf{A}, \mathbf{Q}, \hat{\mathbf{P}}$, with each column being the embedding of a word in the sequence. We then feed them to the bi-directional LSTM as follows:

$$\mathbf{H}^{\mathrm{a}} = \mathrm{BiLSTM}(\mathbf{A}), \quad \mathbf{H}^{\mathrm{q}} = \mathrm{BiLSTM}(\mathbf{Q}), \quad \mathbf{H}^{\mathrm{p}} = \mathrm{BiLSTM}(\hat{\mathbf{P}}), \tag{8.1}$$

where $\mathbf{H}^{\mathrm{a}} \in \mathbb{R}^{l \times A}$, $\mathbf{H}^{\mathrm{q}} \in \mathbb{R}^{l \times Q}$ and $\mathbf{H}^{\mathrm{p}} \in \mathbb{R}^{l \times P}$ are the hidden states for the answer candidate, question and passage respectively; l is the dimension of the hidden states, and A, Q and P are the length of the three sequences, respectively.

Next, we enhance the question representation $\mathbf{H}^{\mathrm{q}}$ with $\mathbf{H}^{\mathrm{a}}$:

$$\mathbf{H}^{\mathrm{aq}} = [\mathbf{H}^{\mathrm{a}}; \mathbf{H}^{\mathrm{q}}], \tag{8.2}$$

where $[\cdot;\cdot]$ is the concatenation of two matrices in row and $\mathbf{H}^{\mathrm{aq}} \in \mathbb{R}^{l\times(A+Q)}$. As most of the answer candidates do not appear in the question, this is for better matching with the passage and finding more answer-related information from the passage.[2] Now we can view each ***aspect*** of the question as a column vector (i.e. a hidden state at each word position in the answer-question concatenation) in the enhanced question representation $\mathbf{H}^{\mathrm{aq}}$. Then the task becomes to measure how well each column vector can be matched by the union passage; and we achieve this by computing the attention vector [58] for each hidden state of sequences $\mathbf{a}$ and $\mathbf{q}$ as follows:

$$\alpha = \mathrm{SoftMax}\left((\mathbf{H}^{\mathrm{p}})^{\mathrm{T}}\mathbf{H}^{\mathrm{aq}}\right), \quad \overline{\mathbf{H}}^{\mathrm{aq}} = \mathbf{H}^{\mathrm{p}}\alpha, \tag{8.3}$$

where $\alpha \in \mathbb{R}^{P\times(A+Q)}$ is the attention weight matrix which is normalized in column through softmax. $\overline{\mathbf{H}}^{\mathrm{aq}} \in \mathbb{R}^{l\times(A+Q)}$ are the attention vectors for each word of the answer and the question by weighted summing all the hidden states of the passage $\hat{\mathbf{p}}$. Now in order to see whether the aspects in the question can be matched by the union passage, we use the following matching function:

$$\mathbf{M} = \mathrm{ReLU}\left(\mathbf{W}^{\mathrm{m}}\begin{bmatrix}\mathbf{H}^{\mathrm{aq}} \bigodot \overline{\mathbf{H}}^{\mathrm{aq}} \\ \mathbf{H}^{\mathrm{aq}} - \overline{\mathbf{H}}^{\mathrm{aq}} \\ \mathbf{H}^{\mathrm{aq}} \\ \overline{\mathbf{H}}^{\mathrm{aq}}\end{bmatrix} + \mathbf{b}^{\mathrm{m}} \otimes \mathbf{e}_{(A+Q)}\right), \tag{8.4}$$

[2]Besides concatenating $\mathbf{H}^{\mathrm{q}}$ with $\mathbf{H}^{\mathrm{a}}$, there are other ways to make the matching process be aware of an answer's positions in the passage, e.g. replacing the answer spans in the passage to a special token like in [100]. We tried this approach, which gives similar but no better results, so we keep the concatenation in this chapter. We leave the study of the better usage of answer position information for future work.

where $\cdot \otimes \mathbf{e}_{(A+Q)}$ is to repeat the vector (or scalar) on the left $A+Q$ times; $(\cdot \bigodot \cdot)$ and $(\cdot - \cdot)$ are the element-wise operations for checking whether the word in the answer and question can be matched by the evidence in the passage. We also concatenate these matching representations with the hidden state representations $\mathbf{H}^{\text{aq}}$ and $\overline{\mathbf{H}}^{\text{aq}}$, so that the lexical matching representations are also integrated into the the final aspect-level matching representations[3] $\mathbf{M} \in \mathbb{R}^{2l \times (A+Q)}$, which is computed through the non-linear transformation on four different representations with parameters $\mathbf{W}^{\text{m}} \in \mathbb{R}^{2l \times 4l}$ and $b^{\text{m}} \in \mathbb{R}^{l}$.

Measuring the Entire Question Matching Next, in order to measure how the entire question is matched by the union passage $\hat{\mathbf{p}}$ by taking into consideration of the matching result at each aspect, we add another bi-directional LSTM on top of it to aggregate the aspect-level matching information [4]:

$$\mathbf{H}^{\text{m}} = \text{BiLSTM}(\mathbf{M}), \quad \mathbf{h}^{\text{s}} = \text{MaxPooling}(\mathbf{H}^{\text{m}}), \tag{8.5}$$

where $\mathbf{H}^{\text{m}} \in \mathbb{R}^{l \times (A+Q)}$ is to denote all the hidden states and $\mathbf{h}^{\text{s}} \in \mathbb{R}^{l}$, the max of pooling of each dimension of $\mathbf{H}^{\text{m}}$, is the entire matching representation which reflects how well the evidences in questions could be matched by the union passage.

Re-ranking Objective Function Our re-ranking is based on the entire matching representation. For each candidate answer $\mathbf{a}_k, k \in [1, K]$, we can get a matching representation $\mathbf{h}^{\text{s}}_k$ between the answer $\mathbf{a}_k$, question $\mathbf{q}$ and the union passage $\hat{\mathbf{p}}_k$ through Eqn. (8.1-8.5). Then we transform all representations into

[3]Concatenating $\mathbf{H}^{\text{aq}}$ and $\overline{\mathbf{H}}^{\text{aq}}$ could help the question-level matching (see Eq. 8.4 in the next paragraph) by allowing the BiLSTM learn to distinguish the effects of the element-wise comparison vectors with the original lexical information. If we only use the element-wise comparison vectors, the model may not be able to know what the matched words/contexts are.

[4]Note that we use LSTM here to capture the conjunction information (the dependency) among aspects, i.e. how all the aspects are jointly matched. In comparison simple pooling methods will treat the aspects independently. Low-rank tensor inspired neural architectures (e.g., [48]) could be another choice and we will investigate them in future work.

scalar values followed by a normalization process for ranking:

$$\mathbf{R} = \text{Tanh}\left(\mathbf{W}^{\text{r}}[\mathbf{h}_1^{\text{s}}; \mathbf{h}_2^{\text{s}}; ...; \mathbf{h}_K^{\text{s}}] + \mathbf{b}^{\text{r}} \otimes \mathbf{e}_K\right), \quad \mathbf{o} = \text{Softmax}(\mathbf{w}^o\mathbf{R} + b^o \otimes \mathbf{e}_K), \tag{8.6}$$

where we concatenate the match representations for each answer in row through $[\cdot;\cdot]$, and do a non-linear transformation by parameters $\mathbf{W}^{\text{r}} \in \mathbb{R}^{l\times l}$ and $\mathbf{b}^{\text{r}} \in \mathbb{R}^l$ to get hidden representation $\mathbf{R} \in \mathbb{R}^{l\times K}$. Finally, we map the transformed matching representations into scalar values through parameters $\mathbf{w}^o \in \mathbb{R}^l$ and $\mathbf{w}^o \in \mathbb{R}$. $\mathbf{o} \in \mathbb{R}^K$ is the normalized probability for the candidate answers to be ground-truth. Due to the aliases of the ground-truth answer, there may be multiple answers in the candidates are ground-truth, we use KL distance as our objective function:

$$\sum_{k=1}^{K} y_k \left(\log(y_k) - \log(o_k)\right), \tag{8.7}$$

where y_k indicates whether $\mathbf{a}_k$ the ground-truth answer or not and is normalized by $\sum_{k=1}^{K} y_k$ and o_k is the ranking output of our model for $\mathbf{a}_k$.

8.2.3 Combination of Different Types of Aggregations

Although the coverage-based re-ranker tries to deal with more difficult cases compared to the strength-based re-ranker, the strength-based re-ranker works on more common cases according to the distributions of most open-domain QA datasets. We can try to get the best of both worlds by combining the two approaches. The ***full re-ranker*** is a weighted combination of the outputs of the above different re-rankers without further training. Specifically, we first use softmax to re-normalize the top-5 answer scores provided by the two strength-based rankers and the one coverage-based re-ranker; we then weighted sum up the scores for the same answer and select the answer with the largest score as

the final prediction.

8.3 Experimental Settings

We conduct experiments on three publicly available open-domain QA datasets, namely, **Quasar-T** [23], **SearchQA** [24] and **TriviaQA** [40]. These datasets contain passages retrieved for all questions using a search engine such as Google or Bing. We do not retrieve more passages but use the provided passages only.

8.3.1 Datasets

The statistics of the three datasets are shown in Table 8.1.

Quasar-T [5] [23] is based on a trivia question set. The data set makes use of the "Lucene index" on the ClueWeb09 corpus. For each question, 100 unique sentence-level passages were collected. The human performance is evaluated in an open-book setting, i.e., the human subjects had access to the same passages retrieved by the IR model and tried to find the answers from the passages.

SearchQA [6] [24] is based on Jeopardy! questions and uses Google to collect about 50 web page snippets as passages for each question. The human performance is evaluated in a similar way to the Quasar-T dataset.

TriviaQA (Open-Domain Setting) [7] [40] collected trivia questions coming from 14 trivia and quiz-league websites, and makes use of the Bing Web search API to collect the top 50 webpages most related to the questions. We focus on the open domain setting (the unfiltered passage set) of the dataset [8] and our model uses all the information retrieved by the IR model.

[5] `https://github.com/bdhingra/quasar`

[6] `https://github.com/nyu-dl/SearchQA`

[7] `http://nlp.cs.washington.edu/triviaqa/data/triviaqa-unfiltered.tar.gz`

[8] Despite the open-domain QA data provided, the leaderboard of TriviaQA focuses on evaluation of RC models over filtered passages that is guaranteed to contain the correct answers (i.e. more like closed-domain setting). The evaluation is also passage-wise, different from the open-domain QA setting.

	#q(train)	#q(dev)	#q(test)	#p	#p(truth)	#p(aggregated)
Quasar-T	28,496	3,000	3,000	100	14.8	5.2
SearchQA	99,811	13,893	27,247	50	16.5	5.4
TriviaQA	66,828	11,313	10,832	100	16.0	5.6

Table 8.1: Statistics of the datasets. #q represents the number of questions for training (not counting the questions that don't have ground-truth answer in the corresponding passages for training set), development, and testing datasets. #p is the number of passages for each question. For TriviaQA, we split the raw documents into sentence level passages and select the top 100 passages based on the its overlaps with the corresponding question. #p(golden) means the number of passages that contain the ground-truth answer in average. #p(aggregated) is the number of passages we aggregated in average for top 10 candidate answers provided by RC model.

8.3.2 Baselines

Our baseline models [9] include the following: GA [22, 23], a reading comprehension model with gated-attention; BiDAF [66], a RC model with bidirectional attention flow; AQA [8], a reinforced system learning to aggregate the answers generated by the re-written questions; R^3 [90], a reinforced model making use of a ranker for selecting passages to train the RC model. As R^3 is the first step of our system for generating candidate answers, the improvement of our re-ranking methods can be directly compared to this baseline.

TriviaQA does not provide the leaderboard under the open-domain setting. As a result, there is no public baselines in this setting and we only compare with the R^3 baseline.[10]

[9] Most of the results of different models come from the public paper. While we re-run model R^3 [90] based on the authors' source code and extend the model to the datasets of SearchQA and TriviaQA datasets.

[10] To demonstrate that R^3 serves as a strong baseline on the TriviaQA data, we generate the R^3 results following the leaderboard setting. The results showed that R^3 achieved F1 56.0, EM 50.9 on Wiki domain and F1 68.5, EM 63.0 on Web domain, which is competitive to the state-of-the-arts. This confirms that R^3 is a competitive baseline when extending the TriviaQA questions to open-domain setting.

8.3.3 Implementation Details

We first use a pre-trained R^3 model [90], which gets the state-of-the-art performance on the three public datasets we consider, to generate the top 50 candidate spans for the training, development and test datasets, and we use them for further ranking. During training, if the ground-truth answer does not appear in the answer candidates, we will manually add it into the answer candidate list.

For the coverage-based re-ranker, we use Adam [43] to optimize the model. Word embeddings are initialized by GloVe [59] and are not updated during training. We set all the words beyond Glove as zero vectors. We set l to 300, batch size to 30, learning rate to 0.002. We tune the dropout probability from 0 to 0.5 and the number of candidate answers for re-ranking (K) in [3, 5, 10] [11].

8.4 Results and Analysis

In this section, we present results and analysis of our different re-ranking methods on the three different public datasets.

8.4.1 Overall Results

The performance of our models is shown in Table 8.2. We use F1 score and Exact Match (EM) as our evaluation metrics [12]. From the results, we can clearly see that the full re-ranker, the combination of different re-rankers, significantly outperforms the previous best performance by a large margin, especially on Quasar-T and SearchQA. Moreover, our model is much better than the human performance on the SearchQA dataset. In addition, we see that our coverage-based re-ranker achieves consistently good performance on the three datasets, even though its performance is marginally lower than the

[11] Our code will be released under `https://github.com/shuohangwang/mprc`.
[12] Our evaluation is based on the tool from SQuAD [61].

	Quasar-T		SearchQA		TriviaQA	
	EM	**F1**	**EM**	**F1**	**EM**	**F1**
GA [22]	26.4	26.4	-	-	-	-
BiDAF [66]	25.9	28.5	28.6	34.6	-	-
AQA [8]	-	-	40.5	47.4	-	-
R^3 [90]	35.3	41.7	49.0	55.3	47.3	53.7
Baseline Re-Ranker (BM25)	33.6	45.2	51.9	60.7	44.6	55.7
Our Full Re-Ranker	**42.3**	**49.6**	**57.0**	**63.2**	**50.6**	**57.3**
Strength-Based Re-Ranker (Probability)	36.1	42.4	50.4	56.5	49.2	55.1
Strength-Based Re-Ranker (Counting)	37.1	46.7	54.2	61.6	46.1	55.8
Coverage-Based Re-Ranker	40.6	49.1	54.1	61.4	50.0	57.0
Human Performance	51.5	60.6	43.9	-	-	-

Table 8.2: Experiment results on three open-domain QA test datasets: Quasar-T, SearchQA and TriviaQA (open-domain setting). EM: Exact Match. Full Re-ranker is the combination of three different re-rankers.

strength-based re-ranker on the SearchQA dataset.

8.4.2 Analysis

In this section, we analyze the benefits of our re-ranking models.

BM25 as an alternative coverage-based re-ranker We use the classical BM25 retrieval model [63] to re-rank the aggregated passages the same way as the coverage-based re-ranker, where the IDF values are first computed from the raw passages before aggregation. From the results in Table 8.2, we see that the BM25-based re-ranker improves the F1 scores compared with the R^3 model, but it is still lower than our coverage-based re-ranker with neural network models. Moreover, with respect to EM scores, the BM25-based re-ranker sometimes gives lower performance. We hypothesize that there are two reasons behind the relatively poor performance of BM25. First, because BM25 relies on a bag-of-words representation, context information is not taken into consideration and it cannot model the phrase similarities. Second, shorter answers tend to be preferred by BM25. For example, in our method of constructing pseudo-passages, when an answer sequence A is a subsequence of another an-

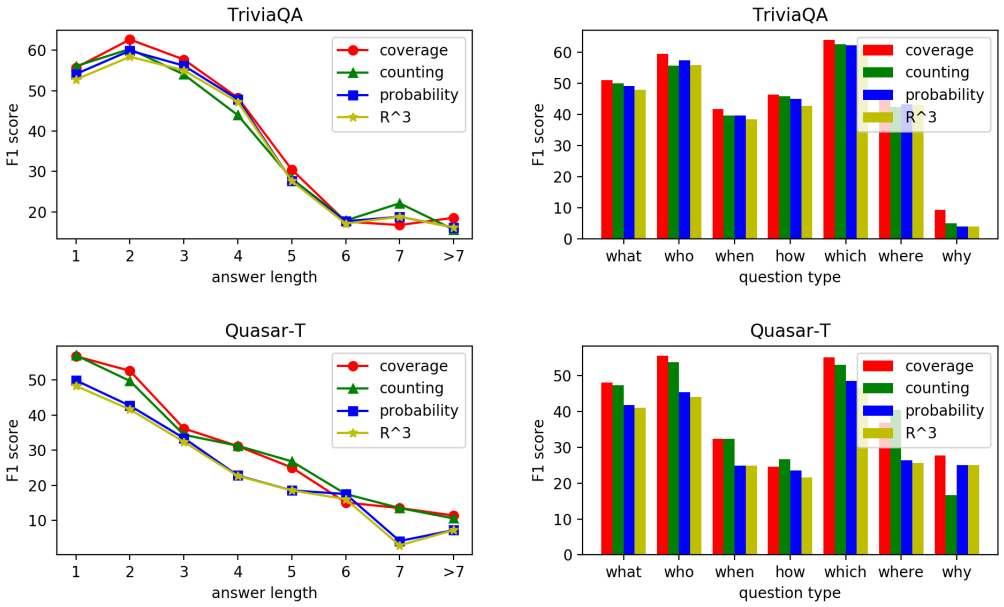

Figure 8.3: Performance decomposition according to the length of answers and the question types.

swer sequence B, the pseudo passage of A is always a superset of the pseudo passage of B that could better cover the question. Therefore the F1 score could be improved but the EM score sometimes becomes worse.

Re-ranking performance versus answer lengths and question types Figure 8.3 decomposes the performance according to the length of the ground truth answers and the types of questions on TriviaQA and Quasar-T. We do not include the analysis on SearchQA because, for the Jeopardy! style questions, it is more difficult to distinguish the questions types, and the range of answer lengths is narrower.

Our results show that the coverage-based re-ranker outperforms the baseline in different lengths of answers and different types of questions. The strength-based re-ranker (counting) also gives improvement but is less stable across different datasets, while the strength-based re-ranker (probability) tends to have results and trends that are close to the baseline curves, which is probably because the method is dominated by the probabilities predicted by

Top-K	Quasar-T EM	Quasar-T F1	SearchQA EM	SearchQA F1	TriviaQA (open) EM	TriviaQA (open) F1
1	35.1	41.6	51.2	57.3	47.6	53.5
3	46.2	53.5	63.9	68.9	54.1	60.4
5	51.0	58.9	69.1	73.9	58.0	64.5
10	56.1	64.8	75.5	79.6	62.1	69.0

Table 8.3: The upper bound (recall) of the Top-K answer candidates generated by the baseline R^3 system (on dev set), which indicates the potential of the coverage-based re-ranker.

the baseline.

The coverage-based re-ranker and the strength-based re-ranker (counting) have similar trends on most of the question types. The only exception is that the strength-based re-ranker performs significantly worse compared to the coverage-based re-ranker on the "*why*" questions. This is possibly because those questions usually have non-factoid answers, which are less likely to have exactly the same text spans predicted on different passages by the baseline.

Potential improvement of re-rankers Table 8.3 shows the percentage of times the correct answer is included in the top-K answer predictions of the baseline R^3 method. More concretely, the scores are computed by selecting the answer from the top-K predictions with the best EM/F1 score. Therefore the final top-K EM and F1 can be viewed as the recall or an upper bound of the top-K predictions. From the results, we can see that although the top-1 prediction of R^3 is not very accurate, there is high probability that a top-K list with small K could cover the correct answer. This explains why our re-ranking approach achieves large improvement. Also by comparing the upper bound performance of top-5 and our re-ranking performance in Table 8.2, we can see there is still a clear gap of about 10% on both datasets and on both F1 and EM, showing the great potential improvement for the re-ranking model in future work.

Candidate Set	Re-Ranker Results EM	F1	Upper Bound EM	F1
top-3	40.5	47.8	46.2	53.5
top-5	**41.8**	50.1	51.0	58.9
top-10	41.3	**50.8**	56.1	64.8

Table 8.4: Results of running coverage-based re-ranker on different number of the top-K answer candidates on Quasar-T (dev set).

Effect of the selection of K for the coverage-based re-ranker As shown in Table 8.3, as K ranges from 1 to 10, the recall of top-K predictions from the baseline R^3 system increases significantly. Ideally, if we use a larger K, then the candidate lists will be more likely to contain good answers. At the same time, the lists to be ranked are longer thus the re-ranking problem is harder. Therefore, there is a trade-off between the coverage of rank lists and the difficulty of re-ranking; and selecting an appropriate K becomes important. Table 8.4 shows the effects of K on the performance of coverage-based re-ranker. We train and test the coverage-based re-ranker on the top-K predictions from the baseline, where $K \in \{3, 5, 10\}$. The upper bound results are the same ones from Table 8.3. The results show that when K is small, like K=3, the performance is not very good due to the low coverage (thus low upper bound) of the candidate list. With the increase of K, the performance becomes better, but the top-5 and top-10 results are on par with each other. This is because the higher upper bound of top-10 results counteracts the harder problem of re-ranking longer lists. Since there is no significant advantage of the usage of K=10 while the computation cost is higher, we report all testing results with K=5.

Effect of the selection of K for the strength-based re-ranker Similar to Table 8.4, we conduct experiments to show the effects of K on the performance of the strength-based re-ranker. We run the strength-based re-ranker (counting) on the top-K predictions from the baseline, where $K \in$

Candidate Set	Re-Ranker Results EM	F1	Upper Bound EM	F1
top-10	**37.9**	46.1	56.1	64.8
top-50	37.8	**47.8**	64.1	74.1
top-100	36.4	47.3	66.5	77.1
top-200	33.7	45.8	68.7	79.5

Table 8.5: Results of running strength-based re-ranker (counting) on different number of top-K answer candidates on Quasar-T (dev set).

$\{10, 50, 100, 200\}$. We also evaluate the upper bound results for these Ks. Note that the strength-based re-ranker is very fast and the different values of K do not affect the computation speed significantly compared to the other QA components.

The results are shown in Table 8.5, where we achieve the best results when K=50. The performance drops significantly when K increases to 200. This is because the ratio of incorrect answers increases notably, making incorrect answers also likely to have high counts. When K is smaller, such incorrect answers appear less because statistically they have lower prediction scores. We report all testing results with K=50.

Examples Table 8.6 shows an example from Quasar-T where the re-ranker successfully corrected the wrong answer predicted by the baseline. This is a case where the coverage-based re-ranker helped: the correct answer "*Sesame Street*" has evidence from different passages that covers the aspects "*Emmy Award*" and "*children 's television shows*". Although it still does not fully match all the facts in the question, it still helps to rank the correct answer higher than the top-1 prediction "*Great Dane*" from the R^3 baseline, which only has evidence covering "*TV*" and "*1969*" in the question.

Q:	Which children 's TV programme , which first appeared in November 1969 , has won a record 122 Emmy Awards in all categories ?		
A1:	**Great Dane**	A2:	**Sesame Street**
P1	The world 's most famous Great Dane first appeared on television screens on Sept. 13 , 1969 .	P1:	In its long history , Sesame Street has received more *Emmy Awards* than any other program , ...
P2	premiered on broadcast television (CBS) Saturday morning , Sept. 13 , 1969 , ... yet beloved great Dane .	P2:	Sesame Street ... is recognized as a pioneer of the contemporary standard which combines education and entertainment in *children 's television shows* .

Table 8.6: An example from Quasar-T dataset. The ground-truth answer is "*Sesame Street*". Q: question, A: answer, P: passages containing corresponding answer.

8.5 Conclusions

We have observed that open-domain QA can be improved by explicitly combining evidence from multiple retrieved passages. We experimented with two types of re-rankers, one for the case where evidence is consistent and another when evidence is complementary. Both re-rankers helped to significantly improve our results individually, and even more together. Our results considerably advance the state-of-the-art on three open-domain QA datasets.

Although our proposed methods achieved some successes in modeling the union or co-occurrence of multiple passages, there are still much harder problems in open-domain QA that require reasoning and commonsense inference abilities. In future work, we will explore the above directions, and we believe that our proposed approach could be potentially generalized to these more difficult multi-passage reasoning scenarios.

Chapter 9

Conclusion

In conclusion, this thesis consists of three major parts: (I). General textual sequence matching models; and their applications to (II). machine reading comprehension and (III). open-domain question answering. By integrating our sequence matching model to question answering systems, we achieved state-of-the-art performance on a number of benchmark QA datasets: SQuAD, MS-MARCO, RACE, SearchQA, Qusar, TriviaQA, InsuranceQA, WikiQA etc.. And we can also see that sequence matching model plays a key role to solve the QA tasks. However, even though most of the question answering can be converted to a sequence matching problem, the model still couldn't understand the language.

In the future work, I would like to focus more on integrating more general knowledge into the model. The simplest way can be just integrating the human crafted knowledge base, such as WordNet etc., but I still believe we can't label all the knowledge by ourselves. We need some unsupervised methods to help us extract it automatically. The success of ELMo [60] and BERT [21] is one way to address this issue. I'll also have some further exploration in this area. Another direction that might be also important for question answering is the reasoning component. For now, most of the models are usually based on the matching model and the attention mechanism, but not be able to do

complicated reasoning. In my future work, I would also like to have some further exploration on the memory/cache based model to enhance the ability of machine reasoning.

Bibliography

[1] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[2] D. Bahdanau, H. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*, 2015.

[3] P. Baudiš and J. Šedivỳ. Modeling of the question answering task in the yodaqa system. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 222–228. Springer, 2015.

[4] M. Bendersky, D. Metzler, and W. B. Croft. Learning concept importance using a weighted dependence model. In *Proceedings of the third ACM International Conference on Web Search and Data Mining*. ACM, 2010.

[5] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013.

[6] A. Bordes, N. Usunier, S. Chopra, and J. Weston. Large-scale simple question answering with memory networks. *Proceedings of the International Conference on Learning Representations*, 2015.

[7] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[8] C. Buck, J. Bulian, M. Ciaramita, A. Gesmundo, N. Houlsby, W. Gajewski, and W. Wang. Ask the right questions: Active question reformulation with reinforcement learning. *arXiv preprint arXiv:1705.07830*, 2017.

[9] D. Chen, J. Bolton, and C. D. Manning. A thorough examination of the CNN/Daily Mail reading comprehension task. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

[10] D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading Wikipedia to answer open-domain questions. In *Proceedings of the Conference on Association for Computational Linguistics*, 2017.

[11] Q. Chen, X. Zhu, Z. Ling, S. Wei, and H. Jiang. Enhancing and combining sequential and tree LSTM for natural language inference. *arXiv preprint arXiv:1609.06038*, 2016.

[12] Q. Chen, X. Zhu, Z. Ling, S. Wei, H. Jiang, and D. Inkpen. Enhanced lstm for natural language inference. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

[13] T. Chen and B. Van Durme. Discriminative information retrieval for question answering sentence selection. In *Proc. of Conf. on EACL*, 2017.

[14] J. Cheng, L. Dong, and M. Lapata. Long short-term memory-networks for machine reading. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

[15] J. Cheng and M. Lapata. Neural summarization by extracting sentences and words. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

[16] E. Choi, D. Hewlett, J. Uszkoreit, I. Polosukhin, A. Lacoste, and J. Berant. Coarse-to-fine question answering for long documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 2017.

[17] P. Clark and P. Harrison. An inference-based approach to recognizing entailment. In *Proceedings of the Text Analysis Conference*, 2009.

[18] M. Collins and T. Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70, 2005.

[19] Y. Cui, T. Liu, Z. Chen, S. Wang, and G. Hu. Consensus attention-based neural networks for chinese reading comprehension. In *arXiv preprint arXiv:1607.02250*, 2016.

[20] I. Dagan, O. Glickman, and B. Magnini. The PASCAL Recognising Textual Entailment Challenge. In *Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment*, 2005.

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

[22] B. Dhingra, H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov. Gated-attention readers for text comprehension. *Proceedings of the Conference on Association for Computational Linguistics*, 2017.

[23] B. Dhingra, K. Mazaitis, and W. W. Cohen. QUASAR: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*, 2017.

[24] M. Dunn, L. Sagun, M. Higgins, U. Guney, V. Cirik, and K. Cho. SearchQA: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

[25] C. Dyer, A. Kuncoro, M. Ballesteros, and N. A. Smith. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*, 2016.

[26] M. Feng, B. Xiang, M. R. Glass, L. Wang, and B. Zhou. Applying deep learning to answer selection: A study and an open task. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 813–820. IEEE, 2015.

[27] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[28] J. Ganitkevitch, B. Van Durme, and C. Callison-Burch. PPDB: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics*, 2013.

[29] O. Glickman, I. Dagan, and M. Koppel. Web based probabilistic textual entailment. In *Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment*, 2005.

[30] A. Graves. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[31] B. F. Green Jr, A. K. Wolf, C. Chomsky, and K. Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224. ACM, 1961.

[32] J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

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

[34] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, 2015.

[35] D. Hewlett, A. Lacoste, L. Jones, I. Polosukhin, A. Fandrianto, J. Han, M. Kelcey, and D. Berthelot. WIKIREADING: A novel large-scale language understanding task over Wikipedia. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

[36] F. Hill, A. Bordes, S. Chopra, and J. Weston. The Goldilocks principle: Reading children's books with explicit memory representations. In *Proceedings of the International Conference on Learning Representations*, 2016.

[37] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[38] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*, 2014.

[39] L. Huang. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of the Conference on Association for Computational Linguistics*, 2008.

[40] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

[41] R. Kadlec, M. Schmid, O. Bajgar, and J. Kleindienst. Text understanding with the attention sum reader network. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

[42] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

[43] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

[44] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of the International Conference on Machine Learning*, 2016.

[45] C. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. *ACM Transactions on Information Systems (TOIS)*, 19(3):242–262, 2001.

[46] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy. RACE: Large-scale reading comprehension dataset from examinations. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2017.

[47] T. Lei, R. Barzilay, and T. Jaakkola. Rationalizing neural predictions. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

[48] T. Lei, W. Jin, R. Barzilay, and T. Jaakkola. Deriving neural architectures from sequence and graph kernels. *International Conference on Machine Learning*, 2017.

[49] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

[50] B. MacCartney. *Natural Language Inference*. PhD thesis, Stanford University, 2009.

[51] B. MacCartney, M. Galley, and C. D. Manning. A phrase-based alignment model for natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2008.

[52] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, and R. Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, 2014.

[53] Y. Mehdad, A. Moschitti1, and F. M. Zanzotto. SemKer: Syntactic/semantic kernels for recognizing textual entailment. In *Proceedings of the Text Analysis Conference*, 2009.

[54] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[55] L. Mou, R. Men, G. Li, Y. Xu, L. Zhang, R. Yan, and Z. Jin. Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

[56] K. Narasimhan, A. Yala, and R. Barzilay. Improving information extraction by acquiring external evidence with reinforcement learning. In *Proc. of EMNLP*, 2016.

[57] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. MS MARCO: a human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

[58] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

[59] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

[60] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[61] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

[62] M. Richardson, C. J. Burges, and E. Renshaw. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013.

[63] S. Robertson, H. Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[64] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, and P. Blunsom. Reasoning about entailment with neural attention. In *Proceedings of the International Conference on Learning Representations*, 2015.

[65] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, and P. Blunsom. Reasoning about entailment with neural attention. In *Proceedings of the International Conference on Learning Representations*, 2016.

[66] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. In *Proceedings of the International Conference on Learning Representations*, 2017.

[67] L. Shen, A. Sarkar, and F. J. Och. Discriminative reranking for machine translation. In *Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics*, 2004.

[68] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013.

[69] P. Soham, S. Ananya, N. Preksha, and M. K. Mitesh. Eliminet: A model for eliminating options for reading comprehension with multiple choice questions. *Openreview*, 2017.

[70] S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2015.

[71] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the Conference on Association for Computational Linguistics*, 2015.

[72] M. Tan, C. dos Santos, B. Xiang, and B. Zhou. Improved representation learning for question answer matching. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

[73] M. Tan, C. d. Santos, B. Xiang, and B. Zhou. Lstm-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv:1511.04108*, 2015.

[74] D. Tang, B. Qin, and T. Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

[75] M. Tapaswi, Y. Zhu, R. Stiefelhagen, A. Torralba, R. Urtasun, and S. Fidler. MovieQA: Understanding stories in movies through question-answering. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[76] A. Trischler, Z. Ye, X. Yuan, J. He, P. Bachman, and K. Suleman. A parallel-hierarchical model for machine comprehension on sparse data. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

[77] A. Trischler, Z. Ye, X. Yuan, J. He, P. Bachman, and K. Suleman. A Parallel-Hierarchical model for machine comprehension on sparse data. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

[78] A. Trischler, Z. Ye, X. Yuan, and K. Suleman. Natural language comprehension with the EpiReader. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

[79] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2015.

[80] E. M. Voorhees. The trec-8 question answering track report. In *Trec*, volume 99, pages 77–82, 1999.

[81] E. M. Voorhees and D. M. Tice. Building a question answering test collection. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 200–207. ACM, 2000.

[82] S. Wan, M. Dras, R. Dale, and C. Paris. Using dependency-based features to take the para-farce out of paraphrase. In *Proceedings of the Australasian Language Technology Workshop*, volume 2006, 2006.

[83] S. Wan, Y. Lan, J. Xu, J. Guo, L. Pang, and X. Cheng. Match-srnn: Modeling the recursive matching structure with spatial RNN. *International Joint Conference on Artificial Intelligence*, 2016.

[84] B. Wang, K. Liu, and J. Zhao. Inner attention based recurrent neural networks for answer selection. In *Proceedings of the Conference on Association for Computational Linguistics*, 2016.

[85] M. Wang, N. A. Smith, and T. Mitamura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2007.

[86] S. Wang and J. Jiang. Learning natural language inference with LSTM. In *Proceedings of the Conference on the North American Chapter of the Association for Computational Linguistics*, 2016.

[87] S. Wang and J. Jiang. A compare-aggregate model for matching text sequences. In *Proceedings of the International Conference on Learning Representations*, 2017.

[88] S. Wang and J. Jiang. Machine comprehension using match-LSTM and answer pointer. In *Proceedings of the International Conference on Learning Representations*, 2017.

[89] S. Wang, M. Yu, S. Chang, and J. Jiang. A co-matching model for multi-choice reading comprehension. In *Proceedings of the Conference on Association for Computational Linguistics*, 2018.

[90] S. Wang, M. Yu, X. Guo, Z. Wang, T. Klinger, W. Zhang, S. Chang, G. Tesauro, B. Zhou, and J. Jiang. R3: Reinforced reader-ranker for open-domain question answering. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2017.

[91] S. Wang, M. Yu, J. Jiang, W. Zhang, X. Guo, S. Chang, Z. Wang, T. Klinger, G. Tesauro, and M. Campbell. Evidence aggregation for answer re-ranking in open-domain question answering. *Proceedings of the International Conference on Learning Representations*, 2017.

[92] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the Conference on Association for Computational Linguistics*, 2017.

[93] Z. Wang, H. Mi, W. Hamza, and R. Florian. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211*, 2016.

[94] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. In *Proceedings of the International Conference on Learning Representations*, 2016.

[95] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *Proceedings of the International Conference on Learning Representations*, 2015.

[96] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

[97] C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. In *Proceedings of the International Conference on Learning Representations*, 2017.

[98] Y. Xu, J. Liu, J. Gao, Y. Shen, and X. Liu. Towards human-level machine reading comprehension: Reasoning and inference with multiple strategies. *arXiv preprint arXiv:1711.04964*, 2017.

[99] Y. Yang, W.-t. Yih, and C. Meek. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

[100] W.-t. Yih, M.-W. Chang, X. He, and J. Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China, July 2015. Association for Computational Linguistics.

[101] W. Yin, S. Ebert, and H. Schütze. Attention-based convolutional neural network for machine comprehension. *arXiv preprint arXiv:1602.04341*, 2016.

[102] W. Yin, H. Schütze, B. Xiang, and B. Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *arXiv preprint arXiv:1512.05193*, 2015.

[103] M. Yu, W. Yin, K. S. Hasan, C. d. Santos, B. Xiang, and B. Zhou. Improved neural relation detection for knowledge base question answering. *Proceedings of the Conference on Association for Computational Linguistics*, 2017.

[104] Y. Yu, W. Zhang, K. Hasan, M. Yu, B. Xiang, and B. Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*, 2016.

[105] H. Zhou, W. Furu, Q. Bing, and L. Ting. Hierarchical attention flow for multiple-choice reading comprehension. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.