ABSTRACT

Title of dissertation: ARCHITECTURE, MODELS, AND ALGORITHMS FOR TEXTUAL SIMILARITY

Hua He, Doctor of Philosophy, 2018

Dissertation directed by: Professor Jimmy Lin
College of Information Studies

Identifying similar pieces of texts remains one of the fundamental problems in computational linguistics. This dissertation focuses on the textual similarity measurement and identification problem by studying a variety of major tasks that share common properties, and presents our efforts to address 7 closely-related similarity tasks given over 20 public benchmarks, including paraphrase identification, answer selection for question answering, pairwise learning to rank, monolingual/cross-lingual semantic textual similarity measurement, insight extraction on biomedical literature, and high performance cross-lingual pattern matching for machine translation on GPUs.

We investigate how to make textual similarity measurement more accurate with deep neural networks. Traditional approaches are either based on feature engineering which leads to disconnected solutions, or the Siamese architecture which treats inputs independently, utilizes single representation view and straightforward similarity comparison. In contrast, we focus on modeling stronger interactions between inputs and develop interaction-based neural modeling that explicitly encodes the alignments of input words or aggregated sentence representations into our models. As a result, our multiple deep

neural networks show highly competitive performance on many textual similarity measurement public benchmarks we evaluated.

Our multi-perspective convolutional neural networks (MPCNN) uses a multiplicity of perspectives to process input sentences with multiple parallel convolutional neural networks, is able to extract salient sentence-level features automatically at multiple granularities with different types of pooling. Our novel structured similarity layer encourages stronger input interactions by comparing local regions of both sentence representations. This model is the first example of our interaction-based neural modeling.

We also provide an attention-based input interaction layer on top of the MPCNN model. The input interaction layer models a closer relationship of input words by converting two separate sentences into an inter-related sentence pair. This layer utilizes the attention mechanism in a straightforward way, and is another example of our interaction-based neural modeling.

We then provide our pairwise word interaction model with very deep neural networks (PWI). This model directly encodes input word interactions with novel pairwise word interaction modeling and a novel similarity focus layer. The use of very deep architecture in this model is the first example in NLP domain for better textual similarity modeling. Our PWI model outperforms the Siamese architecture and feature engineering approach on multiple tasks, and is another example of our interaction-based neural modeling.

We also focus on the question answering task with a pairwise ranking approach. Unlike traditional pointwise approach of the task, our pairwise ranking approach with the use of negative sampling focuses on modeling interactions between two pairs of question

and answer inputs, then learns a relative order of the pairs to predict which answer is more relevant to the question. We demonstrate its high effectiveness against competitive previous pointwise baselines.

For the insight extraction on biomedical literature task, we develop neural networks with similarity modeling for better causality/correlation relation extraction, as we convert the extraction task into a similarity measurement task. Our approach innovates in that it explicitly models the interactions among the trio: named entities, entity relations and contexts, and then measures both relational and contextual similarity among them, finally integrate both similarity evaluations into considerations for insight extraction. We also build an end-to-end system to extract insights, with human evaluations we show our system is able to extract insights with high human acceptance accuracy.

Lastly, we explore how to exploit massive parallelism offered by modern GPUs for high-efficiency pattern matching. We take advantage of GPU hardware advances and develop a massive parallelism approach. We firstly work on *phrase-based* SMT, where we enable phrase lookup and extraction on suffix arrays to be massively parallelized and vastly many queries to be carried out in parallel. We then work on computationally expensive *hierarchical* SMT model, which requires matching grammar patterns that contain "gaps". In order to get high efficiency for the similarity identification task on GPUs, we show developing massively parallel algorithms on GPUs is the most important approach to fully utilize GPU's raw processing power, and developing compact data structures on GPUs is helpful to lower GPU's memory latency. Compared to a highly-optimized, state-of-the-art multi-threaded CPU implementation, our techniques achieve orders of magnitude improvement in terms of throughput.

ARCHITECTURE, MODELS, AND ALGORITHMS FOR TEXTUAL SIMILARITY

BY

HUA HE

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Professor Jimmy Lin, Chair/Advisor
Professor James Allen Reggia
Professor Alan Sussman
Professor Marine Carpuat
Professor Hector Corrada Bravo

Acknowledgments

I owe my utmost gratitude to all the people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost, I would like to thank my advisor, Professor Jimmy Lin. He gives me great amount of flexibility to work on the directions I feel totally passionate about; he works closely with me for all paper deadlines, no matter weekends or midnight; he introduced me great external research collaborators I had fortunate to work with and learn from. Moreover, his passion on research advancement influences me deeply. I had great time as an "explorer" in his team just to explore what I am truly interested in, all thanks to Jimmy's great insights and patience. I think my PhD experience is somewhat unique, I am indebted to him for providing me the environment and the freedom to grow. I feel very fortunate to work with Jimmy.

I would like to thank Professor Adam Lopez at University of Edinburgh for his advice and inputs on everything. Adam gave me great guidance during my early days on the machine translation with GPU project.

I would like to thank Professor Kevin Gimpel at Toyota Technological Institute at Chicago. It was great to meet Kevin in the CLIP lab for the first time and he taught me many things on deep neural network research. Kevin gave me great support to help me write my very first deep learning paper.

I am fortunate to be part of the Computational Linguistics and Information Processing Lab (CLIP), where all the professors and students are open-minded, creative and helpful. There are a lot of colleagues in the lab that I should give a special mention.

Thank Professor Philip Resnik and Professor Douglas Oard for their advice; thank Joe Webster for technical support; thank my labmates, Jinfeng Rao, Weiwei Yang, Xing Niu, Junhui Li, Ke Wu, Anupam Guha, Mossaab Bagdouri, Chen Zhao and Feng Shi for their friendship. I would like to acknowledge support from staff members: Jennifer Story and Tom Hurst.

I would like to thank Professor Alan Sussman, Professor James Reggia, Professor Hector Corrada Bravo, and Professor Marine Carpuat for serving on my dissertation committee and spending their invaluable time reviewing my dissertation draft and providing me their insights.

This dissertation cannot be done without the help from my intern companies where I learned many things and had wonderful experiences for three summers. I love all my exciting internship projects. I want to take this chance to thank my mentors, Navendu Jain, Kris Ganjam, Jessica Lundin, Ryan White, Jacob Devlin, Hany Hassan at Microsoft Research Redmond, and Vivek Kumar at ATT-Labs Research.

Finally, I owe my deepest thanks to my parents. Words simply cannot express my gratitude. This dissertation is therefore dedicated to my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

The process of similarity measurement and identification is fundamental in human thinking. It lets us identify similar structures of different events by transferring knowledge from a known domain to an unknown domain. For example, when we begin to learn new things, we connect what we want to learn to what we have already learned, and involve similarity comparisons between one well-understood item and one unfamiliar item. The similarity measurement and identification process is everywhere, in our thoughts, in our daily lives, in popular movies, in advertisement promotions, in scientific discoveries, and even in young children's dreams. It plays a major role in human thinking, reasoning, problem solving, language use, learning and more. Early research studies [51, 104] suggest that this process "pervades all our thinking, our everyday speech and our trivial conclusions as well as artistic ways of expression and the highest scientific achievements".

Over the past two decades, the vast growth of internet and social media have led to a tremendous increase in user-generated web texts, most of which are frequently updating and noisy. As compelling moments, trends and events occur, we would like to capture and compare them as soon as possible and as best as we can. In this thesis, we focus on the textual similarity problem in the area of computational linguistics and present our approaches with novel models, algorithms, architecture, and systems.

1.1 A List of Textual Similarity Tasks We Addressed

There is a long list of real-world textual similarity tasks.

For example, paraphrase identification and plagiarism detection tasks are classical textual similarity measurement problems [36, 59, 166], as we need to know whether two provided sentences/paraphrases express similar meanings; learning to rank [110, 154], a fundamental task in Information Retrieval (IR), requires the similarity measurement between user queries and documents for effective searching purpose; answer selection in question answering task [38, 56, 84] needs to obtain the most relevant answers given user questions; and once inputs involve multiple languages, we then have cross-lingual similarity measurement task [3, 149], and cross-lingual similarity identification task for machine translation [58, 60, 89]. Obviously, this list can go much longer.

The focus of this dissertation is not just on one or two specific tasks, but on a variety of tasks that share common textual similarity properties. This dissertation presents our efforts to address the following 7 major tasks, which are based on over 20 public benchmark datasets we either used or created [1] for training and evaluation purpose. The first 6 tasks are on textual similarity measurement, and the last task is on textual similarity identification, as below:

[1] Datasets with $(\star)$ notation were created by the author of this dissertation.

- **Paraphrase Identification**
  - Datasets: MSRP [36], TwitterPPDB($\star$) [80]
- **Answer Selection for Question Answering**
  - Datasets: TrecQA [157], WikiQA [168], TREC 2011/2012 Microblogs [112]
- **Semantic Textual Similarity Measurement**
  - Datasets: MSRVID [2], STS2014(5) [3], STS2015(5) [4], SICK [94]
- **Pairwise Learning to Rank**
  - Datasets: TrecQA [169], WikiQA [168]
- **Insight Extraction on Biomedical Literature**
  - Datasets: SemEval2010 [65], Biomedical Literature Dataset($\star$) [62]
- **Cross-lingual Textual Similarity Measurement**
  - Dataset: STS2017(4) [21]
- **Cross-lingual Pattern Matching for Machine Translation on GPU**
  - Datasets: UN/FBIS [58], NIST2002-2016

The above list serves as a recipe of the entire dissertation. In following chapters, we introduce the tasks, show their challenges, and provide our novel approaches to tackle them.

1.2 Challenges

Since we focus on a variety of textual similarity tasks, there are unique challenges we must deal with.

Many of the benchmarks we created and used are sourced from different domains, ranging from standard language study domains such as news and Wikipedia, to noisier ones, such as Twitter, microblogs, and biomedical literature. The variations of text in vocabulary, format, styles, and areas make the problem-solving process challenging. There is therefore a great need to develop robust learning techniques that have good generalization ability with less cross-domain issues.

Since textual similarity is a fundamental concept, there are many related tasks we worked on as shown in previous sections. Traditionally these similarity-related tasks require diverse machine learning classifiers, hand-crafted feature sets, techniques and algorithms [34, 54, 55, 57]. Those differences unfortunately lead to highly disconnected solutions. For example, in the past IR researchers on learning to rank task dominantly prefer to use TF-IDF or BM25 [124] for word matching and weighting, however these sparse features are rarely used by NLP researchers for paraphrase identification or answer selection tasks, despite the fact that the ranking and paraphrase identification tasks are closely related to each other with common textual similarity properties.

There is therefore one important question to be asked. In textual similarity, why traditionally there exist separate solutions (e.g., different feature sets) for the tasks, despite the fact these tasks are closely related? We believe this is due to the way how researchers inject their prior domain knowledge into the problem-solving process. Especially given

traditional feature engineering techniques, different approaches of prior knowledge incorporation inevitably lead to separate solutions. However, the traditional approaches are not optimal as they are labor intensive and require diverse sets of techniques.

In this dissertation, we instead focus on model engineering approach for textual similarity tasks. The difference between model engineering and feature engineering approach lies in the way how the prior domain knowledge is utilized. This dissertation provides our working examples, and show our interaction-based neural modeling is able to better incorporate prior knowledge for many textual similarity tasks and produces improved results. We discuss its details in following sections.

It must be noted that the approaches discussed in this dissertation are purely data-driven. That is, we do no provide our own definitions of "semantically similarity" vs. "relatedness", or "causality" vs. "correlation", and we do not develop new theories to distinguish those. Our models only learn from the annotations based on annotators' understanding of the problem, which may differ among many public benchmarks due to different underlying annotation guidelines.

1.3 The Need for Better Prior Knowledge Incorporation to Similarity Modeling

Given two input sentences, documents or a pair of tweets, the problem of textual similarity measurement is to measure the semantic similarity between them. This dissertation focuses on a variety of similarity-related tasks but not individual ones, as these tasks essentially share common properties. However the key question is how to incorporate prior knowledge of these tasks into novel similarity modeling.

Prior knowledge represents our understandings of the tasks we aim to address. In many cases, the prior knowledge is the key to inspire novel solutions.

1.3.1 From Feature Engineering to Model Engineering

Previously feature engineering was the dominant approach to incorporate prior knowledge, primarily by creating hand-crafted sparse features. Unfortunately, the feature engineering approach lead to disconnected solutions given different tasks, thereby making problem-solving process labour intensive.

This is clearer with examples. Suppose we would like to measure the similarity distance between the two provided sentences:

1. *The economy fails because of the bank's bad decision*.

2. *The bank's bad decision causes the economy to fail*.

Sparse features that exploit string surface information can be created for textual similarity measurement. Typical choices include common n-gram word overlaps across

two sentences, or in a finer granularity, n-gram character overlaps, e.g., character-level edit distance. Such choices represent typical feature engineering approach. In the above example, we observe significant word and character overlaps, such as *economy, bad decision, the bank's, fail*. It is therefore apparent to see the more overlaps in words and characters the two sentences have, the higher their semantic similarity could be. Here the overlap information represents the prior knowledge, sparse features are used to inject the prior knowledge.

However, things get trickier when there exist synonyms and word overlap measurement is not enough for similarity measurement. For example, given the following slightly-modified pair of sentences,

1. *The economy is collapsing due to the bank's poor judgment*.

2. *The bank's bad decision causes the economy to crumble*.

Now the word and character overlaps are lower and string matches are no longer good indicators of textual similarity. In this example, *collapsing* should be matched to *crumble* and *bad decision* should be aligned to *poor judgment*. Traditional approaches [40, 41] utilize external lexical resources (such as WordNet) and develop knowledge-based sparse features to deal with this problem.

Other than string surface overlap features and external knowledge-based features, there are also syntax-based features [34] that use syntactic parsers to model divergence of syntax between two sentences, and corpus-based features [54, 55, 57] with latent semantic analysis, and more. Finally, a feature engineering system can be built to combine those sparse features from different aspects, then train a supervised model to approximate

ground-truth similarity scores.

Sparse features are the primary route in traditional feature engineering approach to encode prior knowledge. In Table 1.1 we show one competition-winning system [10] in 2012 that utilizes many types of features for textual similarity measurement, including surface string surface features, knowledge-based features, stylistic features which compare compares function word frequencies, and also phonetic features which conduct pairwise phonetic comparisons of words, each of these sparse features represents one part of the prior knowledge from researchers.

However, there are challenges that many of the sparse features cannot deal with completely. Challenges include but are not limited to: Elaboration (textual pairs can differ in total information content, such as *POTUS* and *President of the United States*); Phrasal (alternates of phrases, such as *taking over* and *replaces*), Spelling (spelling variants, such as *Trump* and *Trumpf*); Anaphora (a full noun phrase in one sentence that corresponds to the counterpart, such as *@MarkKirk* and *Kirk*); and Reordering (when a word, phrase or the whole sentence reorders, or logically reordered, such as *Matthew Fishbein questioned him* and *under questioning by Matthew Fishbein*).

Also, disadvantages with these sparse features exist. First of all, the development of many sparse features is labor intensive. Second, many of the sparse features turn out to be less useful in other related tasks, which lead to disconnected solutions. For example, learning to rank system rarely utilizes many of those features despite the ranking task and textual similarity task are closely related, which is why in the past NLP researchers consider NLP area to be "fragile". Thirdly, those sparse features rely heavily on external resources, e.g., WordNet, Wikitionary, POS features and syntactic parsers as in Table 1.1.

Sparse Features
Longest common subsequence (2 normalizations)
Longest common substring
Character 2-,3-, and 4-grams Word 1- and 2-grams (Containment, w/o stopwords)
Word 1-, 3-, and 4-grams (Jaccard)
Word 2- and 4-grams (Jaccard, w/o stopwords)
Word Similarity on WordNet aggregated
Greedy String Tiling
Explicit Semantic Analysis (Wikipedia, Wikitionary)
Distributional Thesaurus (POS: Cardinal numbers)

Table 1.1: Feature sets from a typical feature engineering system [10]

The feature engineering approach is just one way to encode prior knowledge, the other approach is model engineering, which is the focus of this dissertation. Model engineering and feature engineering mainly differ in the way how the prior knowledge of the problem is incorporated. Many recent work based on model engineering [29, 56, 59, 74, 76, 111] including ours, completely move away from hand-crafted features and toward modeling with distributed representations and neural network architectures.

In this dissertation, we introduce our model engineering approach that utilizes no sparse features, and require far less external resources than the traditional feature engineering approach, but still significantly outperform previous systems with rich sets of hand-crafted features. We obtain highly competitive performance on many of the tasks, rivaling or far exceeding the current state of the art.

1.3.2 From Siamese Architecture to Interaction-based Neural Modeling

When it comes to model engineering for textual similarity tasks, the Siamese neural network architecture [19] is the most common choice, as adopted by many recent work [74, 76, 137, 180]. The Siamese architecture consists of two sub-network models

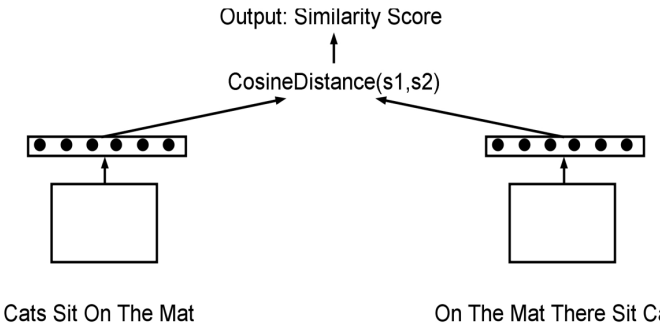


Figure 1.1: Overview of the Siamese architecture. Two input sentences (on the bottom) are processed in parallel by two identical neural networks, outputting sentence representations. The two sentence representations are then compared by a cosine distance metric.

that share weights, with each processing one input in parallel, as shown in Figure 1.1.

The sub-network models of the Siamese architecture firstly converts inputs into two vectors, which represent meanings of inputs. Then a similarity function is used (e.g., a popular choice is cosine function) to measure the similarity between vector representations. The Siamese architecture enjoys simplicity as neural networks are supposed to automatically look for important information of inputs and compare inputs in a low-dimensional vector space, in comparison to the feature engineering approach which requires manual development of hand-crafted features, as discussed in the previous section.

However, the Siamese architecture, is not perfect as it utilizes single view and straightforward similarity comparison, major challenges with the Siamese architecture are:

1. Modeling textual similarity is complicated by the ambiguity and variability of linguistic expression. Ambiguity means that a single sentence can express multiple meanings, while variability means that different phrases/words could express the same meaning. The difficulties require us during model engineering to tolerate

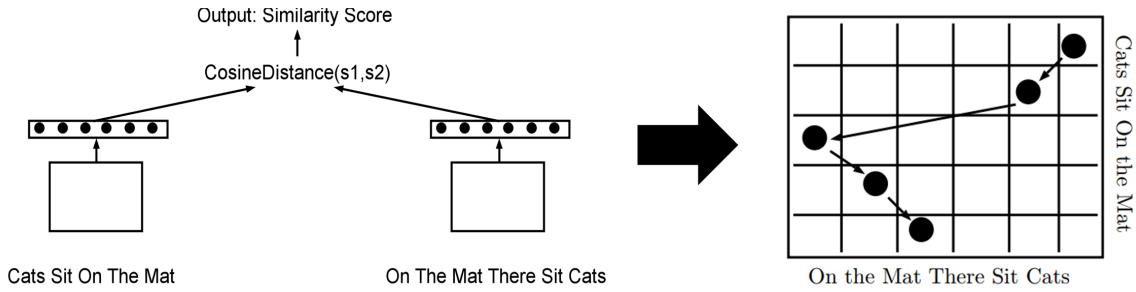


Figure 1.2: From Siamese architecture to our interaction-based neural modeling for textual similarity measurement. Our interaction-based neural networks focus on explicitly modeling various alignments of inputs, in contrast to the Siamese architecture which treats inputs separately.

such ambiguity and provide ways to model the ambiguity. Thus simply "cramming" all the meanings of the input [98] into one vector representation as in the Siamese architecture cannot work well.

2. Given two vector representations, the Siamese architecture uses straightforward comparison with a single similarity function (such as cosine or dot product), but it is not powerful enough to capture interactions between words and their polysemy and to measure their similarity effectively. For example, cosine similarity is only on directional similarity, which introduces bias.

The disadvantages of the Siamese architecture lead to our interaction-based neural modeling for better textual similarity modeling. We show both in Figure 1.2. Our interaction-based neural modeling treats the inputs as a whole but not as separate ones, and place both inputs "closer" in the low-dimensional vector space to encourage stronger interactions, in contrast to the Siamese architecture which treats inputs independently. Our models focus on explicitly modeling alignments of input words or aggregated sentence representations from inputs. We design our model to encode stronger interactions

of inputs into our models. The interaction-based neural modeling design reflects our understanding of textual similarity tasks, and shows the prior knowledge injection process to better address the problem.

In order to achieve higher degree of interactions between inputs, we develop novel interaction-based neural network layers and neural network models; we also utilize popular techniques by directly taking advantages of recent deep neural network progress, such as attention mechanism, very deep neural networks, and many recent innovations. As a result, our models significantly outperform models with the Siamese architecture and feature engineering systems on many textual similarity tasks, as will be shown in following sections.

1.4 Contributions and Structure

The dissertation focuses on a variety of textual similarity related tasks, including *paraphrase identification, answer selection for question answering, semantic textual similarity measurement, pairwise learning to rank, relation extraction, cross-lingual similarity measurement and identification*. We address each of the tasks as evaluated on over 20 public benchmarks.

Chapter 2 reviews related work of the 7 textual similarity tasks, and provides detailed descriptions of public benchmarks as used throughout this dissertation.

Major contributions of the dissertation are identified in this section, we group them by the tasks each corresponds to, listed below.

1. Paraphrase Identification

2. Answer Selection for Question Answering

3. Semantic Textual Similarity Measurement

4. Cross-lingual Textual Similarity Measurement

- In Chapter 3 we provide our first interaction-based neural network model, *multi-perspective convolutional neural network model*, guided by our intuitions to better model interactions between inputs, in contrast to the Siamese architecture or feature engineering approach which treats inputs independently. The model uses a multiplicity of perspectives to process input sentences with multiple parallel convolutional neural networks, it extracts salient sentence-level features automatically at multiple granularities and also uses different types of pooling. The extracted multi-

perspective sentence representations are then compared over local regions using multiple similarity metrics with our novel structured similarity comparison layer. We evaluate our model on multiple public benchmarks and obtained highly competitive performance on all. This model is our first example of interaction-based neural modeling.

- In addition, in Chapter 3.7 we provide an attention-based input interaction layer on top of our multi-perspective convolutional neural network model. The input interaction layer models a closer relationship of input words by converting separate sentences into an inter-related sentence pair. This layer utilizes the attention mechanism [8] in a straightforward way which leads to effective result improvement, and is another example of our interaction-based neural modeling. Note that the attention mechanism is only one of the many techniques to encourage interaction modeling between inputs. In this dissertation we also propose other novel techniques to enforce stronger interations between inputs as well.

- In Chapter 4 we provide our *pairwise word interaction model with very deep neural networks*. We directly encode input word interactions with novel pairwise word interaction modeling and our similarity focus layer. The use of very deep architecture in this model is also the first example in the NLP domain to our best knowledge for better textual similarity modeling. Our pairwise word interaction model significantly outperforms the Siamese architecture and feature engineering approach on multiple tasks, and is another example of our interaction-based neural modeling.

5. Pairwise Learning to Rank

- In Chapter 5 We focus on the question answering task with a pairwise ranking approach. Unlike traditional *pointwise* approach of the task, we use Noise-Contrastive Estimation (NCE) for the pairwise learning. Our pairwise ranking approach with the use of NCE focuses on modeling interactions between two pairs of question and answer inputs, then learns a relative order of the pairs to predict which answer is more relevant to the question. This is also a direct extension which utilizes both our *multi-perspective convolutional neural network model* and *pairwise word interaction model*. We demonstrate its high effectiveness of our approach against competitive pointwise baselines.

6. Insight Extraction on Biomedical Literature

- In Chapter 6 we focus on the task of insight extraction on biomedical literature with textual similarity modeling. We define "insights" of biomedical literature to consist of biomedical and health-related entities plus two specific types of relationships: (a) *cause-effect* and (b) *correlation*. We develop neural networks with novel similarity modeling for better relation extraction, as we convert the extraction task into a similarity measurement task. Our approach innovates in that it explicitly models the interactions among the trio: named entities, entity relations and contexts, and then measures both relational and contextual similarity among them, finally integrate both similarity evaluations for better insight extraction. Note our approaches

are purely data-driven. We do no provide our own definitions of *cause-effect* vs. *correlation*, or try to distinguish both with new theories. Our approach only learns from annotated data given annotators' understanding of the problem.

- We build an end-to-end system to extract such insights, and the system provides a novel combination of recognizing named entities, predicting relationships between extracted entities, and ranking the output. We conduct human evaluations of the system to show it is able to extract insights with high human acceptance accuracy.

7. Cross-lingual Pattern Matching for Machine Translation on GPU

- In Chapter 7 we focus on the textual similarity identification task, and introduce novel contiguous and approximate cross-lingual pattern matching algorithms on GPUs that can run on GPUs efficiently for machine translation. In order to get high efficiency for the similarity identification task on GPUs, we show developing massively parallel algorithms on GPUs is the most important approach to fully utilize GPU's raw processing power, and developing compact data structures on GPUs is also helpful to lower GPU's memory latency. As a result our massively parallel GPU-based approach is orders of magnitude faster than the previous state-of-the-art CPU baseline.

Finally, in support of open science, we have released most of our work [2] openly to the entire research community. This includes source codes of our many interaction-

[2] https://github.com/hohoCode

based neural network models and datasets. We hope these resources are helpful for future research in related areas.

Chapter 2: Background, Related Work, Tasks and Datasets

This chapter reviews related work of the textual similarity tasks we addressed, and introduces public benchmarks we used or created throughout this dissertation. We group the descriptions by the tasks each corresponds to in following sections.

2.1 Semantic Textual Similarity Measurement Tasks

Paraphrase Identification

Semantic Textual Similarity Measurement

Cross-lingual Textual Similarity Measurement

Identifying the degree of semantic textual similarity between two texts is at the crux of many NLP applications that address sentence/document level semantics. We consider the core problem as the semantic textual similarity measurement task (STS), where given two pieces of texts, we would like to know how similar they are in terms of a similarity score (e.g. between $[0, 5]$ or $[0, 1]$).

Paraphrase identification is a task closely related to STS. Given previous work [36, 54], paraphrase identification task is defined as identifying whether the two input texts are "semantically equivalent", but both may differ in syntactic structures or a number of

relevant details. Given this definition, the STS and paraphrase identification tasks closely resemble one another: if two texts are paraphrases, then the semantic similarity score should be high, and vice versa. One unique difference is if one text is the negation of the other, then they are not paraphrases, yet in STS they still have a relatively high similarity score.

Most previous work on modeling semantic textual similarity has focused on feature engineering. Several types of sparse features have been found useful, including: (1) string-based, including n-gram overlap features on both the word and character levels [150] and features based on machine translation evaluation metrics [91]; (2) knowledge-based, using external lexical resources such as WordNet [40, 41]; (3) syntax-based, e.g., modeling divergence of dependency syntax between the two sentences [34]; (4) corpus-based, using distributional models such as latent semantic analysis to obtain features [54, 55].

Several strongly-performing approaches used system combination [34, 91] or multi-task learning. Xu et al. [166] developed a feature-rich multi-instance learning model that jointly learns paraphrase relations between word and sentence pairs. On the opposite, our work in the second part of preliminary work does not utilize any hand-crafted features.

Recent work has moved away from hand-crafted features and towards modeling with distributed representations and neural network architectures. Collobert and Weston [29] used convolutional neural networks in a multi-task setting, where their model is trained jointly for multiple NLP tasks with shared weights. Kalchbrenner et al. [74] introduced a convolutional neural network for sentence modeling that uses dynamic k-max pooling to better model inputs of varying sizes. Kim [76] proposed several modifications

to the convolutional neural network architecture of Collobert and Weston [29], including the use of both fixed and learned word vectors and varying window sizes of the convolution filters.

For the Microsoft Research Paraphrase identification task (Section 2.1.1), Socher et al. [131] used a recursive neural network to model each sentence, recursively computing the representation for the sentence from the representations of its constituents in a binarized constituent parse. They train the model with an auto-encoder criterion that helps build sentence representations recursively for each node of a binarized constituent parse, a syntactically-informed approach to combine word embeddings with a recursive auto-encoder to propagate information through parse trees. Ji and Eisenstein [71] used matrix factorization techniques to obtain sentence representations, and combined them with fine-tuned sparse features using an SVM classifier for similarity prediction. Both Socher et al. and Ji and Eisenstein incorporated sparse features to improve performance, which we do not use in this work.

Hu et al. [68] used convolutional neural networks that combine hierarchical sentence modeling with layer-by-layer composition and pooling. While they performed comparisons directly over entire sentence representations, we instead develop a structured similarity measurement layer to compare local regions. A variety of other neural network models have been proposed for similarity tasks [6, 18, 69, 160].

Most recently, Tai et al. [137] and Zhu et al. [180] concurrently proposed a tree-based LSTM neural network architecture for sentence modeling, where they recursively construct sentence vectors following the structure of syntactic trees. Our approach differs in that we do not use any syntax since high-quality parsers could be harder to obtain for

low-resource languages or noisy domains (e.g., social media data or biomedical texts), yet our performance outperform Tai et al. [137] on the similarity task. This result is appealing because high-quality parsers are challenging to obtain for low-resource languages or specialized domains. Yin and Schütze [172] concurrently developed a convolutional neural network architecture for paraphrase identification, which we compare to in our experiments. Their best results rely on an unsupervised pre-training step, which we do not need to match their performance.

Typically we assume the STS and paraphrase identification task to be monolingual, that is, the two inputs are on same language, however there is a need to measure the textual similarity of texts with different languages, this leads us to cross-lingual STS task. Cross-lingual STS task [21, 139] is gaining popularity recently due to the vast growth of internet and social media, for example, a Spanish speaker would like to query web pages written in English, cross-lingual STS is then useful and important.

We group descriptions of the three tasks in this section as all are closely connected. We then introduce major public benchmarks we used for these tasks to evaluate our interaction-based models.

2.1.1 Datasets

We introduce eight major public benchmarks in this section.

1. Microsoft Research Paraphrase Corpus (MSRP). This data was collected from news sources [36] and contains 5,801 pairs of sentences, with 4,076 for training and the remaining 1,725 for testing. Each sentence pair is annotated with a binary label

STS2014	Domain	Pairs
deft-forum	discussion forums	450
deft-news	news articles	300
headlines	news headlines	750
images	image descriptions	750
OnWN	word sense definitions	750
tweet-news	social media	750
Total		**3,750**

Table 2.1: Data statistics of STS2014 test sets.

indicating whether the two sentences are paraphrases, so the task here is binary classification.

2. Sentences Involving Compositional Knowledge (SICK) dataset. This data was collected for the 2014 SemEval competition [94] and consists of 9,927 sentence pairs, with 4,500 for training, 500 as a development set, and the remaining 4,927 in the test set. The sentences are drawn from image and video descriptions. Each sentence pair is annotated with a relatedness score $\in [1, 5]$, with higher scores indicating the two sentences are more closely-related.

3. Microsoft Video Paraphrase Corpus (MSRVID). This dataset was collected for the 2012 SemEval competition and consists of 1,500 pairs of short video descriptions which were then annotated [2]. Half of it is for training and the other half is for testing. Each sentence pair has a relatedness score $\in [0, 5]$, with higher scores indicating the two sentences are more closely-related.

4. Task 10 of the 2014 SemEval competition on Semantic Textual Similarity (STS2014) [3] provided six different test sets from different domains. Each pair has a similarity

STS2015	Domain	Pairs
answers-forums	forum	375
answers-students	student short answers	750
belief	belief annotations	375
headlines	news headlines	750
images	image descriptions	750
Test Total	-	**3,000**
Train Total	STS2012-2014	**11,342**

Table 2.2: Data statistics for SemEval STS 2015.

score $\in [0, 5]$ which increases with similarity. Following the competition rules, our training data is only drawn from previous STS competitions in 2012 and 2013. We excluded training sentences with lengths longer than the 48-word padding limit, resulting in 7,382 training pairs out of a total of 7,592. Table 2.1 provides a brief description of the six test sets.

5. The 2015 SemEval English STS competition (STS2015) [4] provides five test sets from diverse domains. We tokenize all data using Standford CoreNLP [93]. Each pair has a similarity score $\in [0, 5]$ which increases with similarity. The training data is from previous STS competitions from 2012 to 2014 following the rules of the STS2015 competition. Table 2.2 provides a brief description.

6. TwitterPPDB dataset [80] is the largest human-labeled paraphrase corpus to date (in the year of 2017) of 51,524 sentence pairs in total and the first cross-domain benchmarking for automatic paraphrase identification. Each pair has a similarity score $\in [1, 5]$ which increases with similarity. The dataset has 42,200 pairs for training and 9324 pairs for testing, and was collected on Twitter and manually annotated on Amazon Mechanical Turk.

STS2017	Track	Pairs
Arabic-English	Track 2	250
Spanish-English	Track 4a	250
Spanish-English (OOV)	Track 4b	250
Turkish-English	Track 6	250

Table 2.3: Data statistics of Cross-Lingual STS 2017 test sets.

7. The 2017 SemEval Task 1 on Semantic Textual Similarity Multilingual and Cross-lingual Focused Evaluation STS competition (STS2017) [21] provides four cross-lingual test sets for textual similarity measurement of short texts. These test sets are from Track 2, $4a$, $4b$ and 6 for Arabic-English, Spanish-English, Spanish-English (OOV) and Turkish-English sentence pairs. Each pair has a similarity score $\in [0, 5]$ which increases with similarity. The training data is from previous STS competitions from 2012 to 2015 following the rules of the competition. In order to test the monolingual similarity measurement ability of our models, we simply follow most competing systems in the SemEval 2017 competition: during preprocessing we firstly use Google machine translation API [1] to translate all non-English sentences into English.

2.2 Question Answering and Ranking Tasks

Answer Selection for Question Answering

Pairwise Learning to Rank

[1] https://translate.google.com

The task of answer selection in question answering consists of identifying pertinent answers from a pool of answers related to a question. A similarity score is given to each candidate answer as a function of its frequency of occurrence, and the final answer is selected from the set of candidates sorted in decreasing order of score.

There is a large pool of previous work on applying neural networks to answer selection tasks for question answering [56, 59, 61, 124, 152, 153, 176]. Most previous work are based on pointwise neural network models. In the following chapters we will introduce our own work based on pointwise models for the task. For example, we developed a multi-perspective convolutional neural network model [59] that incorporates fine-grained sentence representations. We also proposed word interaction model [56] which proposed 19-layer deep convolutional neural networks and a similarity focus layer to encourage comparisons between word contexts across sentences. Other researchers [61, 173] have adopted variants of attention mechanisms for convolutional neural networks to better model interactions between inputs. There are also previous work which utilize sparse features (e.g., BM25 or TFIDF) [124, 153].

The pointwise neural network models assume that each query-document pair in the training data has a numerical similarity score, then the learning-to-rank problem can be approximated by a regression problem given a single query-document pair, predict this score. However, the pairwise learning to rank neural network approach is one step beyond the pointwise models, the ranking problem is instead approximated by a classification problem, e.g. learning a binary classifier that can tell which document is better given a pair of documents. The goal is to minimize average number of inversions in ranking. Intuitively the pairwise learning to rank approach is a better fit for the nature of ranking

problems, such as the answer selection task. In the following chapter we will introduce our pairwise learning to rank approach to address the answer selection problem.

2.2.1 Datasets

We introduce major public benchmarks we utilized to train and evaluate our interaction-based models in this section.

1. The WikiQA question answering data set is from Bing query logs by Yang et al. [168]. For each question, the authors selected Wikipedia pages and used sentences in the summary paragraph as candidates, which were then annotated on a crowd-sourcing platform. We followed the same pre-processing steps as Yang et al. [168], where questions with no correct candidate answer sentences are excluded and answer sentences are truncated to 40 tokens. The resulting dataset consists of 873 questions with 8,672 question-answer pairs in the training set, 126 questions with 1,130 pairs in the development set, and 243 questions with 2,351 pairs in the test set.

2. The TrecQA question answering data set has been widely used for the answer selection task during the past decade. To enable direct comparison with previous work, we used the same training, development, and test sets as released by Yao et al. [169]. The TrecQA data consists of 1,229 questions with 53,417 question-answer pairs in the *TRAIN-ALL* training set, 82 questions with 1,148 pairs in the development set, and 100 questions with 1,517 pairs in the test set.

3. TREC 2011 and 2012 Microblog Track test collections [112] are collected on Twit-

ter for the TREC 2011 and 2012 Microblog Tracks (49 topics and 59 topics, respectively). Both use the Tweets2011 collection, which consists of an approximately 1% sample (after spam removal) of tweets from January 23, 2011 to February 7, 2011 (inclusive), totaling approximately 16 million tweets. Relevance judgments were made on a 3-point scale ("not relevant", "relevant", "highly relevant"). We treated both higher grades as relevant, also removed all the re-tweets in our experiments since they are by definition not relevant according to the assessment guidelines.

2.3 Insight Extraction on Biomedical Literature Task

Insight Extraction on Biomedical Literature

Health and medical research has never been more active and diverse than it is at this moment. Research efforts across national and cultural borders make new insights available at a scale with ever decreasing latency. In the face of these developments, individual researchers and practitioners are confronted with a seemingly intractable amount of material as over one million scholarly articles are newly published in the biomedical/health area each year. It would be highly beneficial if high precision insights extracted from existing biomedical/health literature can be provided.

This insight extraction task focused on extracting biomedical insights, which we define as a pair of biomedical entities with either causality or correlation relationship. We focus on extracting the two relationship types from them. It is motivated by the need to better automate biomedical knowledge extraction and identify important information

from them, as new scientific findings appear across a large collection of publications.

We will show the insight extraction task is deeply related to textual similarity measurement task in Chapter 6, where we develop an end-to-end insight extraction system and novel similarity modeling to address this task. Our system also provides a novel combination of recognizing named entities, predicting relationships (insights) between extracted entities, and ranking the output.

In terms of related work, most previous BioNLP work focused on extraction of biomedical concepts only [31, 44, 85, 106, 128], such as drug or protein names. We also conduct relation extraction on general named entities, such as "*smoking*" or "*sleep quality*". Kabiljo et al. [73] compared pattern-matching techniques against a baseline regular expression approach for gene/protein entity extraction. But existing tools for relation extraction are not as comprehensive as entity recognition tools. We also observe other related work on privacy preservation with healthcare datasets [165].

Medical dictionaries and resources are heavily utilized by previous work. For instance, Chen et al. [24] extracted disease-drug relation pairs with MedLEE [45] system for clinical information extraction of EHR records. Liu et al. [86] developed a text-mining system to search for associations among human diseases, genes, drugs, metabolites and toxins against large collections of text-rich biological databases. Previous research efforts also lead to semantic representation program SemRep [115], which exploits biomedical domain knowledge and linguistic analysis of biomedical texts. Other unconventional resource such as web query logs are also utilized [100] to provide early warnings about the presence of devastating diseases.

Feature engineering was the dominant approach in most biomedical relation ex-

traction work with machine learning techniques [35, 167]; different sparse features were explored. For example, word n-gram features, knowledge-based features from medical dictionaries and word position features. We instead propose neural network models that do not require sparse features as in most previous work.

Recent shift from feature engineering to model engineering with neural networks has significantly improved accuracy on many NLP tasks. Jagannatha and Yu [70] adopted an LSTM model for medical entity detection given patient EHR records. There are recent work with the use of deep reinforcement learning on healthcare study [83]. There were recent embedding learning work to jointly represent texts and knowledge base [141, 142], on embedding transfer learning [16] and noise-contrastive estimation [110].

2.3.1 Datasets

We introduce the two datasets we utilized to train and evaluate our interaction-based similarity models in this section: our own dataset of medical/health publications annotated on Universal Human Relevance System (UHRS), a crowdsourcing platform for the end-to-end system evaluation; and SemEval-2010 task 8 dataset for training and evaluation of our relation extraction component:

1. Our own annotated biomedical literature dataset consists of 100 publications from recent biomedical/health journals, which are then annotated on UHRS to evaluate our system. In order to ensure high-quality human annotations, Figure 2.1 provides an annotation interface on UHRS, which displays instructions, title/abstract texts of publications and a list of top-ranked extracted insights from the system output.

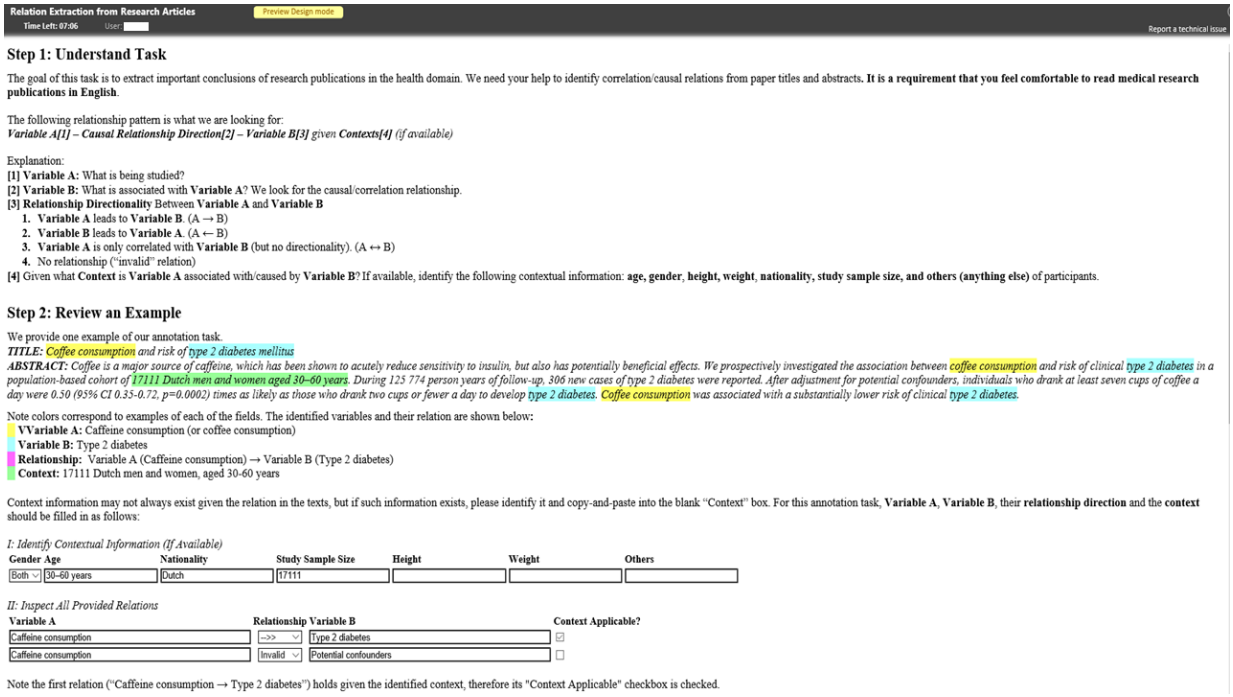


Figure 2.1: Human annotation interface on UHRS platform. Annotators are required to identify and verify extracted entities and correlation/causality relations from the output of our insight extraction system for evaluation.

For fair evaluation the order of extracted insights is randomized then we ask expert annotators with suitable background to verify the correctness of each.

2. SemEval-2010 Task 8 [64] defines 9 relation types between named entities: *Cause-Effect, Instrument-Agency, Product-Producer, Content-Container, Entity-Origin, Entity-Destination, Component-Whole, Member-Collection and Message-Topic*, and a tenth relation type *Other* when two named entities do not have the first 9 relations. SemEval-2010 dataset consists of $10,717$ sentences, with $8,000$ for training and $2,717$ for test. The dataset is human annotated, and each instance provides one sentence which includes two named entities and a relation type between the two entities.

   Since our system focuses on extracting insights, we only use *Cause-Effect* subset

of SemEval-2010 dataset as the positive training/testing examples and treat the remaining 9 categories data such as *Content-Container, Message-Topic* as negatives. We use this dataset for training and evaluating our relation extraction component (Sec. 6.3) only.

2.4 Similarity Identification Task

Cross-lingual Pattern Matching for Machine Translation on GPU

Lastly, we introduce the similarity identification task of cross-lingual pattern matching for machine translation on GPU. Machine translation models can be scaled to large corpora and arbitrarily-long phrases by looking up translations of source phrases on the fly in an indexed parallel corpus using suffix arrays. However, this can be slow because on-demand extraction of phrase tables is computationally expensive. In comparison to similarity measurement tasks where we focus on high quality, for the similarity identification task we move our focus to the efficiency improvement. In Chapter 7 we will show how to address this task by developing novel massively parallel algorithms for general purpose graphics processing units (GPU), which enable suffix array queries for phrase lookup and grammar extraction of machine translation to be massively parallelized.

In following sections, we firstly introduce the background and related work of graphical processing units and suffix array operations for pattern matching, then describe trade-offs between two traditional machine translation architectures. The trade-offs between the two SMT architectures ultimately lead to this similarity identification task.

2.4.1 Graphical Processing Units

Despite Moore's Law and the steady increase of transistor density over the past several decades, there is a consensus among microprocessor architects that we have reached the limits of instruction level parallelism. Beginning in the middle of the last decade, increased transistor counts have primarily been devoted to placing multiple cores on the same die, bringing us into the multi- and many-core era. GPUs are a direct beneficiary of this trend, as vendors have been able to pack an increasing number of cores onto these devices. The advent of general purpose GPUs meant that these processors were no longer limited to high-performance graphics applications, but can be viewed as massively parallel stream processors. We therefore seek to exploit the massive parallelism offered by GPUs to accelerate the contiguous and approximate pattern matching problem.

The GPU architecture is optimized for massively parallel operations on relatively small streams of data, and it also strongly influences the design of parallel algorithms. For example one of the GPUs we are using in our experiments is a NVIDIA Tesla K20c GPU (Kepler Generation), which provides 2496 thread processors (CUDA cores), and its computation proceeds concurrently in groups of 32 threads called *warps*. Each thread in a warp carries out identical instructions in lockstep. When a branching instruction occurs, only threads meeting the branch condition execute, while the rest idle—this is called *warp divergence* and is a source of poor performance. Consequently, GPUs are poor at "irregular" computations that involve conditionals, pointer manipulation, and complex execution sequences. Moreover, although GPUs have relatively high memory bandwidth (over 200Gb/second in K20c) from its RAM, but the bandwidth is shared across all threads.

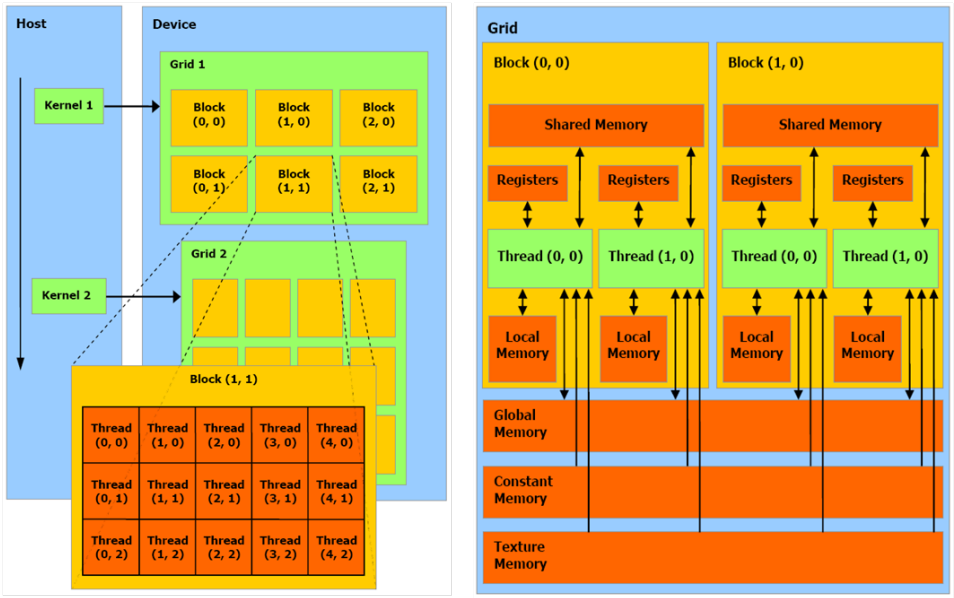


Figure 2.2: Typical CUDA Program Model

When there are tens of thousands threads running concurrently and all request data from the GPU RAM, the memory bandwidth could be limited. Therefore it is beneficial to design parallel GPU algorithms with small memory footprint.

Compute Unified Device Architecture (CUDA) is a parallel computing library for GPU programming. CUDA programming model is shown in Figure 2.2 where GPU organize threads in hierarchy from global thread grid to relatively local thread warps. The GPU memory hierarchy includes many layers, each of which represents different speed/storage trade-offs. For example registers inside GPU can be as fast as several terabytes/second but can only cache small amount of data, while the GPU global memory has a capacity of $2 - 12$ GB which could be limited given large scale application [177] and is also relatively slow in terms of memory access speed.

2.4.2 Suffix Array and Suffix Array Lookup

Although there are many algorithms for high performance text matching, there is one efficient data structure available, particularly used for statistical machine translation. This data structure is *suffix arrays*.

We briefly review the classic algorithms of Manber and Myers [92] here since they form the basis of techniques and analysis in our first work in Section 3.

A suffix array represents all suffixes of a textual data corpus in lexicographical order. Formally, for a text T, the ith suffix of T is the substring of the text beginning at position i and continuing to the end of T. Each suffix can therefore be uniquely identified by the index i of its first word. A suffix array $S(T)$ of T is a permutation of these suffix identifiers $[1, |T|]$ arranged by the lexicographical order of the corresponding suffixes—in other words, the suffix array represents a sorted list of all suffixes in T. With both T and $S(T)$ in memory, we can find any query pattern Q in $O(|Q| \log |T|)$ time by comparing pattern Q against the first $|Q|$ characters of up to $\log |T|$ different suffixes using binary search.

An inefficiency in this solution is that each comparison in the binary search algorithm requires comparing all $|Q|$ characters of the query pattern against some suffix of text T. We can improve on this using an observation about the *longest common prefix* (LCP) of the query pattern and the suffix against which it is compared. Suppose we search for a query pattern Q in the span of the suffix array beginning at suffix L and ending at suffix R. For any suffix M which falls lexicographically between those at L and R, the LCP of Q and M will be at least as long as the LCP of Q and L or Q and R. Hence if we know

the quantity $h = \textsc{Min}(\textsc{Lcp}(Q, L), \textsc{Lcp}(Q, R))$ we can skip comparisons of the first h symbols between Q and the suffix M, since they must be the same.

The solution of Manber and Myers [92] exploits this fact along with the observation that each comparison in binary search is carried out according to a fixed recursion scheme: a query is only ever compared against a specific suffix M for a single range of suffixes bounded by some fixed L and R. Hence if we know the longest common prefix between M and each of its corresponding L and R according to the fixed recursions in the algorithm, we can maintain a bound on h and reduce the aggregate number of symbol comparisons to $O(|Q| + \log |T|)$. To accomplish this, in addition to the suffix array, we precompute two other arrays of size $|T|$ for both left and right recursions (the LCP arrays).

i	$T[i]$	$S_T[i]$	suffix $S_T[i]$: $T[S_T[i]]...T[\|T\|]$
1	#	5	and it mars him # it sets him on and it takes him off # $
2	it	14	and it takes him off # $
3	makes	4	him and it mars him # it sets him on and it takes him off # $
4	him	17	him off # $
5	and	12	him on and it takes him off # $
6	it	8	him # it sets him on and it takes him off # $
7	mars	2	it makes him and it mars him # it sets him on and it takes him off # $
8	him	6	it mars him # it sets him on and it takes him off # $
9	#	10	it sets him on and it takes him off # $
10	it	15	it takes him off # $
11	sets	3	makes him and it mars him # it sets him on and it takes him off # $
12	him	7	mars him # it sets him on and it takes him off # $
13	on	18	off # $
14	and	13	on and it takes him off # $
15	it	11	sets him on and it takes him off # $
16	takes	16	takes him off # $
17	him	1	# it makes him and it mars him # it sets him on and it takes him off # $
18	off	9	# it sets him on and it takes him off # $
19	#	19	# $
20	$	20	$

Figure 2.3: Example text T (showing words rather than than their integer encodings for clarity) and suffix array S_T with corresponding suffixes.

To make our discussion concrete, we show a toy example in Figure 2.3. Suppose

that it contains two sentences: *it makes him and it mars him*, and *it sets him on and it takes him off*. We map each unique word to an integer id, and call the resulting array of integers that encodes the source text under this transformation the text T. Let $|T|$ denote the total number of tokens, $T[i]$ denote its ith element, and $T[i]...T[j]$ denote the substring starting with its ith element and ending with its jth element. Since T encodes multiple sentences, we use special tokens to denote the end of a sentence and the end of the corpus. In our example we use # and $, respectively, as shown in Figure 2.3. Now suppose we want to translate the sentence *it persuades him and it disheartens him*. We encode it under the same mapping as T and call the resulting array the query Q. Our goal is to materialize all of the phrases that could be used to translate Q based on training data T.

Memory use is an important consideration, since we want to apply this data structure onto GPUs which have less memory than CPUs. For the algorithms described here, we require four arrays: the original text T, the suffix array $S(T)$, and the two LCP arrays. We use a representation of T in which each word has been converted to a unique integer identifier; with 32-bit integers the total number of bytes is $16|T|$.

Recent work in computational biology has shown that suffix arrays are particularly amenable to GPU techniques: the suffix-array-based DNA sequence matching system MummurGPU++ [48] has been reported to outperform the already fast MummurGPU 2 [143] based on suffix trees (an alternative indexing structure).

As we will show in Chapter 7, this turns out to be quite modest, even for large parallel corpora. In Section 7.3 we will present a massively parallel algorithm using efficient suffix array lookups for high-performance text matching on GPUs.

2.4.3 Datasets

Since the similarity identification task is on the speed performance of our GPU algorithms on grammar extraction of SMT, we provide descriptions of the parallel corpus and the query test sets as used in our experiments.

1. We use the source (Chinese) side of news articles collected from the Xinhua Agency, with over 27 million words of Chinese in over one million sentences (totaling 137 MB), and we also add source-side parallel text from the United Nations, with over 81 million words of Chinese in over 4 million sentences (totaling 561 MB). In a pre-processing phase, we map every word to a unique integer, with two special integers, representing end-of-sentence and end-of-corpus, respectively.

2. Our test query set consists of all sentences from the NIST 2002–2006 translation campaigns, tokenized and integerized identically to the training data. On average, sentences were around 29 words. In order to fully stress our GPU algorithm, we ran tests on batches of 2,000, 4,000, 8,000, 16,000, or 32,000 sentences. Note that there are just over 8,000 test sentences total from the NIST data; we simply duplicated the test sets when necessary.

2.5 Summary

This chapter provides background of the tasks we addressed, also introduces the datasets we used for evaluation throughout the dissertation. Next in Chapter 3, we introduce our first interaction-based neural network model for textual similarity measurement.

Chapter 3: Interaction-based Multi-Perspective Modeling and Structured Similarity Layer

In this chapter we introduce our interaction-based multi-perspective neural network model [59, 61]. The problem the model addresses is formulated formally as: given a query sentence S_1 and a comparison sentence S_2, the task is to compute their similarity in terms of a score $sim(S_1, S_2)$. This similarity score can be used in many downstream applications, for example, determine whether the two sentences are paraphrases in the paraphrase identification task, or given an answer sentence rank corresponding question sentences in the answer selection task.

Measuring the semantic relatedness of two pieces of text is challenging in that it requires effective similarity measurement between sentences, paragraphs, and documents. Traditional NLP approaches, e.g., developing hand-crafted features, suffer from sparsity because of language ambiguity and the limited amount of annotated data available. Previous neural network models [68, 124, 131, 137, 172, 180] with a typical choice of the Siamese architecture [19] can help alleviate such sparsity and made significant progress over traditional feature engineering approaches, despite the fact the Siamese architecture treats the two inputs separately.

We develop novel interaction-based neural network models that are significantly

different from previous work with the Siamese architecture. Our approach innovates in neural network architectures, and focuses on better encouraging interactions between the two inputs. Unlike several recent neural network models [137, 172], we only use pre-trained word embeddings, but do not use sparse features, unsupervised model pretraining, syntactic parsers, or external resources like WordNet.

Our contributions are:

1. Our multi-perspective convolutional neural network model (MPCNN) compares sentences with a multiplicity of sentence-level perspectives. We firstly model each sentence using multiple convolutional neural networks that extract features at different levels of granularity and multiple types of pooling. We then compare the sentence representations over local regions with multiple similarity metrics using structured similarity measurement layer with a focus on better interaction-based modeling.

2. We also develop an attention-based input interaction layer on top of our MPCNN model, with which we can convert two independent input sentences into an inter-related sentence pair.

We conduct thorough evaluations and obtain strong performance on several tasks, rivaling or exceeding the state of the art. We describe details of our models in following sections.

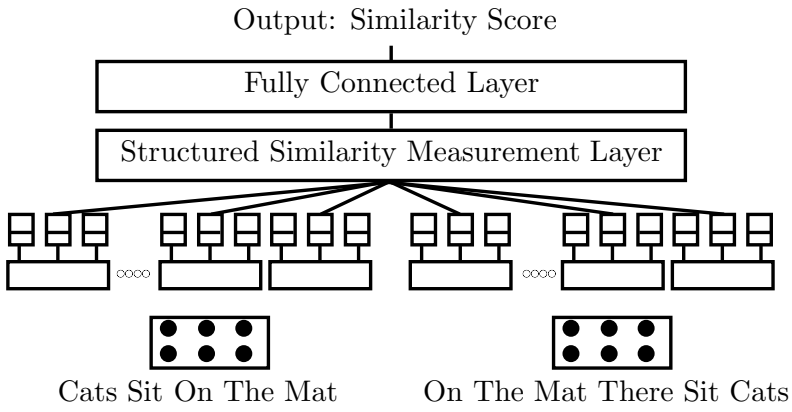


Figure 3.1: Overview of our first model. Two input sentences (on the bottom) are processed in parallel by identical neural networks, outputting sentence representations. The sentence representations are compared by the structured similarity measurement layer. The similarity features are then passed to a fully-connected layer for computing the similarity score (top).

3.1 Model Overview

Modeling textual similarity is complicated by the ambiguity and variability of linguistic expression. Ambiguity means that a single sentence can express multiple meanings; variability means that many different sentences could express the same meaning. Therefore a neural network model with single view and straightforward comparisons, such as the Siamese architecture, are not powerful enough to capture interactions between words and their polysemy. We designed interaction-based neural modeling, exploiting multiple types of input which are processed by multiple types of convolution and pooling, and multiple similarity functions for representation comparisons.

Our multi-perspective convolutional neural network model (shown in Figure 3.1) consists of two main components:

1. A **multi-perspective sentence model** for converting a sentence into a representation for similarity measurement; we use a convolutional neural network architecture with

multiple types of convolution and pooling in order to capture different granularities of information in the inputs on the sentence level.

2. A **structured similarity measurement layer** using multiple similarity measurements, which compare local regions of the sentence representations from the multi-perspective sentence model. The layer encourages interactions between convNet's output representations of the input sentences.

Our model consists of two subnetworks each processing a sentence in parallel. The subnetworks share all of their weights, and are joined by the structured similarity measurement layer for interaction modeling, then followed by a fully connected layer for final similarity score output.

The main difference from prior work lies in our use of multiple types of convolution, pooling, and structured similarity measurement over local regions. We describe our sentence model in Section 3.2 and our structured similarity measurement layer in Section 3.3. And in Section 3.4 we demonstrate highly competitive performance on two SemEval semantic relatedness tasks [2, 94], and on the Microsoft Research paraphrase (MSRP) identification task [36]. In addition, we perform ablation experiments to show the contribution of our modeling decisions for all datasets, demonstrating clear benefits from our use of multiple "perspectives" of the input sentences in interaction-based sentence modeling and structured similarity measurement.

3.2 Multi-Perspective Sentence Modeling

In this section we describe our convolutional neural network for modeling each sentence. We use two types of convolution filters defined on different perspectives of the input (Sec. 3.2.1), and also use multiple types of pooling (Sec. 3.2.2).

Our inputs are streams of tokens, which can be interpreted as a temporal sequence where nearby words are likely to be correlated. Let $sent \in \mathbf{R}^{len \times Dim}$ be a sequence of len input words represented by Dim-dimensional word embeddings, where $sent_i \in \mathbf{R}^{Dim}$ is the embedding of the i-th word in the sequence and $sent_{i:j}$ represents the concatenation of embeddings from word i up to and including word j. We denote the k-th dimension of the i-th word vector by $sent_i^{[k]}$ and we denote the vector containing the k-th dimension of words i to j by $sent_{i:j}^{[k]}$.

3.2.1 Convolution on Multiple Perspectives

We define a convolution filter F as a tuple $\langle ws, w_F, b_F, h_F \rangle$, where ws is the sliding window width, $w_F \in \mathbf{R}^{ws \times Dim}$ is the weight vector for the filter, $b_F \in \mathbf{R}$ is the bias, and h_F is the activation function (a nonlinear function such as $\tanh$). When filter F is applied to sequence $sent$, the inner product is computed between w_F and each possible window of word embeddings of length ws in $sent$, then the bias is added and the activation function is applied. This results in an output vector $out_F \in \mathbf{R}^{1+len-ws}$ where entry i equals

$$out_F[i] = h_F(w_F \cdot sent_{i:i+ws-1} + b_F) \tag{3.1}$$

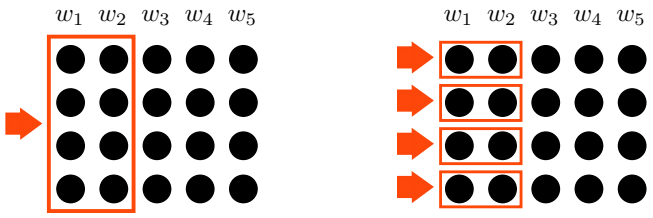


Figure 3.2: Left: a holistic filter matches entire word vectors (here, $ws = 2$). Right: per-dimension filters match against each dimension of the word embeddings independently.

where $i \in [1, 1 + len - ws]$. This filter can be viewed as performing "temporal" convolution, as it matches against regions of the word sequence. Since these filters consider the entirety of each word embedding at each position, we call them holistic filters; see the left half of Figure 3.2.

In addition, we target information at a finer granularity by constructing per-dimension filters $F^{[k]}$ for each dimension k of the word embeddings, where $w_{F^{[k]}} \in \mathbf{R}^{ws}$. See the right half of Figure 3.2. The per-dimension filters are similar to "spatial convolution" filters except that we limit each to a single, predefined dimension. We include separate per-dimension filters for each dimension of the input word embeddings.

Applying a per-dimension filter $F^{[k]} = \langle ws, w_{F^{[k]}}, b_{F^{[k]}}, h_{F^{[k]}} \rangle$ for dimension k results in an output vector $out_{F^{[k]}} \in \mathbf{R}^{1+len-ws}$ where entry i (for $i \in [1, 1 + len - ws]$) equals

$$out_{F^{[k]}}[i] = h_{F^{[k]}}(w_{F^{[k]}} \cdot sent^{[k]}_{i:i+ws-1} + b_{F^{[k]}})$$

Our use of word embeddings in both ways allows more information to be extracted for richer sentence modeling. While we typically do not expect individual dimensions of neural word embeddings to be interpretable to humans, there may still be distinct information captured by the different dimensions that our model could exploit. Furthermore,

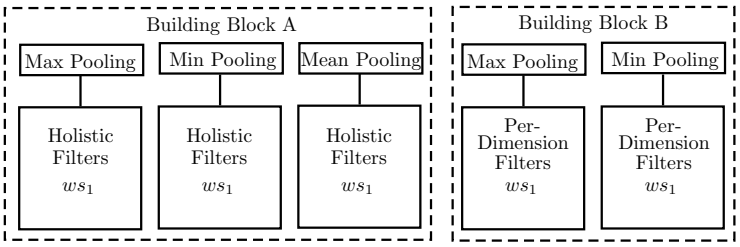


Figure 3.3: Each building block consists of multiple independent pooling layers and convolution layers with width ws_1. Left: $block_A$ operates on entire vectors of word embeddings. Right: $block_B$ operates on individual dimensions of word vectors to capture information of a finer granularity.

if we update the word embeddings during learning, different dimensions could be encouraged further to capture distinct information.

We define a convolution layer as a set of convolution filters that share the same type (holistic or per-dimension), activation function, and width ws. The type, width, activation function, and number of filters $numFilter$ in the layer are chosen by the modeler and the weights of each filter (w_F and b_F) are learned.

3.2.2 Multiple Pooling Types

The output vector out_F of a convolution filter F is typically converted to a scalar for subsequent use by the model using some method of pooling. For example, "max-pooling" applies a $\max$ operation across the entries of out_F and returns the maximum value. We experiment with two additional types of pooling: "min-pooling" and "mean-pooling".

A group, denoted $group(ws, pooling, sent)$, is an object that contains a convolution layer with width ws, uses pooling function $pooling$, and operates on sentence $sent$. We define a building block to be a set of groups. We use two types of building blocks, $block_A$

and $block_B$, as shown in Figure 3.3. We define $block_A$ as

$$\{group_A(ws_a, p, sent) : p \in \{\max, \min, \text{mean}\}\}.$$

That is, an instance of $block_A$ has three convolution layers, one corresponding to each of the three pooling functions; all have the same window size ws_a. An alternative choice would be to use the multiple types of pooling on the *same* filters [114]; we instead use independent sets of filters for the different pooling types.[1] We use blocks of type A for all holistic convolution layers.

We define $block_B$ as

$$\{group_B(ws_b, p, sent) : p \in \{\max, \min\}\}.$$

That is, $block_B$ contains two groups of convolution layers of width ws_b, one with max-pooling and one with min-pooling. Each $group_B(*)$ contains a convolution layer with Dim per-dimension convolution filters. That is, we use blocks of type B for convolution layers that operate on individual dimensions of word vectors.

We use these multiple types of pooling to extract different types of information from each type of filter. The design of each $group(*)$ allows a pooling function to interact with its own underlying convolution layers independently, so each convolution layer can learn to recognize distinct phenomena of the input for richer sentence modeling.

For a $group_A(ws_a, pooling_a, sent)$ with a convolution layer with $numFilter_A$ filters,

[1] We note that $\max$ and $\min$ are not both strictly necessary when using certain activation functions, but they still may help us find a more felicitous local optimum.

we define the output oG_A as a vector of length $numFilter_A$ where entry j is

$$oG_A[j] = pooling_a(out_{F_j}) \tag{3.2}$$

where filters are indexed as F_j. That is, the output of $group_A(*)$ is a $numFilter_A$-length vector containing the output of applying the pooling function on each filter's vector of filter match outputs.[2]

A component $group_B(*)$ of $block_B$ contains Dim filters, each operating on a particular dimension of the word embeddings. We define the output oG_B of $group_B(ws_b, pooling_b, sent)$ as a $Dim \times numFilter_B$ matrix where entry $[k][j]$ is

$$oG_B[k][j] = pooling_b(out_{F_j^{[k]}})$$

where filter $F_j^{[k]}$ is filter j for dimension k.

3.2.3 Multiple Window Sizes

Similar to traditional n-gram-based models, we use multiple window sizes ws in our building blocks in order to learn features of different lengths. We combine building blocks A and B in parallel with varying window sizes $ws = 1, 2, ..$ to form a wide parallel neural network architecture. For example, in Figure 3.4 we use four building blocks, each with one window size $ws = 1$ or 2 for its own convolution layers. In order to retain the original information in the sentences, we also include the entire matrix of word

[2]We note that there is no pooling across multiple filters in a layer/group, or across groups. Each pooling operation is performed independently on the matches of a single filter.

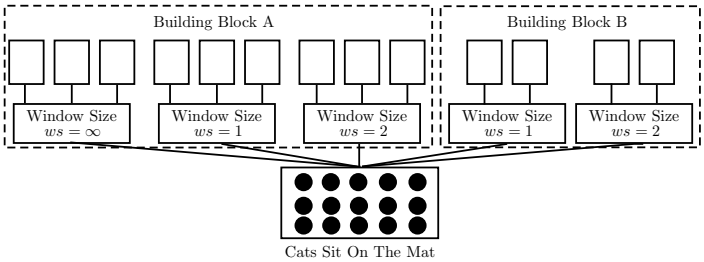


Figure 3.4: Example neural network architecture for a single sentence, containing 3 instances of $block_A$ (with 3 types of pooling) and 2 instances of $block_B$ (with 2 types) on varying window sizes $ws = 1, 2$ and $ws = \infty$; $block_A$ operates on entire word vectors while $block_B$ contains filters that operate on individual dimensions independently.

embeddings in the sentence, which essentially corresponds to $ws = \infty$.

The width ws represents how many words are matched by a filter, so using larger values of ws corresponds to matching longer n-grams in the input sentences, the value range of ws can be considered as how long context information we would like to model. The ranges of ws values and the numbers of filters $numFilter$ of $block_A$ and $block_B$ are empirical choices tuned based on validation data.

3.3 Structured Similarity Layer

In this section we describe the second part of our model, the structured similarity measurement layer.

Given two input sentences, the first part of our model computes sentence representations for each of them in parallel. One straightforward way to compare them is to flatten the sentence representations into two vectors, then use standard metrics like cosine similarity. However, this may not be optimal because different regions of the flattened sentence representations are from different underlying sources (e.g., groups of different widths, types of pooling, dimensions of word vectors, etc.). Vector flattening discard use-

ful compositional information [12] for computing similarity and loses the important information for interaction-based modeling. We therefore perform structured comparisons over particular regions of the sentence representations.

One important consideration is how to identify suitable local regions for comparison so that we can best utilize the compositional information in the sentence representations. There are many possible ways to group local comparison regions. In doing so, we consider the following four aspects: 1) whether from the same building block; 2) whether from convolutional layers with the same window size; 3) whether from the same pooling layer; 4) whether from the same filter of the underlying convolution layers.[3] We focus on comparing regions that share at least two of these conditions.

To concretize this, we provide two algorithms below to identify meaningful local regions. While there exist other sets of comparable regions that share the above conditions, we do not explore them all due to concerns about learning efficiency; we find that the subset we consider performs strongly in practice.

[3]We note that since we apply the same network to both sentences, the same filters are used to match both sentences, so we can directly compare filter matches of individual filters across the two sentences.

3.3.1 Similarity Comparison Units

We define two comparison units for comparing two local regions in the sentence representations:

$$comU_1(\overrightarrow{x}, \overrightarrow{y}) = \{\cos(\overrightarrow{x}, \overrightarrow{y}), L_2Euclid(\overrightarrow{x}, \overrightarrow{y}), |\overrightarrow{x} - \overrightarrow{y}|\} \tag{3.3}$$

$$comU_2(\overrightarrow{x}, \overrightarrow{y}) = \{\cos(\overrightarrow{x}, \overrightarrow{y}), L_2Euclid(\overrightarrow{x}, \overrightarrow{y})\} \tag{3.4}$$

Cosine distance ($\cos$) measures the distance of two vectors according to the angle between them, while L_2 Euclidean distance ($L_2Euclid$) and element-wise absolute difference measure magnitude differences.

We think the groupings contain auxiliary compositionality information and are useful for comparison. For example, one output sentence matrix is shown in Figure 3.5, if $numFilter$ is set to 3 filters per convolution layer and two window sizes of building block ws is set to 1 and 2. This sentence matrix has three equal-length column groups, each comes from max/min/mean pooling layers. Within each column group there are two vectors from two building blocks. And different shapes in the figure correspond to output of different filters. In the end for each input sentence, our neural network model outputs a sentence matrix, with rows and columns each representing different but useful information for comparison purpose. It will be quite a waste and losing grouping information of pooling layers and filters if we flatten the whole matrix into a single vector. Within each basic building block, we apply each pooling layer columnwise to an independent

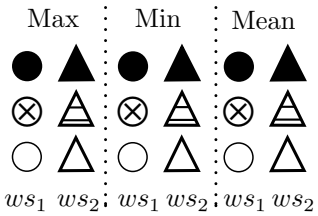


Figure 3.5: One simple sentence matrix from whole neural network, with two window sizes ws_1 and ws_2 and three filters per each convolution layer there are six filters, each in different shapes.

convolution layer. Our model generates sentence matrix. Of which each column are those signatures of an output distribution after applying same filter (hence each with same filter weights w_{F_j}) to the whole sequence of convoluted word sets, and each row are signatures of output distributions given same pooling functions across all filters. Rows and columns represent different meanings, this lays the foundation for the structured similarity comparison in the second part of our model.

3.3.2 Comparison over Local Regions

Algorithms 1 and 2 show how the two sentence representations are compared in our model. Algorithm 1 works on the output of $block_A$ only, while Algorithm 2 deals with both $block_A$ and $block_B$, focusing on regions from the output of the same pooling type and same block type, but with different filters and window sizes of convolution layers.

Given two sentences S_1 and S_2, we set the maximum window size ws of $block_A$ and $block_B$ to be n, let $regM_*$ represent a $numFilter_A$ by $n+1$ matrix, and assume that each $group_*$ outputs its corresponding oG_*. The output features are accumulated in a final vector fea.

Note that we assume the sentence representations for the two sentences have al-

Algorithm 1 Horizontal Comparison

for each pooling $p = \max, \min, \text{mean}$ **do**
  for each width $ws_1 = 1...n, \infty$ **do**
    $regM_1[*][ws_1] = group_A(ws_1, p, S_1)$
    $regM_2[*][ws_1] = group_A(ws_1, p, S_2)$
  end for
  for each $i = 1...numFilter_A$ **do**
    $fea_h = comU_2(regM_1[i], regM_2[i])$
    accumulate fea_h for final layer
  end for
end for

ready been generated from the first part of our model, therefore in Algorithm 1 and Algorithm 2, calling $group_{AB}(*)$ retrieves existing representations only, instead of re-running this component again from scratch.

Algorithm 2 Vertical Comparison

for each pooling $p = \max, \min, \text{mean}$ **do**
  for each width $ws_1 = 1...n, \infty$ **do**
    $oG_{1A} = group_A(ws_1, p, S_1)$
    for each width $ws_2 = 1...n, \infty$ **do**
      $oG_{2A} = group_A(ws_2, p, S_2)$
      $fea_a = comU_1(oG_{1A}, oG_{2A})$
      accumulate fea_a for final layer
    end for
  end for
  for each width $ws_1 = 1...n$ **do**
    $oG_{1B} = group_B(ws_1, p, S_1)$
    $oG_{2B} = group_B(ws_1, p, S_2)$
    for each $i = 1...numFilter_B$ **do**
      $fea_b = comU_1(oG_{1B}[*][i], oG_{2B}[*][i])$
      accumulate fea_b for final layer
    end for
  end for
end for

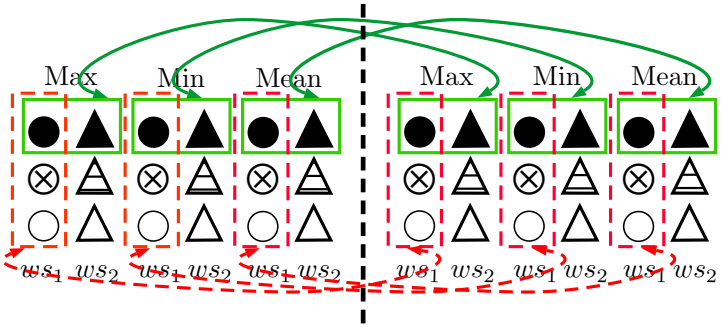


Figure 3.6: Simplified example of local region comparisons over two sentence representations that use $block_A$ only. The "horizontal comparison" (Algorithm 1) is shown with green solid lines and "vertical comparison" (Algorithm 2) with red dotted lines. Each sentence representation uses window sizes ws_1 and ws_2 with max/min/mean pooling and $numFilter_A = 3$ filters.

3.3.3 One Simplified Example

We provide a simplified working example to show how the two algorithms compare outputs of $block_A$ only. If we arrange the sentence representations into the shape of sentence matrices as in Figure 3.6, then in Algorithms 1 and 2 we are essentially comparing local regions of the two matrices in two directions: along rows and columns.

In Figure 3.6, each column of the max/min/mean groups is compared with all columns of the same pooling group for the other sentence. This is shown in red dotted lines in the Figure and listed in lines 2 to 9 in Algorithm 2. Note that both ws_1 and ws_2 columns within each pooling group should be compared using red dotted lines, but we omit this from the figure for clarity.

In the horizontal direction, each equal-sized max/min/mean group is extracted as a vector and is compared to the corresponding one for the other sentence. This process is repeated for all rows and comparisons are shown in green solid lines, as performed by Algorithm 1.

3.3.4 Other Model Details

We now describe other minor details of our model that are shown in Figure 3.1.

Output Fully-Connected Layer. On top of the similarity measurement layer (which outputs a vector containing all fea_*), we stack two linear layers with an activation layer in between, followed by a log-softmax layer as the final output layer, which outputs the similarity score.

Activation Layers. We used element-wise $\tanh$ as the activation function for all convolution filters and for the activation layer placed between the final two layers.

3.4 Experimental Setup

Everything necessary to replicate our experimental results can be found in our open-source code repository.[4]

We consider several sentence pair similarity tasks, Microsoft Research Paraphrase Corpus (MSRP), Sentences Involving Compositional Knowledge (SICK) dataset, Microsoft Video Paraphrase Corpus (MSRVID) and Trec 2011-2012 Microblogs. All datasets are introduced in Chapter 2 of background and related work.

3.4.1 Training

We use a hinge loss for the MSRP paraphrase identification task. This is simpler than log loss since it only penalizes misclassified cases. The training objective is to mini-

[4] http://hohocode.github.io/textSimilarityConvNet/

mize the following loss (summed over examples $\langle x, y_{gold} \rangle$):

$$loss(\theta, x, y_{gold}) = \sum_{y' \neq y_{gold}} \max(0, 1 + f_\theta(x, y') - f_\theta(x, y_{gold})) \tag{3.5}$$

where y_{gold} is the ground truth label, input x is the pair of sentences $x = \{S_1, S_2\}$, θ is the model weight vector to be trained, and the function $f_\theta(x, y)$ is the output of our model.

We use regularized KL-divergence loss for the semantic relatedness tasks (SICK and MSRVID), since the goal is to predict the similarity of the two sentences. The training objective is to minimize the KL-divergence loss plus an L_2 regularizer:

$$loss(\theta) = \frac{1}{m} \sum_{k=1}^{m} KL\left(f^k \;||\; \widehat{f}_\theta^k\right) + \frac{\lambda}{2}||\theta||_2^2 \tag{3.6}$$

where $\widehat{f}_\theta$ is the predicted distribution with model weight vector θ, f is the ground truth, m is the number of training examples, and λ is the regularization parameter. Note that we use the same KL-loss function and same sparse target distribution technique as [137].

3.4.2 Settings

We conduct experiments with ws values in the range $[1, 3]$ as well as $ws = \infty$ (no convolution).

We use multiple kinds of embeddings to represent each sentence, both on words and part-of-speech (POS) tags. We use the $Dim_g = 300$-dimensional GloVe word embeddings [101] trained on 840 billion tokens. We use $Dim_k = 25$-dimensional PARAGRAM

vectors [162] only on the MSRP task since they were developed for paraphrase tasks, having been trained on word pairs from the Paraphrase Database [47]. For POS embeddings, we run the Stanford POS tagger [93] on the English side of the Xinhua machine translation parallel corpus, which consists of Xinhua news articles with approximately 25 million words. We then train $Dim_p = 200$-dimensional POS embeddings using the `word2vec` toolkit [96]. Adding POS embeddings is expected to retain syntactic information which is reported to be effective for paraphrase identification [34]. We use POS embeddings only for the MSRP task.

Therefore for MSRP, we concatenate all word and POS embeddings and obtain $Dim = Dim_g + Dim_p + Dim_k = 525$-dimension vectors for each input word; for SICK and MSRVID we only use $Dim = 300$-dimension GloVe embeddings.

We use 5-fold cross validation on the MSRP *training* data for tuning, then largely reuse the same hyperparameters for the other two datasets. However, there are two changes: 1) for the MSRP task we update word embeddings during training but not so on SICK and MSRVID tasks; 2) we set the fully connected layer to contain 250 hidden units for MSRP, and 150 for SICK and MSRVID. These changes were done to speed up our experimental cycle on SICK and MSRVID; on SICK data they are the same experimental settings as used by [137], which makes for a cleaner empirical comparison.

We set the number of holistic filters in $block_A$ to be the same as the input word embeddings, therefore $numFilter_A = 525$ for MSRP and $numFilter_A = 300$ for SICK and MSRVID. We set the number of per-dimension filters in $block_B$ to be $numFilter_B = 20$ per dimension for all three datasets, which corresponds to $20 * Dim$ filters in total.

We perform optimization using stochastic gradient descent [17]. The backpropaga-

Model	Accuracy	F1
Hu et al. [68] ARC-I	69.6%	80.3%
Hu et al. [68] ARC-II	69.9%	80.9%
Qiu et al. [108]	72.0%	81.6%
Ul-qayyum and Altaf [146]	74.7%	81.8%
Blacoe and Lapata [14]	73.0%	82.3%
Fern and Stevenson [41]	74.1%	82.4%
Finch [43]	75.0%	82.7%
Das and Smith [34]	76.1%	82.7%
Wan et al. [150]	75.6%	83.0%
Socher et al. [131]	76.8%	83.6%
Madnani et al. [91]	77.4%	84.1%
Ji and Eisenstein [71]	**80.41**%	**85.96**%
Yin and Schütze [172](without pretraining)	72.5%	81.4%
Yin and Schütze [172](with pretraining)	78.1%	84.4%
Yin and Schütze [172](pretraining+sparse features)	78.4%	84.6%
This Work	78.60%	84.73%

Table 3.1: Test set results on MSRP for paraphrase identification. Rows in grey are neural network-based approaches.

tion algorithm is used to compute gradients for all parameters during training [49]. We fix the learning rate to 0.01 and regularization parameter $\lambda = 10^{-4}$.

We used different settings for the Trec 2011/2012 Microblogs test collection experiments [112]. We took advantage of existing open-source implementations; DSSM is our own re-implementation. The vocabulary size of our dataset is 90.3K, with around 37% words not found in the GloVe word embeddings. During training, we used stochastic gradient descent together with RMSPROP to iteratively update the model. The learning rate was initially set to 0.001, and then decreased by a factor of three when the development set loss stopped decreasing for three epochs. The maximum number of training epochs was 25.

Model	r	ρ	**MSE**
Socher et al. [132] DT-RNN	0.7863	0.7305	0.3983
Socher et al. [132] SDT-RNN	0.7886	0.7280	0.3859
Lai and Hockenmaier [79]	0.7993	0.7538	0.3692
Jimenez et al. [72]	0.8070	0.7489	0.3550
Bjerva et al. [13]	0.8268	0.7721	0.3224
Zhao et al. [179]	0.8414	-	-
LSTM	0.8477	0.7921	0.2949
Bi-LSTM	0.8522	0.7952	0.2850
2-layer LSTM	0.8411	0.7849	0.2980
2-layer Bidirectional LSTM	0.8488	0.7926	0.2893
Tai et al. [137] Const. LSTM	0.8491	0.7873	0.2852
Tai et al. [137] Dep. LSTM	0.8676	**0.8083**	**0.2532**
This work	**0.8686**	0.8047	0.2606

Table 3.2: Test set results on SICK, grouped as: (1) RNN variants; (2) SemEval 2014 systems; (3) sequential LSTM variants; (4) dependency and constituency tree LSTMs. Evaluation metrics are Pearson's r, Spearman's ρ, and mean squared error (MSE).

3.5 Results

Results on MSRP Data. We report F1 scores and accuracies from prior work in Table 3.1. Approaches shown in gray rows of the table are based on neural networks. The recent approach by [172] includes a pretraining technique which significantly improves results, as shown in the table. We do not use any pretraining but still slightly outperform their best results which use both pretraining and additional sparse features from [91].

When comparing to their model without pretraining, we outperform them by 6% absolute in accuracy and 3% in F1. Our model is also superior to other recent neural network models [68, 131] without requiring sparse features or unlabeled data as in [131, 172]. The best result on MSRP is from [71] which uses unsupervised learning on the MSRP test set and rich sparse features.

Model	Pearson's r
Rios et al. [117]	0.7060
Wang and Cer [155]	0.8037
Beltagy et al. [11]	0.8300
Bär et al. [10]	0.8730
Šarić et al. [120]	0.8803
This work	**0.9090**

Table 3.3: Test set results on MSRVID data. The Bar et al. (2012) and Saric et al. (2012) results were the top two submissions in the Semantic Textual Similarity task at the SemEval-2012 competition.

Results on SICK Data. Our results on the SICK task are summarized in Table 3.2, showing Pearson's r, Spearman's ρ, and mean squared error (MSE). We include results from the literature as reported by [137], including prior work using recurrent neural networks (RNNs), the best submissions in the SemEval-2014 competition, and variants of LSTMs. When measured by Pearson's r, the previous state-of-the-art approach uses a tree-structured LSTM [137]; note that their best results require a dependency parser.

On the contrary, our approach does not rely on parse trees, nor do we use POS / PARAGRAM embeddings for this task. The word embeddings, sparse distribution targets, and KL loss function are exactly the same as used by [137], therefore representing comparable conditions.

Results on MSRVID Data. Our results on the MSRVID data are summarized in Table 3.3, which includes the top 2 submissions in the Semantic Textual Similarity (STS) task from SemEval-2012. We find that we outperform the top system from the task by nearly 3 points in Pearson's r.

Results on Trec2011/2012 Microblogs Data. To rule out the effects of different preprocessing strategies during collection preparation (i.e., stemming, stopword removal,

etc.), we used the open-source implementations of tweet search provided by the TREC Microblog API[5] to retrieve up to 1000 tweets per topic using query likelihood (QL) for scoring. These initial results were then reranked using our proposed models. For effectiveness, we measured average precision (AP) and precision at 15, 30, and 100 (P15, P30, and P100); note that P30 was the official metric used in the TREC Microblog Tracks. Since all the models required training, we used the TREC 2011 topics for training and the TREC 2012 topics for evaluation. Additionally, we randomly selected 5% of query-document pairs from the training set as the development set; those selected samples were removed from the training set.

We considered several lexical and temporal baselines to evaluate our models. The standard query likelihood (QL) approach [105] was used as the lexical baseline. We used the kernel density estimation techniques of Efron et al. [39] as our temporal baseline (with the implementation from Rao et al. [109]). They proposed four different weighting schemes to estimate feedback parameters: uniform, score-based, rank-based, and oracle. The first three take advantage of document timestamp distributions from an initial retrieval, while the oracle method requires actual human relevance judgments. The oracle is useful to illustrate upper bound effectiveness for KDE-based techniques.

Table 3.4 shows our experimental results, with each row representing an experimental condition (numbered for convenience). For each method, we performed significance testing against the lexical baseline (QL) and the best-performing temporal KDE model (rank-based). In addition, we tested the significance of differences. In all cases, we used Fisher's two-sided, paired randomization test [130]. Superscripts indicate the row indexes

[5] https://github.com/lintool/twitter-tools

ID	Method		P15	P30	P100	AP
1	Query Likelihood (QL) [105]		0.381	0.329	0.234	0.200
Temporal Baselines						
2	KDE [39]	uniform	0.366	0.326	0.243	0.203
3		score-based	0.383	0.334	0.244	0.203
4		rank-based	0.387	0.337	0.244	0.204
5		oracle	**0.411**^{1,4}	**0.389**^{1,4}	**0.260**^{1,4}	**0.229**^{1,4}
Neural Ranking Approaches						
6	DSSM [69]		0.187	0.168	0.153	0.102
7	SM [124]		0.203	0.188	0.170	0.116
8	**Multi-Perspective ConvNet** [59]		**0.401**^{1}	**0.356**^{1}	**0.252**^{1}	**0.197**

Table 3.4: Results from the TREC 2011/2012 Microblog Track test collections, using TREC 2011 data for training and TREC 2012 data for evaluation. Superscripts indicate the row indexes from which the metric difference is statistically significant ($p < 0.05$) using Fisher's two-sided, paired randomization test.

for which the metric difference is statistically significant ($p < 0.05$).

From the block in Table 3.4 labeled "Temporal Baselines", we see that the KDE approaches (with the exception of the oracle condition) yield limited improvements over the QL baseline.[6] Looking at the block of Table 3.4 labeled "Neural Ranking Approaches", we find that the SM model and DSSM do not appear to be as effective as our multi-perspective CNN; in particular, the first two models actually perform worse than the simple QL baseline.

3.6 Model Ablation Study

We report the results of an ablation study in Table 3.5. We identify nine major components of our approach, remove one at a time (if applicable), and perform re-training and re-testing for all three tasks. We use the same experimental settings in Sec. 3.4.2 and

[6]These results are consistent with those of Rao et al. [109]; although those experiments affirmed the overall effectiveness of the KDE techniques, results from individual configurations (such as a particular train/test split) may not yield significant improvements.

Gp	ID	Ablation Component	MSRP Accuracy Diff.	MSRVID Pearson Diff.	SICK Pearson Diff.
1	1	Remove POS embeddings (Sec. 3.4.2)	-0.81	NA	NA
	2	Remove PARAGRAM embeddings (Sec. 3.4.2)	-1.33	NA	NA
2	3	Remove per-dimension embeddings, building block A only (Sec. 3.2.1)	-0.75	-0.0067	-0.0014
	4	Remove min and mean pooling, use max pooling only (Sec. 3.2.2)	-0.58	-0.0112	+0.0001
	5	Remove multiple widths, $ws = 1$ and $ws = \infty$ only (Sec. 3.2.3)	-2.14	-0.0048	-0.0012
3	6	Remove cosine and $L_2 Euclid$ distance in $comU_*$ (Sec. 3.3.1)	-2.31	-0.0188	-0.0309
4	7	Remove Horizontal Algorithm (Sec. 3.3.2)	-0.92	-0.0097	-0.0117
	8	Remove Vertical Algorithm (Sec. 3.3.2)	-2.15	-0.0063	-0.0027
	9	Remove similarity layer (completely flatten) (Sec. 3.3)	-1.90	-0.0121	-0.0288

Table 3.5: Ablation study over test sets of all three datasets. Nine components are divided into four groups. We remove components one at a time and show differences.

report differences (in accuracy for MSRP, Pearson's r for SICK/MSRVID) compared to our results in Tables 3.1–3.3.

The nine components can be divided into four groups: (1) input embeddings (components 1–2); (2) sentence modeling (components 3–5); (3) similarity measurement metrics (component 6); (4) similarity measurement layer (components 7–9). For MSRP, we use all nine components. For SICK and MSRVID, we use components 3–9 (as described in Sec. 3.4.2).

From Table 3.5 we find drops in performance for all components, with the largest differences appearing when removing components of the similarity measurement layer. For example, conducting comparisons over flattened sentence representations (removing component 9) leads to large drops across tasks, because this ignores structured information within sentence representations. Groups (1) and (2) are also useful, particularly for the MSRP task, demonstrating the extra benefit obtained from our multi-perspective approach in sentence modeling.

We see consistent drops when ablating the Vertical/Horizontal algorithms that target particular regions for comparison. Also, removing group (3) hinders both the Horizontal and Vertical algorithms (as described in Section 3.3.1), so its removal similarly causes large drops in performance. Though convolutional neural networks already perform strongly when followed by flattened vector comparison, we are able to leverage the full richness of the sentence models by performing structured similarity modeling on their outputs.

3.7 Attention-based Input Interaction Layer

Our multi-perspective convolutional neural network model (MPCNN) utilizes structured similarity measurement layer to encourage interaction-based modeling, and achieve competitive performance. Inspired by this, we also utilize the attention mechanism [7] on the input level and develop an attention-based interaction layer [61] that converts the two independent input sentences into an inter-related sentence pair.

We incorporate our interaction-based input layer into the MPCNN model as the first layer of our model. It is applied over raw word embeddings of input sentences to generate attention-based re-weighted word embeddings. The attention-based re-weightings can guide the *focus* of the MPCNN model onto important input words. That is, words in one sentence that are more relevant to the other sentence receive higher weights.

Our attention layer uses a straightforward attention function that is different from the work [7, 174], and still leads to effectiveness improvement. Also note that the attention mechanism is only one of the many techniques to encourage the interactions between inputs. We also notice there are related attention mechanism work in neural machine translation [7], their attention implementation is complicated in comparison to ours, and also utilized in different areas than the textual similarity domain.

We first define input sentence representation $S^i \in \mathbb{R}^{\ell_i \times d}$ ($i \in \{0, 1\}$) to be a sequence of ℓ_i words, each with a d-dimensional word embedding vector. $S^i[a]$ denotes the embedding vector of the a-th word in S^i. We then define an *attention matrix* $D \in \mathbb{R}^{\ell_0 \times \ell_1}$. Entry (a, b) in the matrix D represents the pairwise word similarity score between the a-th word embedding of S^0 and the b-th word embedding of S^1. The similarity score uses

cosine distance:

$$D[a][b] = cosine(S^0[a], S^1[b])$$

Given the attention matrix D, we generate the attention weight vector $A^i \in \mathbb{R}^{\ell_i}$ for input sentence S^i ($i \in \{0, 1\}$). Each entry $A^i[a]$ of the attention weight vector can be viewed as an attention-based relevance score of one word embedding $S^i[a]$ with respect to all word embeddings of the other sentence $S^{1-i}[:]$. Attention weights $A^i[:]$ sum to one due to the $softmax$ normalization:

$$E^0[a] = \sum_b D[a][b]$$

$$E^1[b] = \sum_a D[a][b]$$

$$A^i = softmax(E^i)$$

We finally define updated embeddings $attenEmb \in \mathbb{R}^{2d}$ for each word as a concatenation of the original and attention-reweighted word embeddings:

$$attenEmb^i[a] = concat(S^i[a], A^i[a] \odot S^i[a])$$

where $\odot$ represents element-wise multiplication.

Our input interaction layer is inspired by recent work that incorporates attention mechanisms into neural networks [7]. Many of these add parameters and computational complexity to the model. However, our attention-based input interaction layer is simpler and more efficient. Moreover, we do not introduce any additional parameters, as we sim-

Ablation study on STS2015	Pearson
Full System (Multi-Perspective ConvNet [59]+Attention-based Input Layer)	**0.8040**
- Remove the attention layer (Multi-Perspective ConvNet only)	0.7948
- Replace PARAGRAM-PHRASE with GloVe (Sec. 4)	0.7622
- Replace PARAGRAM-PHRASE with PARAGRAM-SL999	0.7721
Winning System of STS2015	0.8015

Table 3.6: Ablation study on STS2015 test data.

ply use cosine distance to create the attention weights. Nevertheless, adding this attention layer improves performance, as we show in Section 3.7.1.

3.7.1 Experimental Setup

Experimental Settings. We largely follow the same experimental settings as done in Section 3.4.2, e.g., we perform optimization with stochastic gradient descent using a fixed learning rate of 0.01. We use $d = 300$-dim PARAGRAM-PHRASE XXL word embeddings from Wieting et al. [163], and for ablation study we also use the PARAGRAM-SL999 embeddings from Wieting et al. [161].

Datasets. We use three data sets for evaluation. They are STS2015, WikiQA and TrecQA. Please refer to Chapter 2 of background and related work for further details.

3.7.2 Results

Results and Ablation Study on STS2015. Table 3.7 shows the results on the STS2015 test sets. In Table 3.6 we also show an ablation study. We remove or replace one component at a time from the full system and perform re-training and re-testing.

We observe a significant drop when the attention-based input interaction layer is removed. Our full system performs favorably compared to the winning system by Sultan

STS2015	Domain	3rd	2nd	1st	MP ConvNet+Attention
answers-forums	forum answers	0.7241	0.6946	**0.7390**	0.6664
answers-students	student short answers	0.7569	0.7784	0.7657	**0.7900**
belief	belief annotations	0.7223	0.7482	0.7491	**0.7749**
headlines	news headlines	0.8250	0.8245	0.8250	**0.8267**
images	image descriptions	0.8631	0.8527	0.8644	**0.8786**
Weighted Mean		0.7921	0.7942	0.8015	**0.8040**

Table 3.7: Test results on all five sets in STS2015. We show results of the top three participating systems at the competition in Pearson's r scores with our multi-perspective convnet and attention layer.

Model	Dataset	MAP	MRR
Multi-Perspective ConvNet [59]	TrecQA	0.756	0.830
Multi-Perspective ConvNet [59]+Attention-based Input Interaction Layer	TrecQA	**0.766**	**0.840**
Multi-Perspective ConvNet [59]	WikiQA	0.693	0.709
Multi-Perspective ConvNet [59]+Attention-based Input Interaction Layer	WikiQA	**0.701**	**0.715**

Table 3.8: Results and Ablation Study on TrecQA and WikiQA.

et al. [134] on the STS2015 SemEval competition.

Results on TrecQA and WikiQA. We add the attention-based input interaction layer to our multi-perspective convolutional neural networks, and evaluate the performance on both WikiQA and TrecQA answer selection tasks in Figure 3.8. We observe good performance improvement on both tasks, which demonstrates the attention mechanism's effectiveness.

3.8 Summary

This chapter introduces our novel structured similarity layer, the attention-based input interaction layer, and the multi-perpective sentence-level modeling of our MPCNN model for the interaction-based neural modeling, which is the major difference between our models and the Siamese architecture. In comparison to previous work, our ablation study and results suggest that such interaction-based modeling helps achieve highly com-

petitive performance for several textual similarity measurement tasks we evaluated.

Our model architecture, with its many paths of information flow, is admittedly complex. Though we have removed hand engineering of features, we have added a substantial amount of functional architecture engineering. This may be necessary when using the small training sets provided for the tasks we consider here. We conjecture that a simpler, deeper neural network architecture may outperform our model when given large amounts of training data.

It must be noted that this work was jointly done with our external collaborator Prof. Kevin Gimpel in Toyota Technological Institute at Chicago (TTIC). I thank Kevin for all his contributions, all discussions and great writing improvement.

In the next chapter (Chapter 4) we introduce our pairwise word interaction models with very deep neural networks, which instead focuses on modeling the interactions between word pairs across two sentences.

Chapter 4: Pairwise Word Interaction Modeling with Very Deep Neural Networks

Our multi-perspective convolutional neural networks (MPCNN) as introduced in Chapter 3 encourages interactions between aggregated sentence representations of two inputs. In this chapter, we focus on capturing more fine-grained word-level information, and show our pairwise word interaction modeling (PWI) with very deep neural networks [56, 80].

With this model, our contribution is twofold:

1. First, instead of using sentence modeling, we propose *pairwise word interaction modeling* that encourages explicit word context interactions across sentences. This is inspired by our own intuitions of how people recognize textual similarity: given two sentences $sent_1$ and $sent_2$, a careful reader might look for corresponding semantic units, which we operationalize in our pairwise word interaction modeling technique (Sec. 4.3).

2. Second, based on the pairwise word interactions, we describe a novel *similarity focus layer* which helps the model selectively identify important word interactions depending on their importance for similarity measurement. Since not all words are created equal, important words that can make more contributions deserve extra

"focus" from the model (Sec. 4.4).

We conducted thorough evaluations on over ten test sets from SemEval STS competitions [2, 3, 94], two answer selection tasks (WikiQA [168] and TrecQA [157]), TwitterPPDB [80] and SemEval2017 cross-lingual datasets [21]. We demonstrate highly competitive or state-of-the-art accuracy on all tasks. In addition, we conducted ablation studies and visualized our models to show the clear benefits of modeling pairwise word interactions for textual similarity measurement.

4.1 Model Overview

Figure 4.1 shows our end-to-end model with four major components:

1. Bidirectional Long Short-Term Memory Networks (Bi-LSTMs) [52, 53] are used for context modeling of input sentences, which serves as the basis for all following components (Sec. 4.2).

2. A novel pairwise word interaction modeling technique encourages direct comparisons between word contexts across sentences (Sec. 4.3).

3. A novel similarity focus layer helps the model identify important pairwise word interactions across sentences (Sec. 4.4).

4. A 19-layer very deep convolutional neural network (ConvNet) converts the similarity measurement problem into a pattern recognition problem for final classification (Sec. 4.5).

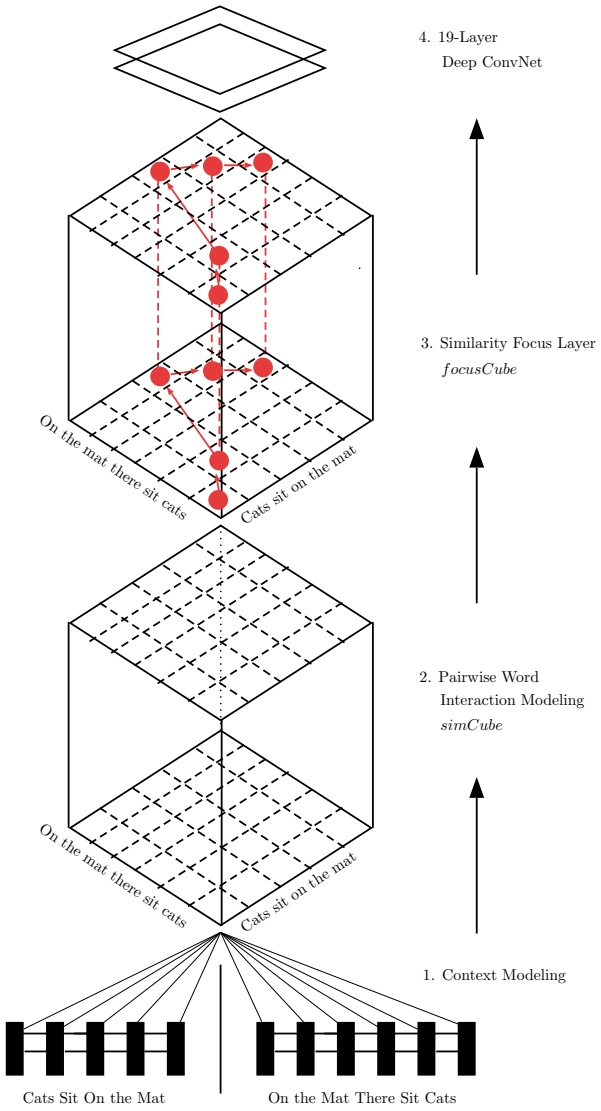


Figure 4.1: Our end-to-end neural network model, consisting of four major components.

To our best knowledge this is the first neural network model, a novel hybrid architecture combining Bi-LSTMs and a very deep ConvNet, that uses a similarity focus mechanism with selective location-based attentions to important pairwise word interactions for the STS problem. We describe details of each component in the following sections.

4.2 Context Modeling

Different words occurring in similar semantic contexts of respective sentences have a higher chance to contribute to the similarity measurement. We therefore need word context modeling, which serves as a basis for all following components of this work.

LSTM [67] is a special variant of Recurrent Neural Networks [164]. It can capture long-range dependencies and nonlinear dynamics between words, and has been successfully applied to many NLP tasks [42, 135]. LSTM has a memory cell that can store information over a long history, as well as three gates that control the flow of information into and out of the memory cell. At time step t, given an input x_t, previous output h_{t-1}, input gate i_t, output gate o_t and forget gate f_t, $LSTM(x_t, h_{t-1})$ outputs the hidden state h_t based on the equations below:

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i) \tag{4.1}$$

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f) \tag{4.2}$$

$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b^o) \tag{4.3}$$

$$u_t = tanh(W^u x_t + U^u h_{t-1} + b^u) \tag{4.4}$$

$$c_t = i_t \cdot u_t + f_t \cdot c_{t-1} \tag{4.5}$$

$$h_t = o_t \cdot tanh(c_t) \tag{4.6}$$

$$LSTM(x_t, h_{t-1}) = h_t \tag{4.7}$$

$$BiLSTMs(x_t, h_{t-1}) = \{LSTM^f, LSTM^b\} \tag{4.8}$$

where σ is the logistic sigmoid activation, W^*, U^* and b^* are learned weight matrices and biases. LSTMs are better than RNNs for context modeling, in that their memory cells and gating mechanisms handle the vanishing gradients problem in training.

We use bidirectional LSTMs (Bi-LSTMs) for context modeling in this work. Bi-LSTMs consist of two LSTMs that run in parallel in opposite directions: one (forward $LSTM^f$) on the input sequence and the other (backward $LSTM^b$) on the reverse of the sequence. At time step t, the Bi-LSTMs hidden state h_t^{bi} is a concatenation of the hidden state h_t^{for} of $LSTM^f$ and the hidden state h_t^{back} of $LSTM^b$, representing the neighbor contexts of input x_t in the sequence. We define the $unpack$ operation below:

$$h_t^{for}, h_t^{back} = unpack(h_t^{bi}) \tag{4.9}$$

Context modeling with Bi-LSTMs allows all the following components to be built over word contexts, rather than over individual words.

4.3 Pairwise Word Interaction Modeling

From our own intuitions, given two sentences in a STS task, a careful human reader might compare words and phrases across the sentences to establish semantic correspondences and from these infer similarity. Our pairwise word interaction model is inspired by such behavior: whenever the next word of a sentence is read, the model would compare it and its context against all words and their contexts in the other sentence. Figure 4.2 illustrates this model.

We first define a comparison unit for comparing two hidden states $\overrightarrow{h_1}, \overrightarrow{h_2}$ of Bi-

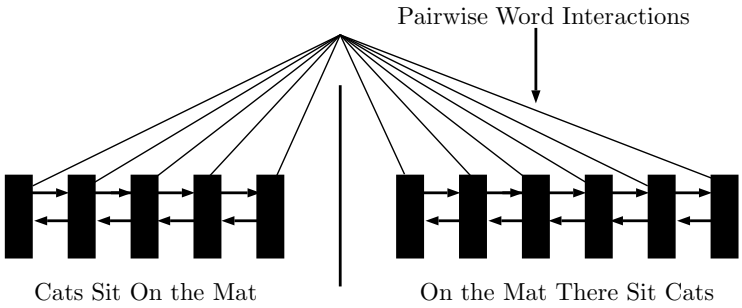


Figure 4.2: Pairwise word interaction modeling. Sentences are encoded by weight-shared Bi-LSTMs. We construct pairwise word interactions for context comparisons across sentences.

Algorithm 3 Pairwise Word Interaction Modeling

Initialize: $simCube \in R^{13 \cdot |sent_1| \cdot |sent_2|}$ to all 1
for each time step $t = 1...|sent_1|$ **do**
  for each time step $s = 1...|sent_2|$ **do**
    $h_{1t}^{for}, h_{1t}^{back} = unpack(h_{1t}^{bi})$
    $h_{2s}^{for}, h_{2s}^{back} = unpack(h_{2s}^{bi})$
    $h_{1t}^{add} = h_{1t}^{for} + h_{1t}^{back}$
    $h_{2s}^{add} = h_{2s}^{for} + h_{2s}^{back}$
    $simCube[1:3][t][s] = coU(h_{1t}^{bi}, h_{2s}^{bi})$
    $simCube[4:6][t][s] = coU(h_{1t}^{for}, h_{2s}^{for})$
    $simCube[7:9][t][s] = coU(h_{1t}^{back}, h_{2s}^{back})$
    $simCube[10:12][t][s] = coU(h_{1t}^{add}, h_{2s}^{add})$
  end for
end for
return $simCube$

LSTMs.

$$coU(\overrightarrow{h_1}, \overrightarrow{h_2}) = \{\cos(\overrightarrow{h_1}, \overrightarrow{h_2}), L_2Euclid(\overrightarrow{h_1}, \overrightarrow{h_2}), DotProduct(\overrightarrow{h_1}, \overrightarrow{h_2})\}$$

Cosine distance (cos) measures the distance of two vectors by the angle between them, while L_2 Euclidean distance ($L_2Euclid$) and dot-product distance ($DotProduct$) measure magnitude differences. We use three similarity functions for richer measurement.

Algorithm 3 provides details of the modeling process. Given the input $x_t^a \in sent_a$

at time step t where $a \in \{1, 2\}$, its Bi-LSTMs hidden state h_{at}^{bi} is the concatenation of the forward state h_{at}^{for} and the backward state h_{at}^{back}. Algorithm 3 proceeds as follows: it enumerates all word pairs (s, t) across both sentences, then perform comparisons using the coU unit four times over: 1) Bi-LSTMs hidden states h_{1t}^{bi} and h_{2s}^{bi}; 2) forward hidden states h_{1t}^{for} and h_{2s}^{for}; 3) backward hidden states h_{1t}^{back} and h_{2s}^{back}; and 4) the addition of forward and backward hidden states h_{1t}^{add} and h_{2s}^{add}. The output of Algorithm 3 is a similarity cube $simCube$ with size $R^{13 \cdot |sent_1| \cdot |sent_2|}$, where $|sent_*|$ is the number of words in the sentence $sent_*$. The 13 values collected from each word pair (s, t) are: the 12 similarity distances, plus one extra dimension for the padding indicator. Note that all word interactions are modeled over word contexts in Algorithm 3, rather than individual words.

Our pairwise word interaction model shares similarities with recent popular neural attention models [7, 118]. However, there are important differences: For example, we do not use attention weight vectors or weighted representations, which are the core of attention models. The other difference is that attention weights are typically interpreted as soft degrees with which the model attends to particular words; in contrast, our word interaction model directly utilizes multiple similarity metrics, and thus is more explicit.

4.4 Similarity Focus Layer

Since not all words are created equal, important pairwise word interactions between the sentences (Sec. 4.3) that can better contribute to the similarity measurement deserve more model focus. We therefore develop a similarity focus layer which can identify important word interactions and increase their model weights correspondingly. This sim-

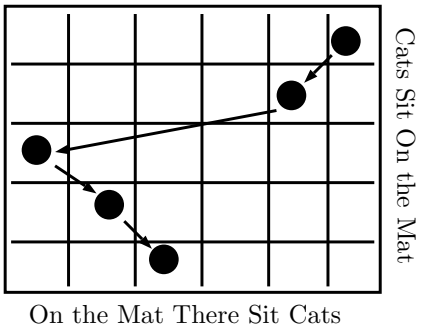


Figure 4.3: The similarity focus layer helps identify important pairwise word interactions (in black dots) depending on their importance for similarity measurement.

ilarity focus layer is directly incorporated into our end-to-end model and is placed on top of the pairwise word interaction model, as in Figure 4.1.

Figure 4.3 shows one example where each cell of the matrix represents a pairwise word interaction. The similarity focus layer introduces re-weightings to word interactions depending on their importance for similarity measurement. The ones tagged with black dots are considered important, and are given higher weights than those without.

Algorithm 4 shows the forward pass of the similarity focus layer. Its input is the similarity cube $simCube$ (Section 4.3). Algorithm 4 is designed to incorporate two different aspects of similarity based on $cosine$ (angular) and $L2$ (magnitude) similarity, thus it has two symmetric components: the first one is based on $cosine$ similarity (Line 5 to Line 13); and the second one is based on $L2$ similarity (Line 15 to Line 23). We also aim for the goal that similarity values of all found important word interactions should be maximized. To achieve this, we sort the similarity values in descending order (Line 5 for $cosine$, Line 15 for $L2$). Note channels 10 and 11 of the $simCube$ contain $cosine$ and $L2$ values, respectively; the padding indicator is in Line 24.

We start with the $cosine$ part first, then $L2$. For each, we check word interaction

Algorithm 4 Forward Pass: Similarity Focus Layer

1: Input: $simCube \in R^{13 \cdot |sent_1| \cdot |sent_2|}$
2: Initialize: $mask \in R^{13 \cdot |sent_1| \cdot |sent_2|}$ to all 0.1
3: Initialize: $s1tag \in R^{|sent_1|}$ to all zeros
4: Initialize: $s2tag \in R^{|sent_2|}$ to all zeros
5: $sortIndex_1 = sort(simCube[10])$
6: **for** each $id = 1...|sent_1| + |sent_2|$ **do**
7: $pos_{s1}, pos_{s2} = calcPos(id, sortIndex_1)$
8: **if** $s1tag[pos_{s1}] + s2tag[pos_{s2}] == 0$ **then**
9: $s1tag[pos_{s1}] = 1$
10: $s2tag[pos_{s2}] = 1$
11: $mask[:][pos_{s1}][pos_{s2}] = 1$
12: **end if**
13: **end for**
14: Re-Initialize: $s1tag, s2tag$ to all zeros
15: $sortIndex_2 = sort(simCube[11])$
16: **for** each $id = 1...|sent_1| + |sent_2|$ **do**
17: $pos_{s1}, pos_{s2} = calcPos(id, sortIndex_2)$
18: **if** $s1tag[pos_{s1}] + s2tag[pos_{s2}] == 0$ **then**
19: $s1tag[pos_{s1}] = 1$
20: $s2tag[pos_{s2}] = 1$
21: $mask[:][pos_{s1}][pos_{s2}] = 1$
22: **end if**
23: **end for**
24: $mask[13][:][:] = 1$
25: $focusCube = mask \cdot simCube$
26: **return** $focusCube \in R^{13 \cdot |sent_1| \cdot |sent_2|}$

candidates moving down the sorted list. Function $calcPos$ is used to calculate relative sentence positions pos_{s*} in the $simCube$ given one interaction pair. We follow the constraint that no word in both sentences should be tagged to be important more than once. We increase weights of important word interactions to 1 (in Line 11 based on $cosine$ and Line 21 based on $L2$), while unimportant word interactions receive weights of 0.1 (in Line 2).

We use a mask matrix, $mask$, to hold the weights of each. The final output of the similarity focus layer is a focus-weighted similarity cube $focusCube$, which is the

element-wise multiplication (Line 25) of the matrix $mask$ and the input $simCube$.

The similarity focus layer is based on the following intuition: given each word in one sentence, we look for its semantically similar twin in the other sentence; if found then this word is considered important, otherwise it contributes to a semantic difference. Though technically different, this process shares conceptual similarity with finding translation equivalences in statistical machine translation [5].

The backward pass of the similarity focus layer is straightforward: we reuse the $mask$ matrix as generated in the forward pass and apply the element-wise multiplication of $mask$ and inflow gradients, then propagate the resulting gradients backward.

4.5 Similarity Classification with Very Deep ConvNet

The $focusCube$ contains focus-weighted fine-grained similarity information. In the final model component we use the $focusCube$ to compute the final similarity score. If we treat the $focusCube$ as an "image" with 13 channels, then semantic similarity measurement can be converted into a pattern recognition (image processing) problem, where we are looking for patterns of strong pairwise word interactions in the "image". The stronger the overall pairwise word interactions are, the higher similarity the sentence pair will have.

Recent advances from successful systems at ImageNet competitions [127, 136] show that the depth of a neural network is a critical component for achieving competitive performance. We therefore use a deep homogeneous architecture which has repetitive convolution and pooling layers.

Very Deep ConvNet Configurations	
Input Size: 32 **by** 32	**Input Size:** 48 **by** 48
Spatial Conv 128: size 3×3, stride 1, pad 1	
ReLU	
Max Pooling: size 2×2, stride 2	
Spatial Conv 164: size 3×3, stride 1, pad 1	
ReLU	
Max Pooling: size 2×2, stride 2	
Spatial Conv 192: size 3×3, stride 1, pad 1	
ReLU	
Max Pooling: size 2×2, stride 2	
Spatial Conv 192: size 3×3, stride 1, pad 1	
ReLU	
Max Pooling: size 2×2, stride 2	
Spatial Conv 128: size 3×3, stride 1, pad 1	
ReLU	
Max Pooling: 2×2, s2	Max Pooling: 3×3, s1
Fully-Connected Layer	
ReLU	
Fully-Connected Layer	
LogSoftMax	

Table 4.1: Deep ConvNet architecture given two padding size configurations for final classification.

Our network architecture (Table 4.1) is composed of spatial max pooling layers, spatial convolutional layers (Conv) with a small filter size of 3×3 plus stride 1 and padding 1. We adopt this filter size because it is the smallest one to capture the space of left/right, up/down, and center; the padding and stride is used to preserve the spatial input resolution. We then use fully-connected layers followed by the final softmax layer for the output. After each spatial convolutional layer, a rectified linear units (ReLU) non-linearity layer [78] is used.

The input to this deep ConvNet is the $focusCube$, which does not always have the same size because the lengths of input sentences vary. To address this, we use zero

padding. For computational reasons we provide two configurations in Table 4.1, for length padding up to either 32×32 or 48×48. The only difference between the two configurations is the last pooling layer. If sentences are longer than the padding length limit we only use the number of words up to the limit. In our experiments we found the 48×48 padding limit to be acceptable since most sentences in our datasets are only $1 - 30$ words long.

4.6 Experimental Setup

Datasets. We conducted experiments on over ten different datasets: three recent SemEval competitions, two answer selection tasks, TwitterPPDB and one cross-lingual dataset. Note that the answer selection task, which is to rank candidate answer sentences based on their similarity to the questions, is essentially the textual similarity measurement problem.

Please refer to Section 2.1.1 for details of all datasets. Here we provide brief descriptions for each dataset.

1. Sentences Involving Compositional Knowledge (SICK) data has been used and introduced in our previous work [59].

2. Microsoft Video Paraphrase Corpus (MSRVID) data has been used and introduced in our previous work [59].

3. Task 10 of the 2014 SemEval competition on Semantic Textual Similarity (STS2014) [3] provided six different test sets from different domains. Each pair has a similarity score $\in [0, 5]$ which increases with similarity.

4. The open domain question-answering WikiQA [168] dataset.

5. The TrecQA dataset [157] from the Text Retrieval Conferences has been used and introduced in previous sections.

6. TwitterPPDB [80], the largest human-annotated paraphrase dataset (in 2017).

7. SemEval2017 [21] cross-lingual dataset provides 4 test sets.

Training. For experiments on SICK, MSRVID, and STS2014, the training objective is to minimize the KL-divergence loss, which has been used by our multi-perspective convolutional neural networks (MPCNN). We used the same hinge loss for all other datasets as used in our MPCNN model. Please refer to Section 3.4.1 for details on the loss functions.

In all cases, we performed optimization using RMSProp [140] with backpropagation [17], with a learning rate fixed to 10^{-4}.

Settings. For the SICK and MSRVID experiments, we used 300-dimension GloVe word embeddings [101]. For experiments on all other datasets, we used 300-dimension PARAGRAM-SL999 embeddings and the PARAGRAM-PHRASE embeddings, trained on word pairs from the Paraphrase Database (PPDB) [47]. We did not update word embeddings in all experiments.

We used the SICK development set for tuning. For the answer selection task (WikiQA and TrecQA), we used the official trec_eval scorer to compute the metrics Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR) and selected the best development model based on MRR for final testing. Our timing experiments were conducted on an Intel Xeon E5-2680 CPU.

Due to sentence length variations, for the SICK and MSRVID data we padded the sentences to 32 words; for all other datasets, we padded the sentences to 48 words.

Model	r	ρ	**MSE**
Mean word vectors	0.8046	0.7294	0.3595
Socher et al. [132] DTRNN	0.7863	0.7305	0.3983
Socher et al. [132] SDTRNN	0.7886	0.7280	0.3859
Lai and Hockenmaier [79]	0.7993	0.7538	0.3692
Jimenez et al. [72]	0.8070	0.7489	0.3550
Bjerva et al. [13]	0.8268	0.7721	0.3224
Zhao et al. [179]	0.8414	-	-
LSTM	0.8477	0.7921	0.2949
Bi-LSTM	0.8522	0.7952	0.2850
2-layer LSTM	0.8411	0.7849	0.2980
2-layer Bi-LSTM	0.8488	0.7926	0.2893
Tai et al. [137] Const. LSTM	0.8491	0.7873	0.2852
Tai et al. [137] Dep. LSTM	0.8676	0.8083	0.2532
Multi-Perspective ConvNet	**0.8686**	**0.8047**	**0.2606**
Pairwise Word Interaction Model	**0.8784**	**0.8199**	**0.2329**

Table 4.2: Test results on SICK grouped as: (1) Simple baseline; (2) RNN variants; (3) SemEval 2014 systems; (4) Sequential LSTM variants; (5) Dependency and constituency tree LSTMs. Evaluation metrics are Pearson's r, Spearman's ρ, and mean squared error (MSE).

4.7 Results

SICK Results (Table 4.2). Our model outperforms previous neural network models, most of which are based on sentence modeling. Our MPCNN model [59] and TreeLSTM work [137] achieve comparable accuracy; for example, their difference in Pearson's r is only 0.1%. In comparison, our model outperforms both by 1% in Pearson's r, over 1.1% in Spearman's ρ, and 2-3% in MSE. Note that we used the same word embeddings, sparse distribution targets, and loss function as in He et al. [59] and Tai et al. [137], thereby representing comparable experimental conditions.

MSRVID Results (Table 4.3). Our model outperforms our own MPCNN work of He et al. [59], which already reports a Pearson's r score of over 0.9,

Model	Pearson's r
Rios et al. [117]	0.7060
Wang and Cer [155]	0.8037
Beltagy et al. [11]	0.8300
Bär et al. [10]	0.8730
Šarić et al. [120]	0.8803
Multi-Perspective ConvNet	**0.9090**
Pairwise Word Interaction Model	**0.9112**

Table 4.3: Test results on MSRVID data.

STS2014	Domain	3rd	2nd	1st	PWI Model
deft-forum	discussion forum	0.5305	0.4711	0.4828	**0.5684**
deft-news	news article	**0.7813**	0.7628	0.7657	0.7079
headlines	new headlines	**0.7837**	0.7597	0.7646	0.7551
image	image description	**0.8343**	0.8013	0.8214	0.8221
OnWN	word sense definition	0.8502	0.8745	0.8589	**0.8847**
tweetnews	social media	0.6755	**0.7793**	0.7639	0.7469
Weighted Mean		0.7549	0.7605	0.7610	**0.7666**

Table 4.4: Test results on all six test sets in STS2014. We show results of the top three participating systems at the competition in Pearson's r scores and our pairwise word interaction model (PWI Model).

STS2014 Results (Table 4.4). Systems in the competition are ranked by the weighted mean (the official measure) of Pearson's r scores calculated based on the number of sentence pairs in each test set. We show the 1st ranked [133], 2nd [75], 3rd [90] systems in the STS2014 competition, all of which are based on heavy feature engineering. Our model does not use any sparse features, WordNet, or parse trees, but still performs favorably compared to the STS2014 winning system [133].

WikiQA Results (Table 4.5). We compared our model to competitive baselines prepared by Yang et al. [168] and also evaluated He et al. [59]'s multi-perspective ConvNet on the same data. The neural network models in the table, paragraph vector (PV) [81], CNN [175], and PV-Cnt/CNN-Cnt with word matching features [168], are mostly based

Model	MAP	MRR
Word Cnt [168]	0.4891	0.4924
Wgt Word Cnt [168]	0.5099	0.5132
PV [81]	0.5110	0.5160
PV-Cnt [168]	0.5976	0.6058
LCLR [171]	0.5993	0.6086
CNN [175]	0.6190	0.6281
CNN-Cnt [168]	0.6520	0.6652
Multi-Perspective ConvNet	**0.6930**	**0.7090**
Multi-Perspective ConvNet+AttentionInput	**0.7010**	**0.7150**
Pairwise Word Interaction Model	**0.7090**	**0.7234**

Table 4.5: Test results on WikiQA data.

Model	MAP	MRR
Cui et al. [33]	0.4271	0.5259
Wang et al. [157]	0.6029	0.6852
Heilman and Smith [63]	0.6091	0.6917
Wang and Manning [156]	0.5951	0.6951
Yao et al. [169]	0.6307	0.7477
Severyn and Moschitti [122]	0.6781	0.7358
Yih et al. [171]	0.7092	0.7700
Wang and Nyberg [151]	0.7134	0.7913
Severyn and Moschitti [123]	0.7459	0.8078
Pairwise Word Interaction Model	**0.7588**	**0.8219**
Multi-Perspective ConvNet	**0.756**	**0.830**
Multi-Perspective ConvNet+AttentionInput	**0.766**	**0.840**

Table 4.6: Test results on TrecQA data.

on sentence modeling. Our model outperforms them all.

TrecQA Results (Table 4.6). This is the second largest dataset in our experiments, with over 55,000 question-answer pairs. Only recently have neural network approaches [175] started to show promising results on this decade-old dataset. Previous approaches with probabilistic tree-edit techniques or tree kernels [63, 156, 169] have been successful since tree structure information permits a fine-grained focus on important words for similarity comparison purposes. Our approach essentially follows this intuition, but in a neural network setting with the use of our similarity focus layer. Our model outperforms previous

work.

TwitterPPDB Results (Table 4.7). We conduct experiments on the binary classification problem of the paraphrase/non-paraphrase, and compare the F1, precision, and recall scores of several representative and state-of-the-art models. This is the largest dataset in our experiments, our PWI model achieves the state-of-the-art performance.

We also introduce the baselines below. **GloVe** [102] is a word representation model that combines global matrix factorization and local context window methods. It provides 300-dimension pre-trained word embeddings for different text genres, which we sum together for each sentence. **WMF/OrMF** is Weighted Matrix Factorization (WMF) [54] is an unsupervised dimension reduction model. The unobserved words are carefully handled, which results in more robust embeddings for short texts. Orthogonal Matrix Factorization (OrMF) is the extension of WMF, with an additional objective to obtain nearly orthogonal dimensions in matrix factorization. The benefit is that redundant information is discounted in the model. **LR** represents the logistic regression (LR) model incorporates 18 simple features based on 1-4 gram overlaps, computed directly from a pair of sentences (s_1 and s_2) [34]. The features are of the form precision$_n$ (number of n-gram matches divided by the number of n-grams in s_1), recall$_n$ (number of n-gram matches divided by the number of n-grams in s_2) and F$_n$ (harmonic mean of recall and precision). The model also includes lemmatized versions of these features. We build our own reimplementation using the Scikit-learn[1] toolkit. **LEX-WMF/LEX-OrMF** This is a state-of-the-art ensemble system that combines WMF or OrMF into the above LR respectively [166]. It is a simplified open-sourced version of LEXDISCRIM [71]. Specifically, as for **(vec)** version, the

[1] http://scikit-learn.org/stable/

Method	F1	Precision	Recall
Random	0.418	0.356	0.507
Edit Distance	0.466	0.790	0.330
GloVe (sim)	0.586	0.436	0.891
LR (sklearn)	0.764	0.812	0.721
WMF (vec)	0.715	0.654	0.789
LEX-WMF (vec)	0.757	0.687	0.842
OrMF (vec)	0.709	0.652	0.777
LEX-OrMF (vec)	0.754	0.685	0.837
WMF (sim)	0.756	0.721	0.795
LEX-WMF (sim)	0.778	0.727	0.837
OrMF (sim)	0.759	0.720	0.802
LEX-OrMF (sim)	0.776	0.726	0.833
MultiP [166]	0.698	0.611	0.813
VDPWI	**0.784**	**0.787**	**0.781**

Table 4.7: Paraphrase models on TwitterPPDB Corpus. Our PWI model outperforms all baseline systems.

latent representation of a pair of sentences $\vec{v_1}$ and $\vec{v_2}$ is converted into a feature vector, $[\vec{v_1} + \vec{v_2}, |\vec{v_1} - \vec{v_2}|]$, by concatenating the element-wise sum $\vec{v_1} + \vec{v_2}$ and absolute different $|\vec{v_1} - \vec{v_2}|$. We also provide the comparison with the **(sim)** variation, which uses the single cosine similarity score between two sentence vectors directly.

SemEval2017 Results (Table 4.8). This is the cross-lingual experiment. The cross-lingual STS task is a relatively new task, the datasets we used are from recent Semeval 2017 [21] competition. There are a total of 3 language pairs in 4 tracks: Spanish-English, Turkish-English, and Arabic-English. We followed the same competition settings as done by other teams. Pearson scores (the official measure) on all 4 cross-lingual tracks are shown and our PWI model is competitive in comparison to the top 3 systems. The Baseline is a straightforward cosine distance baseline with basic sentence embedding method for monolingual similarity as provided officially by SemEval organizers. Note Track4b is special in that it has many OOV vocabularies on purpose from different domains; Track6

System	Arabic-English (Track2)	Spanish-English (Track4a)	Spanish-English (Track4b)	Track 6
Cosine Baseline	0.5155	0.6220	0.0320	0.5456
ECNU [139]	0.7493	0.8131	**0.3363**	**0.7706**
BIT	0.7493	0.8131	0.3363	0.7706
HCTI	0.6836	0.7621	0.1483	0.6741
PWI	**0.7624**	**0.8147**	0.1282	0.7449

Table 4.8: We show the top 3 systems in the SemEval 2017 crosslingual compoetition on the four cross-lingual tracks. Our PWI model achieves competitive performance.

serves as a surprise track in the competition in that there is no training data.

We tried different settings for cross-lingual experiments. We firstly used cross-lingual word embeddings because cross-lingual embeddings have reasonable assumptions: all cross-lingual words should be mapped to the same vector space, if words in two languages have similar meanings they should stay close to each other in the low-dimensional space. We used one recent cross-lingual embeddings [129] (*fastText_multilingual*). However we found out even though cross-lingual embeddings are evaluated extensively on word-level test sets, they still have difficulties to be applied directly on to downstream sentence-level tasks such as textual similarity tasks. We obtained significantly lower performance when using cross-lingual embeddings in comparison to the ones shown in the table. In the end we had to follow almost all teams in the SemEval competition and used machine translation system (which is also approved by the competition) to translate non-English sentences to English first, then finally applied our PWI model for evaluation.

4.8 Analysis

Ablation Studies. Table 4.9 shows the results of ablation studies on SICK and WikiQA data. We removed or replaced one component at a time from the full system and performed re-training and re-testing. We found large drops when removing the context

Ablation on SICK Data	Pearson
Full Model	0.8784
- Remove context modeling (Sec. 4.2)	-0.1225
- Remove entire focus layer (Sec. 4.4)	-0.0083
- Replace entire focus layer with dropout	-0.0314
Ablation on WikiQA Data	**MRR**
Full Model	0.7234
- Remove context modeling (Sec. 4.2)	-0.0990
- Remove entire focus layer (Sec. 4.4)	-0.0327
- Replace entire focus layer with dropout	-0.0403

Table 4.9: Ablation studies on SICK and WikiQA data, removing each component separately.

Model	# of Parameters	Timing (s)
Multi-Perspective ConvNet	10.0 million	2265
Pairwise Word Interaction Model	**1.7 million**	**664**

Table 4.10: Comparison of training efficiency and number of tunable model parameters on SICK data. Timing is the average epoch time in seconds for training on a single CPU thread.

modeling component, indicating that the context information provided by the Bi-LSTMs is crucial for the following components (e.g., interaction modeling). The use of our similarity focus layer is also beneficial, especially on the WikiQA data. When we replaced the entire similarity focus layer with a random dropout layer ($p = 0.3$), the dropout layer hurts accuracy; this shows the importance of directing the model to focus on important pairwise word interactions, to better capture similarity.

Model Efficiency and Storage. Our MPCNN model uses multiple types of convolution and pooling for sentence modeling. This results in a wide architecture with around 10 million tunable parameters. Our approach only models pairwise word interactions and does not require such a complicated architecture. Compared to that previous work, Table 4.10 shows that our model is 3.4$\times$ faster in training and has 83% fewer tunable parameters.

	A	man	is	playing	the	drum
A	8.99	0.69	0.43	0.32	0.38	0.22
man	0.70	9.93	0.62	0.45	0.46	0.38
is	0.64	0.80	8.50	0.62	0.58	0.36
practicir	0.46	0.67	0.66	6.51	0.62	0.48
the	0.35	0.56	0.66	0.64	7.85	0.52
drum	0.27	0.47	0.46	0.55	0.64	8.82
	A	man	is	carrying	a	tree
A	0.53	0.33	0.32	0.33	5.53	0.49
tree	0.32	0.30	0.19	0.20	0.38	8.73
is	0.35	0.31	0.21	0.06	0.03	0.40
being	0.28	0.37	2.60	0.18	0.13	0.38
picked	0.15	0.18	0.10	1.60	0.07	0.27
up	0.26	0.27	0.06	0.13	0.05	0.21
by	0.43	0.36	0.08	1.33	0.15	0.29
a	6.50	0.45	0.03	0.08	0.16	0.23
man	0.50	8.60	0.45	0.34	0.34	0.34

Table 4.11: Visualization of cosine values (multiplied by 10) in the $focusCube$ given two sentence pairs in the SICK test set.

Visualization. Table 4.11 visualizes the cosine value channel of the $focusCube$ for pairwise word interactions given two sentence pairs in the SICK test set. Note for easier visualization, the values are multiplied by 10. Darker red areas indicate stronger pairwise word interactions. From these visualizations, we see that our model is able to identify important word pairs (in dark red) and tag them with proper similarity values, which are significantly higher than the ones of their neighboring unimportant pairs. This shows that our model is able to recognize important fine-grained word-level information for better similarity measurement, suggesting the reason why our model performs well.

4.9 Summary

In this chapter we introduced a novel neural network model with a focus on word-level interaction modeling for the textual similarity measurement task. Our pairwise word

interaction model and the similarity focus layer can better capture fine grained semantic information, compared to previous approaches with the Siamese architecture that attempt to cram all sentence information into a fixed-length vector. We demonstrated the highly competitive accuracy of our approach over multiple datasets. In the next chapter we apply our PWI and MPCNN models to pairwise learning to rank task, again with a focus on interaction-based modeling.

Chapter 5: Pairwise Learning to Rank

In this chapter we utilize both our pairwise word interaction model (PWI) as introduced in Chapter 4 and multi-perspective convolutional neural network model (MPCNN) as introduced in Chapter 3, and we again focus on interaction-based modeling with joint representation learning over pairs of inputs for the answer selection task in a pairwise ranking fashion [110].

As introduced in Chapter 2, answer selection is an important component of an overall question answering system: given a question q and a candidate set of sentences $\{c_1, c_2, \ldots c_n\}$, the task is to identify sentences that contain the answer. In a standard pipeline architecture [138], answer selection is applied to the output of a module that performs textual retrieval or lightweight term-based matching. Selected sentences can then be directly presented to users or serve as input to subsequent stages that identify "exact" answers [148].

Although answer selection is formally considered a *pointwise* classification problem, in reality candidate sentences are ranked in decreasing probability of containing the answer, and results are evaluated using similarity measurement metrics on ranked lists. The nature of the task inspired us to formalize it as a pairwise learning to rank problem.

We develop a *pairwise* learning to rank approach to learn the relative order of an-

swer pairs with the use of Noise-Contrastive Estimation (NCE). Given a question sentence, our NCE approach takes a pair of candidate answer sentences as input and learns to predict which answer is more relevant to the question. We use NCE to learn joint representations of the triplet input (question, positive answer, negative answer) directly, then stack a triplet ranking loss function on top to learn nonlinear feature correlations from the joint representations. The objective is to minimize the total number of inversions in the rankings. Unlike traditional pointwise approach of the task, our pairwise ranking approach with NCE focuses on modeling interactions between two pairs of question and answer inputs.

Our approach is flexible in that it can take advantage of existing pointwise models as base components. We demonstrate the effectiveness of our approach against competitive pointwise baselines [56, 59, 95, 119, 124, 159, 171, 176]. We show that joint representation learning on a pairwise learning to rank fashion from triplets with NCE comprising the question and both positive and negative answers is superior than learning a pointwise representation on (question, positive answer) pairs. In contrast to previous work, our proposed pairwise ranking approach takes advantage of Noise-Contrastive Estimation (NCE) that a good model should be able to discriminate a good sample from its neighboring bad samples. Given pairs of positive and negative samples, the model should learn differentiable representations to distinguish positive and negative samples. Experiments on both TrecQA and WikiQA datasets show that our approach achieves state-of-the-art effectiveness without using sparse features, syntactic parsers, or external knowledge sources like WordNet.

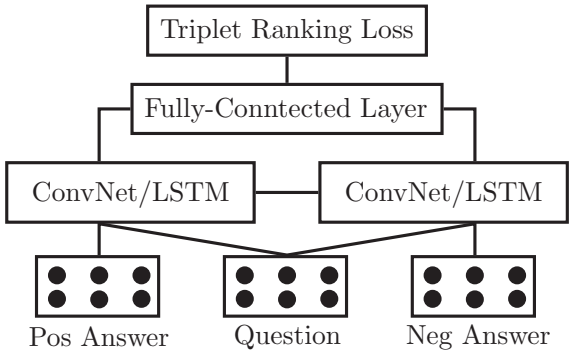


Figure 5.1: Architecture of our pairwise ranking model, which is trained on triplets comprised of (question, positive answer, negative answer).

5.1 Model Architecture

We show the overall architecture of our pairwise ranking model in Figure 5.1, consisting of two major components. The base component is comprised of two pointwise neural network models, each of which takes a pair of (question, answer) sentences and produces a similarity score to represent the semantic distance of the pair. Our pairwise ranking model is expected to output a larger similarity score given the positive pair and a smaller score given the negative pair. On top, we evaluate a triplet ranking loss function (see below) to learn the joint representation of answer pairs.

Our pairwise ranking model consists of two parallel pointwise models, each processing a pair of question and answer sentences. This is different from the pointwise "Siamese" architecture in that the inputs are now pairs of questions and answers. Parameters of the two base models are shared during training. In testing, we take one of the two base models to produce the feature representation of a (question, answer) pair, and then feed those features into the fully-connected layer for computing a relevance score. As discussed earlier in this dissertation although many previous pointwise models also use

the "Siamese" structure, there are important differences: each base component of a typical pointwise model only takes one input sentence, whereas each base component of our pairwise model takes a (question, answer) pair as input and explores features interactions between the sentence pairs.

In this work we use our own existing work [56, 59] as introduced in Chapter 4 and Chapter 3 as the base models; these have achieved highly competitive effectiveness on multiple tasks, i.e., paraphrase identification, semantic textual similarity measurements, and question answering. More importantly, these two models represent different approaches to model design. As described in previous chapter, our MPCNN model [59] focuses on *sentence-level* interaction-based modeling, which tries to capture semantic similarities from multiple perspectives by applying different types of similarity metrics, convolutional filters, poolings, and window sizes on input sentences. This model also incorporates a structured similarity layer over sentence representations for fine-grained comparisons. Our PWI model [56] proposes a *word-level* model and develop a pairwise word interaction layer to explicitly identify word pairs across input sentences, a similarity focus layer to guide model attention onto important word pairs, and a 19-layer very deep ConvNet for classification.

Experimental results in Section 5.2 show that our pairwise learning to rank approach, when paired with each as the base component, is able to make further effectiveness improvements.

5.1.1 Triplet Ranking Loss Function

Both word-level and sentence-level base components take input sentence pairs of (question, answer) and produce a score through a representation function $f(\cdot)$. This score captures how semantically similar the two input sentences are. Our goal is to learn a representation function $f(\cdot)$ such that given some question q, positive pairs (q, p^+) are assigned larger similarity scores than negative pairs (q, p^-):

$$f(q, p^+) > f(q, p^-), \forall q, p^+, p^-$$

where q, p^+, p^- denote the question, positive answer, and negative answer, respectively. We then use a triplet ranking loss, which minimizes the distance between the question q and a positive answer p^+, and maximizes the distance between the q and a negative answer p^-, summed over positive pairs (q, p^+) and the corresponding negative pool $p^- \in N(q, p^+)$:

$$\min_W \sum_{(q,p^+)} \sum_{p^- \in N} max(0, 1 - (f(q, p^+) - f(q, p^-)) + \lambda \|W\|^2$$

where λ is a regularization parameter, and W is the parameters of neural network model $f(\cdot)$.

5.1.2 Sampling Strategy

To counter overfitting, it is common practice to train a model with a large variety of samples. However, due to its pairwise nature, our pairwise ranking model requires $O(N^2)$ time complexity to enumerate all pairs, which is computationally impractical. Therefore, it is essential to balance the coverage of training samples and limit computational resources. We use three negative sampling strategies to select the most informative negative samples.

Random Sampling. We randomly select a number of negative samples for each positive answer.

Max Sampling. We select the most difficult negative samples. In each epoch, we compute the similarities between all (p^+, p^-) pairs using the trained model from the previous training epoch. Then we select the negative answers by maximizing their similarities to the positive answer:

$$maxNeg^i(p^-) = \arg\max_{p^-} sim(f^{i-1}(q, p^+), f^{i-1}(q, p^-))$$

where $maxNeg^i(p^-)$ is the selected negative sample in epoch i, $f^{i-1}(\cdot, \cdot)$ is the trained model in epoch $i-1$, and sim is the *cosine* distance. In the first epoch, we randomly select negative samples.

Mix Sampling. We take advantages of both random sampling and max sampling by selecting half of the samples from each strategy.

5.2 Experimental Setup

We evaluated our pairwise ranking model on two major question answering datasets, WikiQA [168] and TrecQA [158]. Please refer to Chapter 2 for details.

The TrecQA dataset [158] is a widely-used benchmark for question answering as packaged by Yao et al. [170]. However, in the literature [95, 119, 124, 159, 171, 176], we observe two versions of TrecQA: both have the same training set but their development and test sets differ due to different pre-processing.

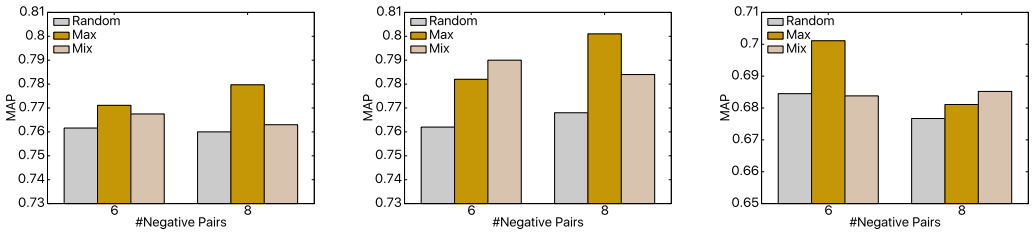
Previous work [56, 95, 124, 176] used the version that has 82 questions in the development set and 100 questions in the test set (what we call "Raw TrecQA"). However, there exists questions that have no answer sentences.[1] More recent work [119, 159] further cleaned the dataset by removing questions that have only positive/negative answers or no answers, resulting in only 65 questions in the development set and 68 questions in the test set (what we call "Clean TrecQA"). We evaluated our model on both versions for a fair comparison against previous work, and we show that the MAP/MRR scores reported on both TrecQA versions are *not* comparable. Relevant statistics are shown in Table 5.1.

We conducted experiments with two base components, each with its own settings. We used the GloVe word embeddings [102] for the sentence-level model [59] and PARAGRAM-PHRASE word embeddings [163] for the word-level model [56]. Both word embeddings have 300 dimensions. Words not found in the vocabulary were initialized randomly with values uniformly sampled from $[-.05, .05]$. We did not update word embeddings in all

[1] MAP and MRR scores computed with the official `trec_eval` scorer do not change if questions with empty answer sets are removed, since the scorer will ignore questions with empty answer set.

Dataset	Split	#Questions	#Pairs	%PosRate
Raw TrecQA	TRAIN	1229	53417	12.0
	DEV	82	1148	19.3
	TEST	100	1517	18.7
Clean TrecQA	DEV	65	1117	18.4
	TEST	68	1442	17.2
WikiQA	TRAIN	873	8672	12.0
	DEV	126	1130	12.4
	TEST	243	2351	12.5

Table 5.1: Statistics of TrecQA and WikiQA datasets



(a) Raw TrecQA (b) Clean TrecQA (c) WikiQA

Figure 5.2: Comparison of sampling strategies for PairwiseRank+SentLevel.

experiments. SGD with a learning rate of 10^{-3} for the sentence-level base component and RMSPROP with a learning rate of 10^{-4} for the word-level base component were used for training. We chose these different word embeddings and optimizers in order to keep experimental settings the same as in previous work [56, 59]. We used the $\tanh$ function as the non-linear activation function and a dropout layer ($p = 0.5$) in the fully-connected layer, plus parameter regularization ($\lambda = 10^{-4}$). Our code and data are available online.[2]

Effectiveness is measured in terms of Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR). In all experiments, we selected training models that obtain the best MRR scores on the development set for testing.

[2] https://github.com/castorini/NCE-CNN-Torch

Reference	MAP	MRR
Wang et al. [153] s(2015)	0.713	0.791
Miao et al. [95] (2015)	0.734	0.812
Severyn et al. [124] (2015)	0.746	0.808
SentLevel [59] (2015)	0.762	0.830
WordLevel [56] (2016)	0.755	0.825
PairwiseRank+SentLevel	**0.780**	**0.834**
PairwiseRank+WordLevel	0.764	0.827

(a) Raw TrecQA

Reference	MAP	MRR
Santos et al. [119] (2016)	0.753	0.851
Wang et al. [159] (2016)	0.771	0.845
SentLevel [59] (2015)	0.777	0.836
WordLevel [56] (2016)	0.738	0.827
PairwiseRank+SentLevel	**0.801**	**0.877**
PairwiseRank+WordLevel	0.762	0.854

(b) Clean TrecQA

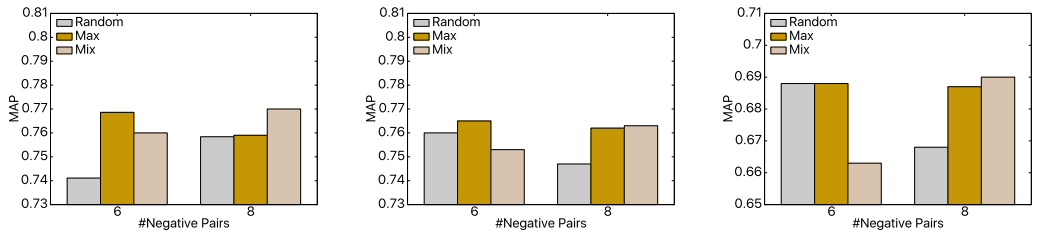
Reference	MAP	MRR
Miao et al. [95] (2015)	0.689	0.707
Santos et al. [119] (2016)	0.689	0.696
Wang et al. [159] (2016)	0.706	0.723
SentLevel [59] (2015)	0.693	0.709
WordLevel [56] (2016)	**0.709**	**0.723**
PairwiseRank+SentLevel	0.701	0.718
PairwiseRank+WordLevel	0.693	0.710

(c) WikiQA

Table 5.2: Results on TrecQA and WikiQA.

5.3 Results

The results of our experiments on the three different test sets are shown in Table 5.2. SentLevel denotes the MPCNN model [59] and WordLevel denotes the PWI model [56]. The rows with PairwiseRank show our pairwise ranking approach with the use of the two different base components (either the WordLevel or SentLevel model). We compare our results to others reported in the literature on an ACL wiki site [1].



(a) Raw TrecQA (b) Clean TrecQA (c) WikiQA

Figure 5.3: Comparison of sampling strategies for PairwiseRank+WordLevel.

Our best results for all the three datasets in Table 5.2 were obtained from max sampling. On both the raw and clean versions of the TrecQA data, PairwiseRank+SentLevel model achieves MAP and MRR scores that are among the best reported in the literature. The models achieve better effectiveness on clean TrecQA than on raw TrecQA. This is because the clean version removes questions with only negative answers: these questions will always have zero values when computing MAP and MRR scores, thereby degrading the overall effectiveness. Thus, numbers reported on raw and clean TrecQA data should not be compared.

In terms of the comparison between the original models and our pairwise ranking models, the two original base models (WordLevel and SentLevel) achieve competitive effectiveness, while our pairwise ranking models (PairwiseRank +SentLevel/WordLevel) can still improve on them in most cases. On the clean TrecQA data, this improvement is quite large. On the WikiQA data, the WordLevel model remains the best, but PairwiseRank+SentLevel improves over the base SentLevel model, bringing its effectiveness up to being on par with our best model.

Overall, these empirical results show that the joint representations learned from the triplet inputs are effective, and that our pairwise ranking approach is able to exploit such joint information.

We also studied the effects of the three different sampling strategies. Figures 5.2 and 5.3 show the effectiveness of different sampling strategies with PairwiseRank+SentLevel and PairwiseRank+WordLevel models. We set the number of negative samples to six and eight. We only show the MAP figures here due to space limitations, but the patterns for MRR scores are generally similar. For the PairwiseRank+SentLevel and PairwiseRank+WordLevel models, max and mix sampling consistently outperforms random sampling. In most settings, max sampling obtains the best performance. This shows that more "challenging" negative training samples are beneficial for training a better model in our pairwise ranking approach.

5.4 Summary

In this work, we propose and evaluate a pairwise learning to rank approach using NCE with a focus on modeling pairwise input interactions. Our pairwise learng to rank approach can also utilize our developed MPCNN and PWI models as plug-in components. Experiments show that our approach can improve upon the component models under competitive settings on standard QA datasets with the NCE training. We also present three strategies for selecting negative samples and demonstrate the effectiveness of selecting the most "difficult" training examples to distinguish between good and bad answers.

It must be noted that this work was jointly done with Jinfeng Rao in the CLIP lab at UMD. I thank Jinfeng for all the great contributions.

Next in Chapter 6, we show how we build an end-to-end system to extract insights from biomedical literature, and also introduce our novel textual similarity modeling to

address the problem.

Chapter 6: Textual Similarity Modeling for Insight Extraction on BioMedical Literature

Biomedical literature offers a rich set of knowledge sources to discover important facts and find associations among them. For instance, MEDLINE contains over 18 million references to articles published since 1946 and sourced from over 5500 journals worldwide [128]. Two major processing tasks performed on the biomedical text are: (1) identify and classify biomedical entities (NER) into predefined categories such as proteins, genes, or diseases, and (2) infer pair-wise relationships among named entities e.g., protein-protein interaction [107], gene-protein, and medical problem-treatment.

This chapter presents an end-to-end system we build [62] that processes biomedical text to automatically extract two specific types of relationships among biomedical entities: (a) cause-effect and (b) correlation. We focus on the similarity modeling and model interactions among representations of biomedical entities, the relationship, and contexts.

It must be noted that the approaches discussed in this chapter are purely data-driven. That is, we do not provide our own definitions of "causality" or "correlation", also we do not develop new theories to distinguish both. Our neural network models only learn from annotations based on annotators' understanding of those definitions.

This system is motivated by the need to better automate biomedical knowledge

extraction and identify important information from them, as new scientific findings appear across a large collection of publications. For instance, given user sleep patterns, existing biomedical research can be better utilized to provide *insights*: inform about potential *effect* (e.g., "*diabetes*", "*obesity*") due to the *cause* (e.g., "*sleep disorder*") and suggest appropriate treatment.

Since biomedical articles usually have title and abstract summarizing the contents of the full-text article, we focus on extracting the two relationship types from them. Unfortunately, mining this summary data still poses several key challenges. Similar to full-text, this data comprises unstructured text with domain-specific vocabulary, issues of synonymy (e.g., *"heart attack"* vs. *"myocardial infarction"*), acronyms, abbreviations and rapidly evolving terminology due to new scientific discoveries. While the titles are short and informative, they do not contain the key information that would be contained in the abstract.

Many of these challenges are also applicable for biomedical relation extraction. Further, identifying particular relation types is challenging because relations are expressed as discontinuous spans of text, and the relation types are typically application-specific. Finally, there is often little consensus on how to best annotate relation types resulting in lack of high quality annotated corpora for training.

We develop neural networks with novel similarity modeling for better causality / correlation relation extraction, as we map the extraction task into a representational similarity measurement task in the vector space. Our approach innovates in that it explicitly measures both relational and contextual similarity among representations of named entities, entity relations and contexts. Our system also provides a novel combination of

recognizing named entities, predicting relationships (insights) between extracted entities, and ranking the output. We conduct human evaluations of the system to show it is able to extract insights with high human acceptance accuracy, and on a SemEval task evaluation its causality/correlation relation extraction compares favorably against previous state-of-the-art work.

Our major contributions in this chapter include:

1. We build an end-to-end system to extract insights from biomedical literature.
2. We innovate in similarity measurement modeling with deep neural networks for better relation extraction.
3. Our human evaluation show our system can achieve competitive acceptance accuracy.

6.1 System Overview

We provide a recipe to build a system for biomedical insight extraction and use it as a guide of the system (Algorithm 5).

To make our discussion concrete, we will use a sample biomedical article in Example 1. Given the text, at line 4 of Algorithm 5 we firstly look for all named entities using a shallow parser and public medical dictionaries (see details in Section 6.2). Many named entities could be found, for example, *"clinical study"*, *"sleep disturbances in middle-aged men"* and *"diabetes"*. Next given any pair of previously extracted entities within a sentence, at line 6 our neural network-based relation extractor checks if a valid causality/correlation relationship exists (Section 6.3). For example, our models can identify

Algorithm 5 System Overview

```
1: Input: Biomedical article title and abstract
2: Preprocess the input texts
3: for each sentence of the input do
4:     Identify all possible named entities
5:     for each named entity pair (A⃗, B⃗) do
6:         if causality/correlation holds then
7:             Extract and Score (A⃗, B⃗)
8:         end if
9:     end for
10: end for
11: Rank all extracted (A⃗, B⃗) pairs
12: return top ranked entity pairs
```

that the entity *"sleep disturbances in middle-aged men"* has a correlation relationship with *"diabetes"* but not with *"clinical study"*. Later each valid entity pair is scored via the ranking component at line 7 (Section 6.4). In the final step, the system returns top ranked insight(s) to users: "*sleep disturbances in middle-aged men* $\rightarrow$ *diabetes*" given this example.

> ***RESEARCH METHODS****: A group of 6,599 initially healthy, nondiabetic middle-aged men took part in a prospective, population-based study. The incidence of diabetes during a mean follow-up of 14.8 years was examined in relation to self-reported difficulties in falling asleep.*
> ***RESULTS****: A total of 615 subjects reported either difficulties in falling asleep or use of hypnotics (seen as makers of sleep disturbances). Among those, 281 of the men developed diabetes during the follow-up period. The clinical study suggests sleep disturbances in middle-aged men are likely associated with diabetes.*

Example 6.1: Sample Text

Figure 6.1 presents the system which consists of three major neural network-based components: (1) a named entity extractor, (2) a causality/correlation relation extractor, and (3) an insight ranker. Our system reads in biomedical texts, then provides insights in the end. We primarily innovate in the relation extraction component. Next, we describe

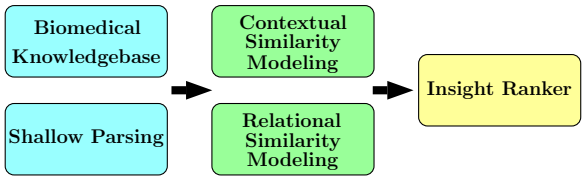


Figure 6.1: Three major components of the system.

each of these components in detail.

6.2 Named Entity Extraction

Named entity extraction in biomedical domain is challenging due to the domain-specific and rapidly evolving terminology. For example, *"Diabetes mellitus type 1"*, *"Type 1 diabetes"*, *"IDDM"*, or *"juvenile diabetes"* all express the same concept. Given frequent evolution of entity naming for new drugs, diseases or abbreviations, this task becomes more complicated.

Most existing off-the-shelf biomedical entity recognizers narrowly focus on specific biomedical terms. Instead we aim to improve the system recall by extracting both specific biomedical concepts such as *"gene tmem230"* or *"prostate cancer"* as well as general noun phrases such as *"sleep quality"*, *"daily exercises"*, or *"men with diabetes"*. Thus the scope of the system is broader.

We design an entity extractor by using both an in-domain medical knowledge base for keyword matching, and a domain-independent neural network-based shallow parser for entity boundary detection. We present the procedure below:

1. We firstly use a large public dictionary, *Metathesaurus* of the Unified Medical Language System (UMLS) [15] to obtain in-domain biomedical terms. UMLS *Metathesaurus* is a set of dictionaries providing large collections of biomedical vocabularies.

We extract over 3.3 million of biomedical terms from UMLS, then utilize the *Aho Corasick* pattern matching algorithm to create a dictionary lookup tool. Our tool can efficiently locate all UMLS terms given input texts, since it has a linear complexity due to its trie tree data structure.

2. We also use a neural network-based shallow parser [30] to identify boundaries of general noun phrases, which are not limited to biomedical terms. Usage of shallow parser is to improve system recall on named entity recognition.
3. Our named entity extraction component aims to locate all entities of input texts. The result list is an output concatenation of both step 2 and 3, and is later provided to the causality/correlation relation extraction component for further processing. If entity overlaps exist, only phrases with longest matching sequence are extracted.

Our insight extraction system adopts a coarse-to-fine design approach. First, we focus on improving recall for the entity extraction task. Then we show how the causality/correlation relation extraction component (Sec. 6.3) processes extracted named entities to achieve high precision.

6.3 Relation Extraction as Similarity Measurement

We first provide our model design intuition: if a causality/correlation relationship holds between two named entities, then representations of the two entities should be semantically similar and close to the representation of the relation in a low-dimensional vector space. Therefore we map the causality/correlation relation extraction into a similarity measurement task in the vector space.

Our novel approach learns representations of named entities $(\vec{A}, \vec{B})$, context words and the relation vector $\vec{R}$, then explicitly measures two aspects of the similarity: 1) **relational similarity** between entities and relation (Sec. 6.3.2); plus, 2) **contextual similarity** between entities and sentence context (Sec. 6.3.3).

The intent of our approach is to enforce such structure of the vector space: as the similarity among entities, relation and contexts gets stronger, a fit of all should be observed for better causality/correlation relation extraction. We develop two neural network models with such property; both are utilized in the relation extraction component of the system.

We define input sentence representation $S \in \mathbb{R}^{\ell \times d}$ to be a sequence of ℓ words, each with a d-dimensional word embedding vector. $x_t \in \mathbb{R}^d$ denotes the embedding vector of the t-th word ($t \in [1, l]$) in S. Model details are described in the following sections.

6.3.1 Context Modeling

Different words occurring in similar contexts should have a higher chance to contribute to similarity measurement and relation extraction. We use bidirectional LSTMs (BiLSTM) for context modeling as a basis for all following models. We use LSTM for context modeling, please refer to Section 4.2 for details. We only show high level equations of LSTMs here:

$$h_t = LSTM(x_t, h_{t-1}) \tag{6.1}$$

$$h_t^{bi} = concat(h_t^{for}, h_t^{back}) \tag{6.2}$$

$$H^S[t] = h_t^{bi} \tag{6.3}$$

Context modeling with BiLSTM allows our following model components to be built over contexts rather than over individual words. Given named entity positions of the sentence, we get $\vec{A}$ and $\vec{B}$ from context H^S.

6.3.2 Relational Similarity Modeling

Relational similarity modeling focuses on interactions between named entities and relations in the vector space. When the named entity $\vec{A}$ goes through a transformation process induced by the relation $\vec{R}$, our intent of relational similarity modeling is to force the transformed entity to be translated to the other named entity $\vec{B}$ in the same vector space so that the relation $\vec{R}$ holds between the two named entities.

We show the following objective function of our relational similarity modeling:

$$\vec{A} + \vec{B} - \vec{R} \simeq 0 \tag{6.4}$$

To model the transformation process in Equation 6.4, we need to know how to measure the similarity of the triplet $(\vec{A}, \vec{B}, \vec{R})$. Therefore we develop a similarity measurement function $SimiScore(\vec{A}, \vec{B}, \vec{R})$ with learnable weights (W^*), the similarity function takes an input named entity pair of $(\vec{A}, \vec{B})$ and a relation $\vec{R}$, returns a similarity score

Function 1 $SimiScore(\vec{A}, \vec{B}, \vec{R})$

1: $conC = concat(\vec{A}, \vec{B})$
2: $entityC = W^C \cdot conC$
3: $relationT = W^D \cdot \vec{R}$
4: $dist = W^{di} \cdot tanh(entityC + relationT)$
5: **return** $dist$

$dist \in R^1$ representing how semantically close $(\vec{A}, \vec{B}, \vec{R})$ are, as in Function 1.

We utilize a ranking approach during training to incorporate the constraint of Equation 6.4 into the relational similarity model. Our goal is to learn a function $SimiScore(\cdot)$ so that the positive triplet $(\vec{A}, \vec{B}, \vec{R^+})$ is assigned a larger score than that of the negative triplet $(\vec{A}, \vec{B}, \vec{R^-})$:

$$SimiScore(\vec{A}, \vec{B}, \vec{R^+}) > SimiScore(\vec{A}, \vec{B}, \vec{R^-}) \tag{6.5}$$

where R^+ denotes the positive causality/correlation relation, R^- denotes a non-causality/non-correlation relation. The ranking approach maximizes the similarity score between the entity pair $(\vec{A}, \vec{B})$ and a positive relation $\vec{R}^+$ while minimizing the score with the negative $\vec{R}^-$, thus ensuring that the positive connection is larger than the negative one as in Figure 6.2.

Our relational similarity model and the ranking training approach facilitate the transformation process of ($\vec{A}$, $\vec{B}$) and $\vec{R}$ in the vector space, which in the end leads to better constraint satisfaction of objective Equation 6.4.

The relational similarity model is placed on top of BiLSTM as part of the system. We initialize named entities $\vec{A}/\vec{B}$ as h_A^{bi}/h_B^{bi} from the BiLSTM model, then initialize relation representations $\vec{R^+}/\vec{R^-}$ as random vectors. During training both $\vec{R^+}/\vec{R^-}$ are updated.

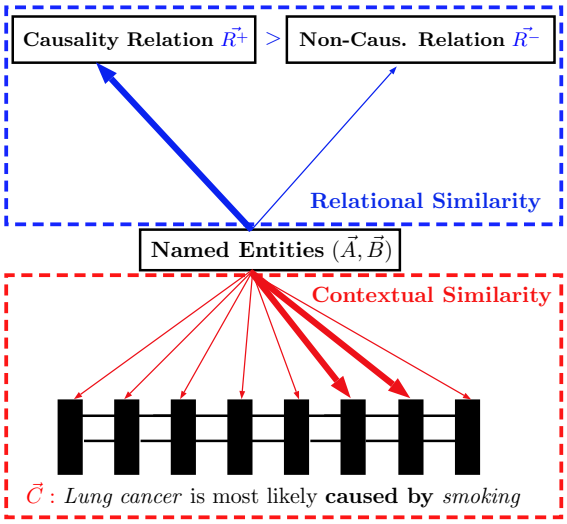


Figure 6.2: Our causality/correlation relation extraction component models both relational similarity (blue) and contextual similarity (red). Thicker arrows indicate stronger similarity between named entities $(\vec{A}, \vec{B})$ and relation $\vec{R}$/sentence context.

6.3.3 Contextual Similarity Modeling

Since not all words of a given title/abstract are created equal, important context words around named entities that can better contribute to the causality/correlation relation extraction deserve more model focus. We develop a contextual similarity model that can increase model weights onto important context words to better utilizing contextual information.

For example, given a sentence, *lung cancer is most likely caused by smoking*, the context words *caused by* are important clues to suggest there exists a causality/correlation relationship between the two named entities. Clue words that require model attentions usually include, e.g. *lead to, is associated with, because of*, while others are not obvious, such as *promote, reflect, reduce, make*.

Our system does not require a manually prepared list of clue words, but an attention mechanism [7] is utilized to better identify them by conducting similarity measurement

between context word representation h_t^{bi} (not including entity words) and extracted named entities $(\vec{A}, \vec{B})$ (from Sec. 6.2). Resulting similarity scores of words are accumulated in $atten \in \mathbb{R}^{\ell}$.

$$mix = W^a \cdot concat(\vec{A}, \vec{B}) \tag{6.6}$$

$$E[t] = dotProd(mix, h_t^{bi}), \forall t \in [1, l] \tag{6.7}$$

$$atten = softmax(E) \tag{6.8}$$

where we concatenate both entity representations $(\vec{A}, \vec{B})$, apply linear transformation with weights W^a to obtain a representation mix of both entities. We then use dot product $dotProd$ to measure the similarity between mix and each context word, finally normalize the attention weights $atten[:]$ with $softmax$. The weights of $atten$ indicate the importance of each context word with respect to both named entities.

The attention weights should better guide the focus of the model onto important context words of the sentence. That is, context words that are closer to entity representation mix should have better chances to be clue words. We define the attention re-weighted sentence representation $attenSen \in \mathbb{R}^{2dim}$:

$$attenSen = atten \odot H^S \tag{6.9}$$

where $\odot$ represents element-wise multiplication.

Figure 6.2 illustrates an example where representation mix of named entities at-

tends to context words one at a time. Important context clue words "*caused by*" should receive higher attention weights than irrelevant neighbor words.

The re-weighted sentence representation $attenSen$ is used together with entity representations $(\vec{A}, \vec{B})$ for final prediction.

In summary, both models described in this section focus on different aspects of similarity measurement in relation extraction: the contextual similarity model utilizes context information around named entities, while the relational similarity model focuses on enforcing a transformation constraint between entities and relation in the vector space. We adopt both models for better relation extraction, in the end only pairs of named entities that are recognized positively by either one of the models are passed to the next stage of the system.

6.4 Ranking of Extracted Insights

The last major component of our system is to rank extracted relations $(\vec{A}, \vec{B}, \vec{R})$ from the output of the relation extraction component, as there could be many extracted relations but not all of them are important enough as insights of the article. Importance scores of extracted relations are obtained by following a set of rules below:

1. We utilize the output classification probability ($\in [0, 1]$) of the relational similarity model as the base ranking score.

2. We use our MPCNN model [59] as introduced in Chapter 3 to measure the similarity ($\in [0, 1]$) between the title of the article and extracted relation, since the MPCNN model has competitive performance on multiple benchmarks for textual similarity

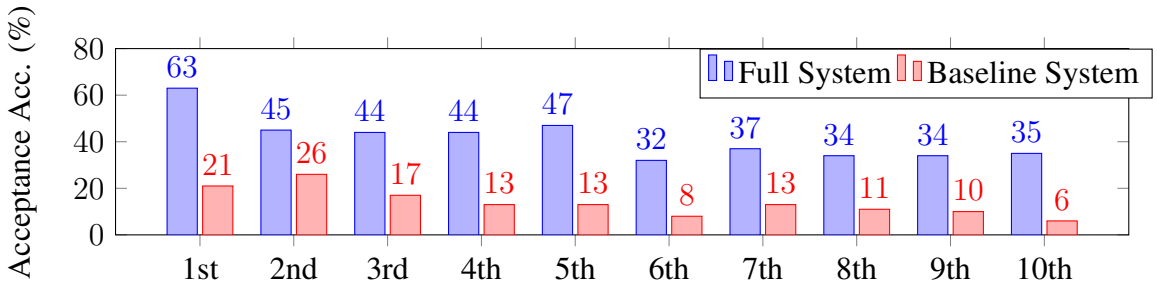


Figure 6.3: Human evaluation results of the full system and a baseline system on UHRS. We show the acceptance accuracy for each of the top ten positions given both systems' output lists. We primarily focus on the first 1 and 3 positions, namely *Precision*@1 and *Precision*@3.

measurement. We compare title text with "$\vec{A}$ leads to $\vec{B}$" of an extracted relation, if the similarity score is over a threshold of 0.75, we increase the extracted relation's ranking score by 15%. If the extracted relation is from the title text, we also boost its ranking score by 15% because of its location importance.

Once all extracted relations are scored, our system only returns the top ranked insights to users.

6.5 Experimental Setup

We use two datasets for evaluation of our system and models. The two datasets are SemEval-2010 and our own annotated biomedical literature datasets. Please refer to the chapter of background of related work for details.

Training. Two loss functions are adopted to train relation extraction neural network models.

For contextual similarity model (Sec. 6.3.3), a hinge loss is used. The training objective is to minimize the following loss, summed over examples $\langle x, y_{gold} \rangle$:

$$loss_{contextSim}(w, x, y_{gold}) =$$

$$\sum_{y' \neq y_{gold}} \max(0, 1 + f_w(x, y') - f_w(x, y_{gold})) \tag{6.10}$$

where input x represents an entity pair $(\vec{A}, \vec{B})$ plus its sentence context, y_{gold} is the ground truth label and y' is the model predicted label. Both y' and y_{gold} indicate the relation type with directionality (e.g. directional causality). w represents weights of contextual similarity model with BiLSTM, function $f_w(x, y')$ outputs the model predicted label value, function $f_w(x, y_{gold})$ outputs the model ground truth label value, and n is the number of training examples.

For relational similarity model (Sec. 6.3.2), a Bayesian Personalized Ranking (BPR) loss [113] is used. The label of the relational similarity model is binary because the BPR loss ranks positive inputs above negative inputs, thereby requiring the supervision signal to distinguish positives from negatives. Due to BPR loss's ranking nature, each training instance of the relational similarity model include one positive input $(x, \vec{R^+})$ and one negative input $(x, \vec{R^-})$. Given a positive correlation/causality input $(\vec{R^+})$, we generate negative training examples by matching the input x with each of the negative relation labels $(\vec{R^-})$. BPR loss is shown to be better tailored for ranking tasks empirically [147]:

$$loss_{relationSim}(w, x, \vec{R^+}, \vec{R^-}) =$$

$$\sum_{\vec{R^-}} -\log(\sigma(f'_w(x, \vec{R^+}) - f'_w(x, \vec{R^-}))) \tag{6.11}$$

where σ is the sigmoid function, function $f'_w(x, \vec{R})$ represents the relational similarity model with BiLSTM, and outputs a similarity score for ranking purpose (Sec. 6.3.2).

In all experiments, we perform optimization using RMSProp [140] with backpropagation [17] and a learning rate fixed to 10^{-4} and a momentum parameter 0.9.

Settings and Preprocessing. We preprocess datasets with Stanford CoreNLP toolkit [93]. We tokenize, lowercase, sentence split and dependency parse all words of both datasets. We set LSTM hidden state $dim = 500$.

Two sets of $d = 300$-dimension word embeddings are utilized. The first one is 300-dimension GloVe word embeddings [101] trained on 840 billion tokens; for better biomedical/health domain adaptation, we also train second word embeddings using the GloVe toolkit on biomedical research articles with over 1 billion tokens. We do not update word embeddings in all experiments.

During system deployment, we only initialize input words with the medical word embeddings if they do not exist in GloVe embeddings' vocabulary. We also concatenate embeddings of both input words and their head words on dependency trees as input for relation extraction models. We follow the task settings and compute F1-score with the official evaluation script only on *Cause-Effect* subset of SemEval-2010 data, then the best model based on F1 is selected for final system deployment. We set a distance limit and do not extract relations between two named entities if the distance is larger than 15.

System Ablation Study	*Precision*@1
Full System	63%
- Remove ReRanker (Sec. 6.4)	-5%
- Replace with BiGRU (Sec. 6.3)	-42%

Table 6.1: Ablation studies on the full system.

6.6 Results

Human Evaluation of the Entire System. We firstly provide a full end-to-end evaluation of the system on UHRS with human annotators.

For each biomedical publication, top 10 candidate insights from the system are listed for further inspection. The annotators are required to understand the texts, carefully inspect each insight, finally either accept it if it is one of the article insights or simply reject it. The annotation task requires understanding of biomedical/health publications and is non-trivial, therefore the system evaluation is completed by five expert annotators, who all hold postgraduate degrees and/or have biomedical background.

We also provide a baseline system, of which its relation extraction component is a bidirectional gated RNN model (BiGRU) [27]. BiGRU model and the ranking component are major differences between our full and baseline system. Since typically only a limited number of key findings is presented in one article, we evaluate the system with an averaged acceptance accuracy at top 1 (*Precision*@1) and top 3 (*Precision*@3) positions of the output rank list, which represent on average the number of extracted insights accepted by annotators among the first 1 and 3 output.

Figure 6.3 shows annotation results with acceptance accuracies for each of the ten output positions given biomedical titles and articles. The *Precision*@3 of our full system is

Model	**F1 score⋆**
Tymoshenko et. al [145]	82.30%
Trastz et. al [144]	87.63%
Rink et. al [116]	89.63%
BiGRU	89.89%
TreeLSTM [97]	91.57%
Contextual similarity modeling	90.77%
Relational similarity modeling	**92.28%**

Table 6.2: Test results (F1 score) on the *Cause-Effect* subset(⋆) of SemEval-2010 dataset. Results are grouped as 1) Top 3 participating teams in SemEval-2010 competition; 2) Baseline BiGRU model; 3) Recent state-of-the-art treeLSTM model [97]; 4) Our work.

50.6%, which is significantly better than the baseline system's 21.3%. For top 3 extracted insights on the list, our full system on average have 1.5 insights accepted by annotators. Furthermore, the acceptance accuracy *Precision*@1 of our system is 63% in comparison to that of the baseline system's 21%.

Table 6.1 shows the ablation study on the removal of the ranking component (Sec. 6.4) and the replacement of BiGRU model for causality/correlation relation extraction. We observe significant performance difference.

Evaluation of Relation Extraction Component. We also evaluate the relation extraction component (Sec. 6.3) on *Cause-Effect* subset of SemEval-2010 dataset. Note our causality/correlation relation extraction component is *not* supposed to be a general purpose one, since our system only focuses on insight extraction of biomedical/health literature. We compare our relation extraction models against previous work on the *Cause-Effect* subset of the data, Table 6.2 shows our relational similarity model, without the use of sparse features or external resources such as WordNet, outperforms recent state-of-the-art treeLSTM model [97]. It also shows BiGRU model is reasonably competitive on this dataset,

Excess	oil	,	**dirt**	and	bacteria	cause	**acne**	.	
0	0	0	0	0	0	0.9962	0	0.0037	
The	**bombing**	resulted	in	the	**deaths**	of	1318	in	Hanoi
0	0	0.0005	0.9579	0.0415	0	0	0	0	0
Ambient	vanadium	pentoxide	**dust**	produces	**irritation**	of	the	eyes	...
0	0	0	0	0.99	0	0	0	0	...
Electron	**beam**	is	generated	by	an	explosive	**emission**	cathode	
0	0	0	0.0053	0.9946	0	0	0	0	

Table 6.3: Visualization of model attention weights $atten$ given four SemEval-2010 test sentences.

which is why we use it in our baseline system for comparison purpose.

6.7 Analysis and Case Study

Visualization of Contextual Similarity Model. We show values of attention weights, $atten$ of Equation 8 and 9 from within the contextual similarity model (Sec. 6.3.3). Given four sentences in the test set of SemEval-2010 data, the model predicts that all provided entity pairs (in bold) have the causality/correlation relation. From Table 6.3 we observe the model is able to do its expected job: it can recognize important clues words, such as "*result in*", "*produce*", "*generated by*" and "*cause*"; the model produces attention weights (each $\in [0, 1]$) to tell the importance of clue words for causality/correlation relation extraction. We also observe the model tends to focus more on prepositions of clue words, such as "*by*" of "*generated by*" and "*in*" of "*result in*", this is probably because we use head words as extra inputs (Sec. 6.5) to the model.

Case Study. We lastly provide case study of our system. We show two biomedical articles' titles and abstracts as examples, with only necessary omissions to remove irrelevant texts due to the space limit.

Given **Case 1**, our system outputs the top insight "*the slow negative shift of the*

Case 1: *Scalp recorded direct current potential shifts associated with the transition to sleep in man.* ***Abstract:*** *Cortical direct current (DC) potentials are considered to reflect the state of cortical excitability which may change characteristically from wakefulness to sleep. The present experiments examined changes in the scalp recorded DC potential in 10 healthy humans ... It is reasonable to assume that the slow negative shift of the DC potential at the transition from wakefulness to sleep reflects increased cortical excitability.*

Case 2: *Alcohol consumption and self-reported sunburn: a cross-sectional, population-based survey.* ***Abstract:*** *Heavy drinking has been associated with several cancers, including melanoma and basal cell carcinoma. ... 299,658 adults reported their use of alcohol in the preceding month and a history of sunburn in the preceding year. Approximately 33.5% of respondents reported a sunburn within the past year. ... Excessive drinking is associated with higher rates of sunburn among American adults. The observed relationship typifies the high-risk behavior associated with excessive drinking and suggests one pathway linking alcohol use with skin cancer.*

Example 6.2: Case Study

DC potential $\rightarrow$ *increased cortical excitability*" with a score of 0.71. Given **Case 2**, our system outputs top 3 insights: "*excessive drinking* $\rightarrow$ *skin cancer*" with a score of 0.55, "*excessive drinking* $\rightarrow$ *alcohol*" with a score of 0.43, and "*excessive drinking* $\rightarrow$ *sunburn*" with a score of 0.31. The above examples show that our system can provide reasonable insights from biomedical text.

6.8 Summary

In this chapter we build an end-to-end system for insight extraction on biomedical literature and develop novel similarity measurement modeling with deep neural networks to model interactions among representations of biomedical entities, the relation, and contexts. Our evaluation shows the interaction-based similarity measurement modeling leads to improved performance of causation/correlation relation extraction, and the system is able to extract insights with competitive human acceptance accuracy and its relation extraction component compares favorably against previous work.

It must be noted that this work was started when I was an research intern at Microsoft Research. This work is only possible with huge support from Navendu Jain, Kris Ganjam, Jessica Lundin and Ryen White. I thank them for everything.

Next in Chapter 7, we introduce our GPU-based approach for a high performance textual similarity identification task.

Chapter 7: High Performance Cross-Lingual Pattern Matching on GPUs

In previous chapters, we discuss how to improve the effectiveness of textual similarity measurement tasks with our interaction-based neural modeling, in this chapter we move on to a textual similarity identification task and instead focus on efficiency improvement. We introduce our work on high performance cross-lingual pattern matching task with GPUs for grammar extraction of statistical machine translation (SMT), and this cross-lingual similarity identification task is also a natural extension of our cross-lingual similarity measurement task as discussed previously in Chapter 4.7.

Our work [58, 60] consists of the following three major components:

1. Parallel phrase lookup with suffix array on GPUs (Section 7.3);
2. Contiguous pattern matching and extraction for phrase-based SMT on GPU (Section 7.4) [58];
3. Gappy pattern matching for hierarchical translation grammars of SMT on GPU (Section 7.5) [60].

In this chapter we focus on developing novel parallel algorithms that can run on GPUs efficiently and provide two take-home messages of this chapter.

1. Increase the utilization of GPU will lead to improved performance. Therefore de-

veloping GPU algorithms with higher level of massive parallelism is the key. We show our examples in following sections for the similarity identification task.

2. GPU supports tens of thousands of concurrently running processing threads, which all require data loading from the same GPU RAM. Unfortunately the transfer rate of current generation's GPU RAM speed is still not catching up with GPU processing speed improvement, therefore it is critical to design GPU algorithms with smaller memory footprint. We provide our example to develop compact data structures in following sections.

We start by introducing the task and the motivation of our work in Section 7.1 and Section 7.2, and move on to our GPU-based approach in following sections. Please refer to Chapter 2 for details of GPUs, datasets and other related background.

7.1 Motivation: Trade-offs Between Two SMT Architectures

There exists two architectures for traditional SMT systems. The first one has the *batch architecture* as shown in Figure 7.1, including the popular *Moses* [77] SMT system. The batch architecture firstly runs time-consuming grammar extraction over the parallel corpus, then stores the extracted grammar either in memory or in disk for further processing.

The other one has the *on-demand architecture* as shown in Figure 7.2, which, unlike the batch architecture, does not need to store the grammar beforehand. This on-demand architecture instead generates the indexed parallel texts in memory and extracts translation units on demand when they are needed to decode new input. This architecture has

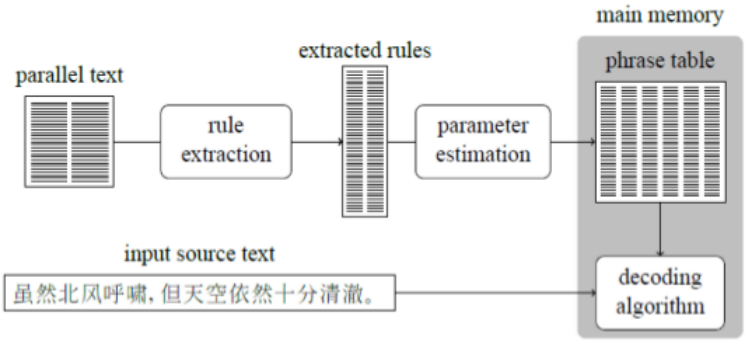


Figure 7.1: Batch Architecture of SMT System

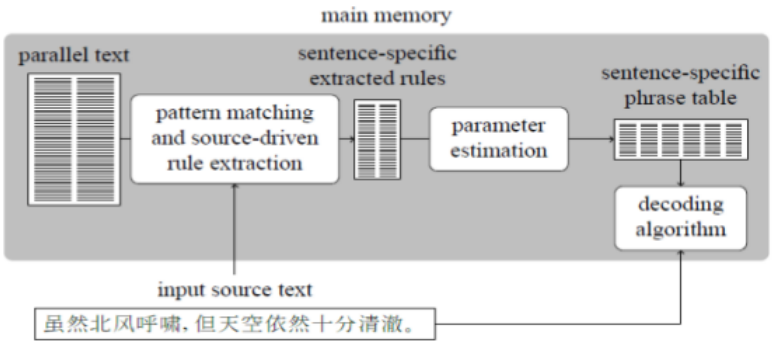


Figure 7.2: On-demand Architecture of SMT System

several advantages: It requires only a few gigabytes to represent a model that would otherwise require a terabyte [89], thus it scales to arbitrarily large models with very little computational pre-processing; for instance, Lopez [89] showed that it could represent a notional terabyte-sized translation model with ease. It can adapt incrementally to new training data [82], making it useful for interactive translation [50]. It supports rule extraction that is sensitive to the input sentence, enabling leave-one-out training [126] and the use of sentence similarity features [103]. However on the other hand, a limiting factor in its application is computational overhead at runtime, when translation units must be found very quickly. The two SMT architectures represent trade-offs between space and efficiency.

In this chapter, we focus on the on-demand architecture, which has been applied

successfully to phrase-based [20, 178], hierarchical [87, 88, 89], and syntax-based [32] statistical models. We show how to develop novel massively parallelized algorithms on GPUs to address its efficiency concerns in following sections.

7.2 Translation by Pattern Matching

Algorithm 7: Translation by Pattern Matching

```
1:  for each input sentence do
2:    for each phrase in the sentence do
3:      Find its occurrences in the source text (Section 7.3 and 7.5)
4:      for each occurrence do
5:        Extract any aligned target phrase (Section 7.4 and 7.5)
6:      end for
7:      for each extracted phrase pair do
8:        Compute feature values (Section 7.4 and 7.5)
9:      end for
10:   end for
11:   Decode as usual using the scored rules
12: end for
```

Lopez [89] provide a recipe of "translation by pattern matching" for the on-demand SMT architecture. We use it as a guide for our pattern matching work on GPUs (in Algorithm 7). We encounter a computational bottleneck in lines 2–9, since there are vast numbers of query phrases, matching occurrences, and extracted phrase pairs to process. In following sections we tackle each challenge in turn.

7.3 Massively Parallel Phrase Lookup with Suffix Array

A first key idea underlying the application of translation by pattern matching is to query for source translation units at runtime (Line 3 of Algorithm 7). Since the number of possible translation units may be quite large (for example, all substrings of a source sentence), query speed can become a performance bottleneck.

In this section, we introduce the first major component of our work on parallel phrase lookup, we show how to exploit the massive parallelism offered by GPUs to accelerate suffix array queries (Sec. 7.3.2). In our best configuration, we observe a speedup factor of nearly three orders of magnitude compared to a single-threaded CPU implementation (Sec. 7.4.5).

The contribution of current section is algorithmic: we specifically focus on a crucial aspect of translation by pattern matching—performance of suffix array queries—and thus our technique is agnostic with respect of particular translation models. Since our experiments verify that our GPU-based implementation produces *exactly* the same results as a CPU-based reference implementation, we can simply replace that component and reap significant performance advantages with no impact on translation quality. Although GPUs have previously been applied for DNA sequence matching using suffix trees [121] and suffix arrays [48], to our knowledge this is one of the first application of GPU acceleration techniques for statistical machine translation that we are aware of.

7.3.1 Suffix Array Efficiency Considerations in Translation

Since all prior work on translation by pattern matching was carried out on serial architectures, a number of techniques have been devised to reduce the number of queries and make them as efficient as possible in that setting.

1. Binary search bounds for longer substrings are initialized to the bounds of their longest prefix. Substrings are queried only if their longest prefix string was matched in the text.

2. In addition to conditioning on the longest prefix, Zhang and Vogel [178] and Lopez [87] condition on a successful query for the longest proper suffix.

3. Lopez [87] queries each unique substring of a sentence exactly once, regardless of how many times it appears in an input sentence.

4. Lopez [87] directly indexes one-word substrings with a small auxilliary array, so that their positions in the suffix array can be found in constant time. For longer substrings, this optimization reduces the $\log |T|$ term of query complexity to $\log(count(a))$, where a is the first word of the query string.

It is important to note that although these efficiency tricks are important to the serial algorithms that serve as our baselines, not all of them carry over cleanly to parallel architectures. In particular, optimizations (1), (2), and (3) introduce order dependencies between various queries. In our GPU implementation we disregard these optimizations in order avoid dependencies so that we can fully exploit large-scale parallelization op-

portunities. We note that we have not yet fully implemented (4), which is orthogonal to parallelization: this is left for future work.

7.3.2 GPU Parallelism on All-Substring Suffix Array Queries

As a strong first step towards removing the computational bottleneck outlined in previous section, we will focus specifically on the computational expense of line 3 of Algorithm 7: finding all occurrences of each source phrase in the original parallel text. This is a classic application of *string pattern matching*: given a short *query pattern*, find all instances of its occurrences in a much larger *text*. Solving it efficiently is particularly important, since it occurs many times: for an input sentence F of length $|F|$, each of its $O(|F|^2)$ substrings is a potential query pattern.

A natural approach to parallelism is to perform all substring queries in parallel. There are no dependencies between iterations of the loop beginning on line 2 of Algorithm 7, so for input sentence F, we can parallelize by searching for all $O(|F|^2)$ substrings simultaneously. Indeed, query-level parallelism is used in several popular bioinformatics systems [48, 121] that use GPU computing. However, naïve application of query-level parallelism would lead to a large number of wasted threads, since most long substrings of an input sentence will not be found in the text. Therefore, we employ a two-pass strategy: on the first pass, we simply compute, for each position i in the input sentence, the length j of the longest substring in F that appears in T. These computations are carried out simultaneously for every position i. During this pass, we also compute the suffix array bounds of the one-word substring $F[i]$, to be used as input on the second pass—a vari-

Initial Word	Longest Match	Substrings	Threads	
			1st pass	2nd pass
The	2	*The, The government*	2	2
government	2	*government, government puts*	2	2
puts	3	*puts, puts more, puts more tax*	2	4
more	2	*more, more tax*	2	2
tax	1	*tax*	2	0
on	0	–	2	0
its	0	–	2	0
citizens	0	–	2	0
Total Threads:			16	10

Table 7.1: Example of how large numbers of suffix array queries can be factored across two highly parallel passes on a GPU to perform all queries for a single input sentence.

ant of optimizations (1) and (4) discussed in Sec. 7.3.1. On the second pass, we search for all substrings $F[i,k]$ for all $k \in [i+1, i+j]$. These computations are carried out simultaneously for all substrings longer than one word.

Ever finer-grained parallelization is possible. Each query in a suffix array actually requires two binary searches: one each for the first and last match in $S(T)$. In a serial CPU architecture, these queries can be optimized to share some effort; however in the final analysis additional work is required to find both boundaries. On the other hand, the abundance of inexpensive threads on a GPU permit us to perform both queries simultaneously; by doing this we utilize more of the GPU and obtain further speedups. Therefore, we use two threads to perform each query in the first and second pass.

As a simple example, consider an input sentence "The government puts more tax on its citizens", and suppose that substrings "The government", "government puts", and "puts more tax" are found in the training text, while none of the words in "on its citizens" are found. Then the computation factors as shown in Table 7.1.

Even for this simple query, a large number of threads are spawned. Since all threads during a pass are carried out in parallel, and since each thread carries out a binary search which takes no more than $O(|Q| + \log |T|)$ time, the total time required to resolve all of the queries is very fast.

Even these fine-grained strategies still do not make optimal use of GPU parallelism, because the number of threads that the device can handle concurrently is typically far more than the number of queries. For example, our experimental GPU (Sec. 7.4.5) can manage up to 14,336 simultaneous threads. With this in mind, we can perform queries for multiple input sentences simultaneously. As we will show, the effect result of this is to amortize query costs across many sentences.

7.3.3 Experimental Setup

We tested our GPU-based suffix array implementation under the conditions in which it would be used for a large-scale Chinese-to-English machine translation task, in particular replicating the data conditions of [89]. Experiments were performed on two data sets. First, we used the source (Chinese) side of news articles collected from the Xinhua Agency. Second, we added source-side parallel text from the United Nations. And the input data consists of all sentences from the NIST 2002–2006 translation campaigns. In order to fully stress our GPU algorithm, we ran tests on batches of 2,000, 4,000, 8,000, 16,000, or 32,000 sentences. Please refer to Sec. 2.4.3 for dataset details.

To be precise the task is as follows: for every source sentence, we query the suffix array built from the source corpora to find *every* matching substring. In the case of com-

mon words, there could be many matches—we retrieve them all. Note that the focus of these experiments is on raw query speed for source corpora matches: in a real translation system, we would also need to store the target side of the parallel corpus and the alignments, from which phrases could be extracted. Such processing is not included in our experiments.

Our GPU device is Nvidia's Tesla C2050 (Fermi Generation), which has 448 CUDA cores with peak memory bandwidth 144GB/s. Our CPU is a Intel Xeon processor X5260 with 3.33GHz and 6MB L2 cache. Note that the GPU was released in early 2010 and it represents previous generation technology. Nvidia's Kepler GPUs boasts raw processing power in the 2 TFlops double precision range, which is approximately four times the GPU we have.

As a baseline, we compared against the publicly available implementation of the CPU-based algorithms described by Lopez [88] found in the pycdec [22] extension of the cdec machine translation system [37]. Note that we do not test the slower and more complex queries for hierarchical (gappy) patterns described by [87]; we test and measure only queries for normal substrings, which use all of the optimizations discussed in Sec. 7.3.1. Though they share no code, both our implementation and the baseline are written primarily in C.[1]

Our source corpora and test data are exactly the same as that presented by Lopez [89], and using the CPU implementation as reference enabled us to confirm that the results were identical. Thus, we are confident that our results represent a fair comparison under

[1]The [22] implementation is actually written in Cython, a language for building python applications with performance-critical components in C. In particular, all of the suffix array code that we instrumented for these experiments is compiled to C. The implementation is based on the original implementation of [88], but with many minor enhancements.

exactly the same experimental conditions, and are not attributed to misconfiguration or other flaws in experimental procedures. Note that the CPU implementation runs in a single thread, on the same machine that hosts the GPUs (described above). We have verified that both CPU and GPU implementations give *exactly* the same results.

7.3.4 Results

Table 7.2 shows the results of our experiment. The top portion of the table shows the total number of input sentences and the total number of queries generated (to query all matching substrings). The bottom two sections of the table shows results on just the Xinhua data and also the Xinhua plus UN data. Peformance of our GPU algorithm consists of three different components:

- Kernel: the time required for query computation on the GPU.
- Query to GPU: transfer time for input sentences from main memory to the GPU.
- Output from GPU: transfer time for results from GPU back to main memory.

As Table 7.2 shows, the transfer in and out adds moderate overhead to the overall computation.

We emphasize that figures reported (in milliseconds) represent the *total* amount of time necessary to process *all* sentences in the dataset. The results are averaged over five trials; the timing from run to run is sufficiently consistent that we omit the 95% confidence interval on running time here for simplicity of presentation. Because of the need to transfer data to and from the GPU and the tremendous amount of parallelism

Input Sentences	2,000	4,000	8,000	16,000	32,000
Total Queries	108,267	2,233,799	3,907,375	7,814,750	15,629,500
Xinhua					
Kernel (ms)	22.95	45.93	73.47	142.71	281.51
Query to GPU (ms)	0.40	0.60	0.93	1.55	2.63
Output From GPU (ms)	4.74	6.31	9.01	15.77	28.95
Total (ms)	**28.09**	**52.84**	**83.41**	**160.03**	**313.09**
Speedup	**662**×	**704**×	**892**×	**930**×	**951**×
Xinhua + UN					
Kernel (ms)	34.95	71.35	117.50	230.07	456.85
Query To GPU (ms)	0.39	0.60	0.92	1.53	2.63
Output From GPU (ms)	4.74	6.42	9.20	16.25	29.89
Total (ms)	**40.08**	**78.37**	**127.62**	**247.85**	**489.37**
Speedup	**529**×	**541**×	**664**×	**684**×	**693**×

Table 7.2: Query times for simultaneous processing of different numbers of input sentences.

offered by the device, it makes most sense to process sentences in batch mode, as we have done here.

Note that we observe super-linear speedup as the number of query sentences increase: this is simply because we do not saturate the GPU until the number of test query sentences reaches 32,000.

As a reference, the performance of the CPU version is 9.3 ms per sentence for the Xinhua corpus and 10.6 ms per sentence for the Xinhua plus UN corpus (averaged across our NIST test data which contains 8607 sentences). The CPU implementation is sequential (i.e., it processes one query after another in a single thread). We report speedup with respect to the CPU implementation for both corpora. Note utilizing all available processing power on the GPU, we achieve a throughput of 102207 sentences per second for the Xinhua corpus, which is a 951× speedup compared to the CPU throughput of 107.5 sentences per second.

Note that query times aren't substantially longer on the larger corpus. Even though the number of words is nearly quadrupled, query times increase by only 35% on average on our tests. This is mostly a consequence of that fact that the UN data is out of domain, so it doesn't greatly increase substring coverage for the test data, despite its size. Nevertheless, Lopez [89] demonstrated that including this data could yield significant improvements to translation performance.

When considering these results, an astute reader might note that we are comparing performance of a single-threaded algorithm with a fully-saturated GPU. A more fair comparison would be to compare against a CPU-based algorithm that takes advantage of all available cores. Unfortunately, a multi-threaded implementation of the baseline does not exist. In fact, as we note in Sec. 7.3.1, several of the optimizations used by the baseline implementation make it difficult to parallelize. Nevertheless, even if we make the conservative assumption that the baseline CPU implementation could be scaled linearly to all available cores (for example, eight cores)—improving its speed by the same factor—our unoptimized GPU implementation would still remain two orders of magnitude faster.

In the next section, we will show how to perform contiguous grammar extraction for phrase-based machine translation on GPU.

7.4 Contiguous Grammar Extraction for Phrase-based SMT

Algorithm 8: Contiguous Grammar Extraction for Phrase-based SMT

```
1:  for each input sentence do
2:    for each contiguous phrase in the sentence do
3:      Find its occurrences in the source text
4:      for each occurrence do
5:        Extract any aligned contiguous target phrase
6:      end for
7:      for each extracted phrase pair do
8:        Compute feature values
9:      end for
10:   end for
11:   Decode as usual using the scored rules
12: end for
```

We now focus on phrase-based statistical machine translation where contiguous grammar extraction is required. For easier illustration purpose we show the "translation by pattern matching" algorithm again, where there is substantial computational cost in searching a parallel corpus for source phrases, extracting their translations, and scoring them on the fly, since the number of possible translation units may be quite large (for example, all substrings of a source sentence). We show how to exploit the massive parallelism offered by GPUs to eliminate the computational bottlenecks associated with "on the fly" extraction. Since we have already described contiguous phrase lookup with suffix array on GPUs previously in Section 7.3 (Line 3 of Algorithm 8), in this section the following novel contribution is presented: We propose novel data structures and parallel algorithms for contiguous phrase extraction (Sec. 7.5.2, Line 5 of Algorithm 8 in red) and scoring (Sec. 7.5.3, Line 8 of Algorithm 8 in red) that are amenable to GPU parallelization.

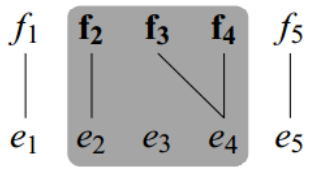


Figure 7.3: Source phrase $f_2 f_3 f_4$ and target phrase $e_2 e_3 e_4$ are extracted as a consistent pair, since the back-projection is contained within the original source span.

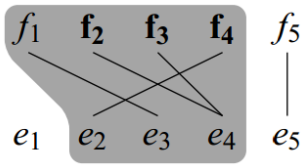


Figure 7.4: Source phrase $f_2 f_3 f_4$ and target phrase $e_2 e_3 e_4$ should not be extracted, since the back-projection is *not* contained within the original source span.

The resulting implementation achieves at least an order of magnitude higher throughput than a state-of-the-art single-threaded CPU implementation (Sec. 7.4.5).

7.4.1 Extracting Aligned Target Phrases

The problem at line 5 of Algorithm 8 is to extract the target phrase aligned to each matching source phrase instance. Efficiency is crucial since some source phrases occur hundreds of thousands of times.

Phrase extraction from word alignments typically uses the consistency check of [99]. A *consistent* phrase is one for which no words inside the phrase pair are aligned to words outside the phrase pair. Usually, consistent pairs are computed offline via dynamic programming over the alignment grid, from which we extract all consistent phrase pairs up to a heuristic bound on phrase length.

The online extraction algorithm of Lopez [88] checks for consistent phrases in a different manner. Rather than finding all consistent phrase pairs in a sentence, the algo-

rithm asks: given a specific source phrase, is there a consistent phrase pair of which it is one side? To answer this, it first computes the projection of the source phrase in the target sentence: the minimum span containing all words that are aligned to any word of the source span. It then computes the projection of the target span back into the source; if this back-projection is contained within the original source span, the phrase pair is consistent, and the target span is extracted as the translation of the source. Figure 7.3 shows a "good" pair for source phrase $f_2 f_3 f_4$, since the back-projection is contained within the original source span, whereas Figure 7.4 shows a "bad" pair for source phrase $f_2 f_3 f_4$ since the back-projection is *not* contained within the original source span.

7.4.2 Sampling Consistent Phrases

Regardless of how efficient the extraction of a single target phrase is made, the fact remains that there are many phrases to extract. For example, in our Chinese Xinhua dataset (see Sec. 7.4.5), from 8,000 input query sentences, about 20 million source substrings can be extracted. The standard solution to this problem is to sample a set of occurrences of each source phrase, and only extract translations for those occurrences [20, 178]. As a practical matter, this can be done by sampling at uniform intervals from the matching span of a suffix array. Lopez [88] reports a sample size of 300; for phrases occurring fewer than 300 times, all translations are extracted.

7.4.3 GPU Implementation with Compact Data Structures

We present novel data structures and an algorithm for efficient phrase extraction, which together are amenable to massive parallelization on GPUs. The basic insight is to pre-compute data structures for the source-to-target alignment projection and back-projection procedure described by Lopez [88] for checking consistent alignments.

Let us consider a single matching substring (from the output of the suffix array queries), span $[i, j]$ in the source text T. For each k, we need to know the leftmost and rightmost positions that it aligns to in the target T'. For this purpose we can define the target span $[i', j']$, along with leftmost and rightmost arrays L and R as follows:

$$i' := \min_{k \in [i,j]} L(k)$$

$$j' := \max_{k \in [i,j]} R(k)$$

The arrays L and R are each of length $|T|$, indexed by absolute corpus position. Each array element contains the leftmost and rightmost extents of the source-to-target alignments (in the target), respectively. Note that in order to save space, the values stored in the arrays are sentence-relative positions (e.g., token count from the beginning of each sentence), so that we only need one byte per array entry. Thus, i' and j' are sentence-relative positions (in the target).

Similarly, for the back-projection, we use two arrays L' and R' on the target side (length $|T'|$) to keep track of the leftmost and rightmost positions that k' in the target

training text align to, as below:

$$i'' := \min_{k' \in [s'+i', s'+j']} L'(k')$$

$$j'' := \max_{k' \in [s'+i', s'+j']} R'(k')$$

The arrays L' and R' are indexed by absolute corpus positions, but their contents are sentence relative positions (on the source side). To index the arrays L' and R', we also need to obtain the corresponding target sentence start position s'. Note that the back-projected span $[i'', j'']$ may or may not be the same as the original span $[i, j]$. In fact, this is exactly what we must check for to ensure a consistent alignment.

The suffix array gives us i, which is an absolute corpus position, but we need to know the sentence-relative position, since the spans computed by R, L, R', L' are all sentence relative. To solve this, we introduce an array P (length $|T|$) that gives the relative sentence position of each source word.

We then pack the three source side arrays (R, L, and P) into a single RLP array of 32-bit integers (note that we are actually wasting one byte per array element). Finally, since the end-of-sentence special token is not used in any of R, L, or P, its position in RLP can be used to store an index to the start of the corresponding target sentence in the target array T'. Now, given a source phrase spanning $[i, j]$ (recall, these are absolute corpus positions), our phrase extraction algorithm is as follows:

where s is the source sentence start position of a given source phrase and s' is the target sentence start position. If the back-projected spans match the original spans, the phrase pair $T[i, j]$ and $T'[s' + i', s' + j']$ is extracted.

Algorithm 9: Efficient Phrase Extraction Algorithm

1: **for each** source span $[i, j]$ **do**
2: Compute $[i', j']$
3: $s := i - P[i] - 1$
4: $s' := RLP[s]$
5: $i'' := \min_{k' \in [s'+i', s'+j']} L'(k')$
6: $j'' := \max_{k' \in [s'+i', s'+j']} R'(k')$
7: If $i - s = i''$ and $j - s = j''$ then
8: Extract $T[i, j]$ with $T'[s' + i', s' + j']$
9: **end for**

In total, the data structures RLP, R', and L' require $4|T| + 2|T'|$ bytes. Not only is this phrase extraction algorithm fast—requiring only a few indirect array references—the space requirements for the auxiliary data structures are quite modest.

Given sufficient resources, we would ideally parallelize the phrase table creation process for each occurrence of the matched source substring. However, the typical number of source substring matches for an input sentence is even larger than the number of threads available on GPUs, so this strategy does not make sense due to context switching overhead. Instead, GPU thread blocks (groups of 512 threads) are used to process each source substring. This means that for substrings with large numbers of matches, one thread in the GPU block would process multiple occurrences. This strategy is widely used, and according to GPU programming best practices from NVIDIA, allocating more work to a single thread maintains high GPU utilization and reduces the cost of context switches.

7.4.4 Computing Every Feature

Finally, we arrive at line 7 in Algorithm 8, where we must compute feature values for each extracted phrase pair. Following the implementation of grammar extraction used

in cdec [88], we compute several widely-used features:

1. Pair count feature, $c(e, f)$.
2. The joint probability of all target-to-source phrase translation probabilities, $p(e|f) = c(e, f)/c(f)$, where e is target phrase, f is the source phrase.
3. The logarithm of the target-to-source lexical weighting feature.
4. The logarithm of the source-to-target lexical weighting feature.
5. The coherence probability, defined as the ratio between the number of successful extractions of a source phrase to the total count of the source phrase in the suffix array.

The output of our phrase extraction is a large collection of phrase pairs. To extract the above features, aggregate statistics need to be computed over phrase pairs. To make the solution both compact and efficient, we first sort the unordered collection of phrases from the GPU into an array, then the aggregate statistics can be obtained in a single pass over the array, since identical phrase pairs are now grouped together.

7.4.5 Experimental Setup

We follow the same experimental setup in Section 7.3.3, where we use Xinhua data (around one million sentences) and Xinhua+UN data (around four million sentences) as the parallel corpus, and NIST 2002–2006 data as the input query set for evaluation. We also use NVIDIA's Tesla C2050 GPU and 3.33 GHz dual-core Intel Xeon X5260 processor. Our baseline is the same CPU-based grammar extractor (written primarily in C) from the cdec machine translation system [37] and we run the CPU baseline in a

Input Sentences Number of Words		2,000 57,868	4,000 117,854	6,000 161,883	8,000 214,246	16,000 428,492
Xinhua + UN						
With Sampling (s_{300})	GPU (words/second)	2021	2558	2933	3439	6737
	CPU (words/second)	157				
	Speedup	**13**×	**16**×	**19**×	**22**×	**43**×
No Sampling (s_{∞})	GPU (words/second)	500.5	770.1	984.6	1243.8	2472.3
	CPU (words/second)	0.23				
	Speedup	**2194**×	**3375**×	**4315**×	**5451**×	**10836**×

Table 7.3: Comparing the GPU and CPU implementations for phrase extraction on two different corpora. Throughput is measured in words per second under different test set sizes; the 95% confidence intervals across five trials are given in parentheses, along with relative speedups comparing the two implementations.

single thread. Please refer to Section 7.3.3 for more experimental setup details. Unlike Section 7.3.3, we did not run tests on 32,000 sentences due to the GPU memory limitation.

We timed our GPU implementation as follows: from the loading of query sentences, extractions of substrings and grammar rules, until all grammars for all sentences are generated in memory. Timing does not include offline preparations such as the construction of the suffix array on source texts and the I/O costs for writing the per-sentence grammar files to disk. This timing procedure is exactly the same for the CPU baseline. We are confident that our results represent a fair comparison between the GPU and CPU.

7.4.6 Results

Table 7.8 shows performance results comparing our GPU implementation against the reference CPU implementation for phrase extraction. In one experimental condition, the sampling parameter for frequently-matching phrases is set to 300, per [88], denoted s_{300}. The experimental condition without sampling is denoted s_{∞}. Following standard settings, the maximum length of the source phrase is set to 5 and the maximum length of

the target phrase is set to 15 (same for both GPU and CPU implementations). The table is divided into two sections: the top shows results on the Xinhua data, and the bottom on Xinhua + UN data. Columns report results for different numbers of input sentences. Performance is reported in terms of throughput: the number of processed words per second on average (i.e., total time divided by the batch size in words). The results are averaged over five trials, with 95% confidence intervals shown in parentheses. Note that as the batch size increases, we achieve higher throughput on the GPU since we are better saturating its full processing power. In contrast, performance is constant on the CPU regardless of the number of sentences processed.

The CPU throughput on the Xinhua data is 1.13 words per second without sampling and 200 words per second with sampling. On 16,000 test sentences, we have mostly saturated the GPU's processing power, and observe a 7217× speedup over the CPU implementation without sampling and 62× speedup with sampling. On the larger (Xinhua + UN) corpus, we observe 43× and 10836× speedup with sampling and no sampling, respectively.

Interestingly, a run without sampling on the GPU is still substantially faster than a run with sampling on the CPU. On the Xinhua corpus, we observe speedups ranging from nine times to forty times, as shown in Table 7.4. Without sampling, we are able to extract up to twice as many phrases.

In previous CPU implementations of on-the-fly phrase extraction, restrictions were placed on the maximum length of the source and target phrases due to computational constraints (in addition to sampling). Given the massive parallelism afforded by the GPU, might we be able to lift these restrictions and construct the complete phrase table? To

# Sent.	2000	4000	6000	8000	16000
Speedup	9.6×	14.3×	17.5×	20.9×	40.9×
Phrases	2.1×	1.8×	1.7×	1.6×	1.6×

Table 7.4: Comparing no sampling on the GPU with sampling on the CPU in terms of performance improvements (GPU over CPU) and increases in the number of phrase pairs extracted (GPU over CPU).

answer this question, we performed an experiment without sampling and without any restrictions on the length of the extracted phrases. The complete phrase table contained about 0.5% more distinct pairs, with negligible impact on performance.

When considering these results, an astute reader might note that we are comparing performance of a single-threaded implementation with a fully-saturated GPU. To address this concern, we conducted an experiment using a multi-threaded version of the CPU reference implementation to take full advantage of multiple cores on the CPU (by specifying the `-j` option in cdec); we experimented with up to four threads to fully saturate the dual-core CPU. In terms of throughput, the CPU implementation scales linearly, i.e., running on four threads achieves roughly 4× throughput. Note that the CPU and GPU implementations take advantage of parallelism in completely different ways: cdec can be characterized as embarrassingly parallel, with different threads processing each complete sentence in isolation, whereas our GPU implementation achieves intra-sentential parallelism by exploiting many threads to concurrently process each sentence. In terms of absolute performance figures, even with the 4× throughput improvement from fully saturating the CPU, our GPU implementation remains faster by a wide margin. Note that neither our GPU nor CPU represents state-of-the-art hardware, and we would expect the performance advantage of GPUs to be even greater with latest generation hardware, since

Phrase Extraction	I/O	Decoding	
GPU: 11.0	3.7	1 thread	55.7
		2 threads	35.3
CPU: 166.5		3 threads	31.5
		4 threads	26.2

Table 7.5: End-to-end machine translation performance: time to process the NIST05 test set in seconds, broken down in terms of the three processing stages.

the number of available threads on a GPU is increasing faster than the number of threads available on a CPU.

Since phrase extraction is only one part of an end-to-end machine translation system, it makes sense to examine the overall performance of the entire translation pipeline. For this experiment, we used our GPU implementation for phrase extraction, serialized the grammar files to disk, and used cdec for decoding (on the CPU). The comparison condition used cdec for all three stages. We used standard phrase length constraints (5 on source side, 15 on target side) with sampling of frequent phrases. Finally, we replicated the data conditions in [88], where our source corpora was the Xinhua data set and our development/test sets were the NIST03/NIST05 data; the NIST05 test set contains 1,082 sentences.

Performance results for end-to-end translation are shown in Table 7.5, broken down in terms of total amount of time for each of the processing stages for the entire test set under different conditions. In the decoding stage, we varied the number of CPU threads (note here we do not observe linear speedup). In terms of end-to-end results, complete translation of the test set takes 41 seconds with the GPU for phrase extraction and CPU for decoding, compared to 196 seconds using the CPU for both (with four decoding threads in both cases). This represents a speedup of 4.8×, which suggests that even selective

optimizations of individual components in the SMT pipeline using GPUs can make a substantial difference in overall performance.

In next section, we show details of our work on gappy pattern matching for hierarchical phrase-based machine translation on GPUs, which is an extension of contiguous pattern matching for phrase-based SMT as described in this section.

7.5 Gappy Pattern Matching for Hierarchical Phrase-Based SMT

Our previous sections demonstrated orders of magnitude speedup in contiguous pattern matching and extraction for phrase-based SMT models. However, some popular *hierarchical* phrase-based SMT models use "gappy" phrases [25, 46, 125], and the algorithm of our previous work does not work for these models. We therefore need pattern matching and phrase extraction that is able to handle variable-length gaps [87].

In this section we present a novel GPU algorithm for on-demand extraction of hierarchical translation models based on matching and extracting **gappy phrases**. Our experiments examine both grammar extraction and end-to-end translation, comparing quality, speed, and memory use.

We then compare against the GPU system for our own previous GPU work on phrase-based translation and *cdec* (a state-of-the-art CPU system for hierarchical translation [37]). Our system outperforms the former on translation quality by 2.3 BLEU (replicating previously-known results) and outperforms the latter on speed, improving grammar extraction throughput by at least an order of magnitude on large batches of sentences while maintaining the same level of translation quality. Our contribution is to show, complete with an open-source implementation, how GPUs can vastly increase the speed of hierarchical grammar extraction, particularly for high-throughput SMT applications.

Our pattern matching algorithm is organized around two general design principles: brute force scans and fine-grained parallelism. Brute force array scans avoid warp divergence since they access data in regular patterns. Rather than parallelize larger algorithms that use these scans as subroutines, we parallelize the scans themselves in a fine-grained

manner to obtain high throughput.

The relatively small size of the GPU memory also affects design decisions. Data transfer between the GPU and the CPU has high latency, so we want to avoid shuffling data as much as possible. To accomplish this, we must fit all our data structures into the 5 GB memory available on our particular GPU. As we will show, this requires some tradeoffs in addition to careful design of algorithms and associated data structures.

7.5.1 Finding Every Gappy Phrase

i	$T[i]$	$S_T[i]$	suffix $S_T[i]$: $T[S_T[i]]...T[\|T\|]$
1	#	5	and it mars him # it sets him on and it takes him off # $
2	it	14	and it takes him off # $
3	makes	4	him and it mars him # it sets him on and it takes him off # $
4	him	17	him off # $
5	and	12	him on and it takes him off # $
6	it	8	him # it sets him on and it takes him off # $
7	mars	2	it makes him and it mars him # it sets him on and it takes him off # $
8	him	6	it mars him # it sets him on and it takes him off # $
9	#	10	it sets him on and it takes him off # $
10	it	15	it takes him off # $
11	sets	3	makes him and it mars him # it sets him on and it takes him off # $
12	him	7	mars him # it sets him on and it takes him off # $
13	on	18	off # $
14	and	13	on and it takes him off # $
15	it	11	sets him on and it takes him off # $
16	takes	16	takes him off # $
17	him	1	# it makes him and it mars him # it sets him on and it takes him off # $
18	off	9	# it sets him on and it takes him off # $
19	#	19	# $
20	$	20	$

Figure 7.5: Example text T and suffix array S_T with corresponding suffixes.

Line 3 of Algorithm 7 searches T for all occurrences of all phrases in Q. We call each phrase a pattern, and our goal is to find all phrases of T that match each pattern, i.e., the problem of pattern matching. Our *hierarchical* translation model permits phrases with at most two gaps, so Q is a source of $\mathcal{O}(|Q|^6)$ patterns, since there are up to six possible

subphrase boundaries: $\star uv$, $uv\star$, $u \star v$, $u \star v\star$, $\star u \star v$ and $u \star v \star w$.

Passes 1-2: Finding contiguous patterns

To find contiguous phrases (patterns without gaps), we use the algorithm as described previously in Section 7.3. It requires a suffix array [92] computed from T.

Previously in Section 2.4.2 we have introduced the concept of suffix array. We show the same suffix array example here for easier illustration in Figure 7.5, which indexes two sentences: *it makes him and it mars him*, and *it sets him on and it takes him off*. Let $|T|$ denote the total number of tokens of the sentences, the ith suffix of a 1-indexed text T is the substring $T[i]...T[|T|]$ starting at position i and continuing to the end of T. The suffix array S_T is a permutation of the integers $1, ..., |T|$ ordered by a lexicographic sort of the corresponding suffixes. Please refer to Section 2.4.2 for more details on this suffix array example.

Given S_T, finding a pattern P is simply a matter of binary search for the pair of integers (ℓ, h) such that for all i from ℓ to h, P is a prefix of the $S_T[i]$th suffix of T. Thus, each integer $S_T[i]$ identifies a unique match of P. Therefore in our example of Figure 7.5, the pattern *it* returns $(7, 10)$, corresponding to matches at positions 2, 6, 10, and 15; while *him and it* returns $(3, 3)$, corresponding to a match at position 4. A longest common prefix (LCP) array enables us to find h or ℓ in $\mathcal{O}(|Q| + \log |T|)$ comparisons [92].

Every substring of Q is a contiguous pattern, but if we searched T for all of them, most searches would fail, wasting computation. Instead, our work previously described in Section 7.4 use two passes. The first computes, concurrently for every position i in $1, ..., |Q|$, the endpoint j of the longest substring $Q[i]...Q[j]$ that appears in T. It also

computes the suffix array range of the one-word substring $Q[i]$. Taking this range as input, for all k from i to j the second pass concurrently queries T for pattern $Q[i]...Q[k]$. This pass uses two concurrent threads per pattern—one to find the lowest index of the suffix array range, and one to find the highest index.

Passes 3-4: Finding one-gap patterns (New)[2]

Passes 1 and 2 find contiguous phrases, but we must also find phrases that contain gaps. We use the special symbol $\star$ to denote a variable-length gap. The set of one-gap patterns in Q thus consists of $Q[i]...Q[j]\star Q[i']...Q[j']$ for all i, j, i', and j' such that $i \leq j < i' - 1$ and $i' \leq j'$. When the position in Q is not important we use strings u, v, and w to denote contiguous patterns; for example, $u \star v$ denotes an arbitrary one-gap pattern. We call the contiguous strings u and v of $u \star v$ its subpatterns, e.g., *it* $\star$ *him* is a pattern with subpatterns *it* and *him*.

When we search for a gappy pattern, $\star$ can match any non-empty substring of T that does not contain $ or #. Such a match may not be uniquely identified by the index of its first word, so we specify it with a tuple of indices, one for the match of each subpattern. Pattern *it* $\star$ *him* has six matches in T: $(2, 4)$, $(2, 8)$, $(6, 8)$, $(10, 12)$, $(10, 17)$, and $(15, 17)$. Passes 3 and 4 search T for all one-gap patterns using the novel GPU algorithm described below.

A pattern $u \star v$ cannot match in T unless both u and v match in T, so we use the output of pass 1, which returns all (i, j) pairs such that $Q[i]...Q[j]$ matches in T. Concurrently for every such i and j, pass 3 enumerates all i' and j' such that $j < i' - 1$

[2]Note the word "(New)" in the title means the passes are unique for gappy pattern matching work in this section, and are not used previously in Section 7.4.

and $Q[i']...Q[j']$ matches in T, returning each pattern $Q[i]...Q[j] \star Q[i']...Q[j']$. Pass 3 then sorts and deduplicates the combined results of all threads to obtain a set of unique patterns. These operations are carried out on the GPU using the algorithms of [66].

Pass 4 searches for matches of each pattern identified by pass 3. We first illustrate with *it* $\star$ *him*. Pass 2 associates *it* with suffix array range $(7, 10)$. A linear scan of S_T in this range reveals that *it* matches at positions 2, 6, 10, and 15 in T. Likewise, *him* maps to range $(3, 6)$ of S_T and matches at 4, 17, 12, and 8 in T. Concurrently for each match of the less frequent subpattern, we scan T to find matches of the other subpattern until reaching a sentence boundary or the maximum phrase length, an idea we borrow from the CPU implementation of [9]. In our example, both *it* and *him* occur an equal number of times, so we arbitrarily choose one—suppose we choose *it*. We assign each of positions 2, 6, 10, and 15 to a separate thread. The thread assigned position 2 scans T for matches of *him* until the end of sentence at position 9, finding matches $(2, 4)$ and $(2, 8)$.

As a second example, consider *it* $\star$ *and*. In this case, *it* has four matches, but *and* only two. So, we need only two threads, each scanning backwards from matches of *and*. Since most patterns are infrequent, allocating threads this way minimizes work. However, very large corpora contain one-gap patterns for which both subpatterns are frequent. We simply precompute all matches for these patterns and retrieve them at runtime, as in [87]. This precomputation is performed once given T and therefore it is a one-time cost.

Materializing every match of $u \star v$ would consume substantial memory, so we only emit those for which a translation of the substring matching $\star$ is extractable using the check in Sec. 7.5.2. The success of this check is a prerequisite for extracting the translation of $u \star v$ or any pattern containing it, so pruning in this way conserves GPU memory

without affecting the final grammar.

Passes 5-7: Finding two-gap patterns (New)

We next find all patterns with two gaps of the form $u \star v \star w$. The search is similar to passes 3 and 4. In pass 5, concurrently for every pattern $u \star v$ matched in pass 4, we enumerate the pattern $u \star v \star w$ for every w such that $u \star v \star w$ is a pattern in Q and w matched in pass 1. In pass 6, concurrently for every match (i, j) of $u \star v$ for every $u \star v \star w$ enumerated in pass 5, we scan T from position $j + |v| + 1$ for matches of w until we reach the end of sentence. As with the one-gap patterns, we apply the extraction check on the second $\star$ of the two-gap patterns $u \star v \star w$ to avoid needlessly materializing matches that will not yield translations.

7.5.2 Extracting Every Gappy Target Phrase

In line 5 of Algorithm 7 we must extract the aligned translation of every match of every pattern found in T. Efficiency is crucial since some patterns may occur hundreds of thousands of times.

We extract translations from word alignments using the consistency check of Och et al. [99]. A pair of substrings is consistent only if no word in either substring is aligned to any word outside the pair. For example, in Figure 7.6 the pair (*it sets him on, los excita*) is consistent. The pair (*him on and, los excita y*) is not, because *excita* also aligns to the words *it sets*. Only consistent pairs can be translations of each other in our model.

Given a specific source substring, our algorithm asks: is it part of a consistent pair? To answer this question, we first compute the minimum target substring to which all words

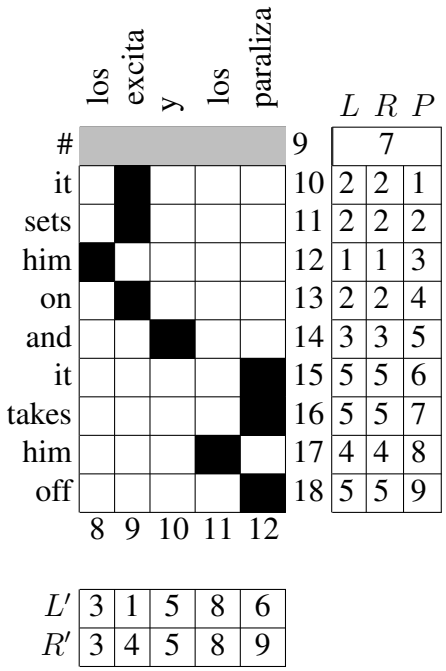


Figure 7.6: An example alignment and the corresponding L, R, P, L' and R' arrays.

in the substring align. We then compute the minimum substring to which all words of this candidate translation align. If this substring matches the input, the candidate translation is returned; otherwise, extraction fails. For example, *it sets him on* in range $(10, 13)$ aligns to *los excita* in range $(8, 9)$, which aligns back to $(10, 13)$. So this is a consistent pair. However, *him on and* in range $(12, 14)$ aligns to *los excita y* in range $(8, 10)$, which aligns back to *it sets him on and* at $(10, 14)$. So *him on and* is not part of a consistent pair and has no extractable translation.

To extract gappy translation units, we subtract consistent pairs from other consistent pairs [25]. For example, (*him, los*) is a consistent pair. Subtracting it from (*it sets him on, los excita*) yields the translation unit (*it sets* $\star$ *on,* $\star$ *excita*).

Our basic building block is the EXTRACT function previously described in Section 7.4.3, which performs the above check using byte arrays denoted L, R, P, L', and R'

(Figure 7.6) to identify extractable target phrases in target text T'. When $T[i]$ is a word, $L[i]$ and $R[i]$ store the sentence-relative positions of the leftmost and rightmost words it aligns to in T', and $P[i]$ stores $T[i]$'s sentence-relative position. When $T[i]$ is a sentence boundary, the concatenation of bytes $L[i]$, $R[i]$, and $P[i]$, denoted $LRP[i]$, stores the position of the corresponding sentence in T'. Bytes $L'[i']$ and $R'[i']$ store the sentence-relative positions of the leftmost and rightmost words $T'[i']$ aligns to.

We first calculate the start position p of the source sentence containing $T[i]...T[j]$, and the start position p' of the corresponding target sentence:

$$p = i - P[i]$$

$$p' = LRP[p]$$

We then find target indices i' and j' for the candidate translation $T'[i']...T'[j']$:

$$i' = p' + \min_{k \in i,...,j} L[k]$$

$$j' = p' + \max_{k \in i,...,j} R[k]$$

We can similarly find the translation $T[i'']...T[j'']$ of $T'[i']...T'[j']$:

$$i'' = p + \min_{k' \in i',...,j'} L'[k']$$

$$j'' = p + \max_{k' \in i',...,j'} R'[k']$$

If $i = i''$ and $j = j''$, EXTRACT(i, j) returns (i', j'), the position of $T[i]...T[j]$'s trans-

```
1: k ← i
2: while T[k − 1] ≠ # do
3:    k ← k − 1
4:    if EXTRACT(k, i + |u| − 1) succeeds then
5:       (i′, j′) ← EXTRACT(k, i + |u| − 1)
6:       if EXTRACT(k, i − 1) succeeds then
7:          (p′, q′) ← EXTRACT(k, i − 1)
8:          return T′[i′]...T′[p′] ⋆ T′[q′]...T′[j′]
9:       end if
10:   end if
11: end while
12: if T[k − 1] = # then
13:    return failure
14: end if
```

lation. Otherwise the function signals that there is no extractable translation. Given this function, extraction proceeds in three passes.

Pass 8: Extracting contiguous patterns

Each match of pattern u is assigned to a concurrent thread. The thread receiving the match at position i returns the pair consisting of u and its translation according to EXTRACT$(i, i+|u|-1)$, if any. It also returns translations for patterns in which u is the only contiguous subpattern: $\star\ u$, $u\ \star$, and $\star\ u\ \star$. We extract translations for these patterns even if u has no translation itself. To see why, suppose that we reverse the translation direction of our example. In Figure 7.6, *excita* is not part of a consistent pair, but both $\star$ *excita* and $\star$ *excita* $\star$ are.

Consider $\star\ u$. Since $\star$ matches any substring in T without boundary symbols, the leftmost position of $\star\ u$ is not fixed. So, we seek the smallest match with an extractable translation, returning its translation with the following algorithm.

The case of $u\ \star$ is symmetric.

We extend this algorithm to handle $\star\ u\ \star$. The extension considers increasingly

distant pairs (k, ℓ) for which $\star\ u\ \star$ matches $T[k]...T[\ell]$, until it either finds an extractable translation, or it encounters both sentence boundaries and fails.

Pass 9: Extracting one-gap patterns (New)

In this pass, each match of pattern $u\ \star\ v$ is assigned to a thread. The thread receiving match (i, j) attempts to assign i', j', p' and q' as follows.

$$(i', j') = \textsc{Extract}(i, j + |v| - 1)$$

$$(p', q') = \textsc{Extract}(i + |u|, j - 1)$$

If both calls succeed, we subtract to obtain the translation $T'[i']...T'[p'] \star T'[q']...T'[j']$. We extract translations for $\star\ u\ \star\ v$ and $u\ \star\ v\ \star$, using the same algorithm as in pass 7.

Pass 10: Extracting two-gap patterns (New)

In this pass, we extract a translation for each match of $u\ \star\ v\ \star\ w$ concurrently, using a straightforward extension of the subtraction algorithm in pass 8. Since we only need patterns with up to two gaps, we do not consider other patterns.

To optimize GPU performance, for the passes above, we assign all matches of a gappy pattern to the same thread block. This allows threads in the same thread block to share the same data during initialization, therefore improving memory access coherence.

7.5.3 Computing Every Feature

Our feature computation follows Section 7.5.3. Though differences remain, for example, we use log operation to compute most feature values. We use α and β to denote

arbitrary strings of words and $\star$ symbols, and our input is a multiset of (α, β) pairs collected by passes 7-9, which we denote by Π. We compute the following features for each unique (α, β) pair.

Log-count features. We need two aggregate statistics: The count of (α, β) in Π, and the count of all pairs in Π for which α is the source pattern. We then compute the two features as $\log(1 + \text{count}(\alpha, \beta))$ and $\log(1 + \text{count}(\alpha))$. Unlike Section 7.5.3 we do not use $\text{count}(\alpha)$.

Translation log-probability. Given the aggregate counts above, this feature is $\log \frac{\text{count}(\alpha,\beta)}{\text{count}(\alpha)}$.

Singleton indicators. We compute two features, to indicate whether (α, β) occurs only once, i.e., $\text{count}(\alpha, \beta) = 1$, and whether α occurs only once, i.e., $\text{count}(\alpha) = 1$. We do not use this feature previously in Section 7.5.3.

Lexical weight. Consider word pairs a, b with $a \in \alpha$, $b \in \beta$, and neither a nor b are $\star$. Given a global word translation probability table $p(a|b)$, which is externally computed from the word alignments directly, the feature is $\sum_{a \in \alpha} \max_{b \in \beta} \log p(a|b)$. This feature is similar as the ones in Section 7.5.3.

Since Π is the result of parallel computation, we must sort it. We can then compute aggregate statistics by keeping running totals in a scan of the sorted multiset. With many instantiated patterns, we would quickly exhaust GPU memory, so this sort is performed on the CPU. We compute the log-count features, translation log-probability, and singleton indicators this way. However, the lexical weight feature is a function only of the aligned translation pair itself and corresponding word-level translation possibilities calculated externally from word alignment. Thus, the computation of this feature can be parallelized

on the GPU. So, we have multiple feature extraction passes based on the number of gaps:

Pass 11 (CPU): One-gap features (New).

Pass 12 (CPU): Two-gap features (New).

Pass 13 (CPU): Contiguous features.

Pass 14 (GPU): Lexical weight feature (New).

7.5.4 Sampling

In serial implementations of on-demand extraction, very frequent patterns are a major computational bottleneck [20, 89]. Thus, for patterns occurring more than n times, for some fixed n (typically between 100 and 1000), these implementations deterministically sample n matches, and only extract translations of these matches. To compare with these implementations, we also implement an optional sampling step. Though we use the same sampling rate, the samples themselves are not the same, since our extraction checks in passes 4 and 6 alter the set of matches that are actually enumerated, thus sampled from. The CPU algorithms do not use this check.

7.5.5 Experimental Setup

We tested our algorithms in an end-to-end Chinese-English translation task using data conditions similar to those of Lopez [89] and He et al. [58] as described previously in Section 7.4. Our implementation of hierarchical grammar extraction on the GPU, as detailed in the previous section, is written in C, using CUDA library v5.5 and GCC v4.8, compiled with the -O3 optimization flag. Our code is open source and available for re-

searchers to download and try out.[3]

Hardware. We used NVIDIA's Tesla K20c GPU (Kepler Generation), which has 2496 CUDA cores and 5 GB memory, with a peak memory bandwidth of 208 GB/s. The server hosting the GPU has two Intel Xeon E5-2690 CPUs, each with eight cores at 2.90 GHz (a total of 16 physical cores; 32 logical cores with hyperthreading). Both were released in 2012 and represent comparable generation hardware technology. All GPU and CPU experiments were conducted on the same machine, which runs Red Hat Enterprise Linux (RHEL) 6.

Training Data. As in Section 7.3.3, we use the same Xinhua data (around one million sentences) and Xinhua+UN data (around four million sentences) as the training corpus.

Test Data. For performance evaluations, we ran tests on sentence batches of varying sizes: 100, 500, 1k, 2k, 4k, 6k, 8k, 16k and 32k. As in Section 7.3.3, these sentences are drawn from the NIST 2002–2008 MT evaluations (on average 27 words each). But unlike Section 7.3.3, our test data is also from the Chinese side of the Hong Kong Parallel Text (LDC2004T08) when the NIST data are smaller than the target batch size. Large batch sizes are necessary to saturate the processing power of the GPU. The size of the complete batch of 32k test sentences is 4892 KB.

Baselines. We compared our GPU implementation for on-demand extraction of hierarchical grammars against the corresponding CPU implementation by Lopez [88] found in cdec [37], which is same as described in Section 7.3.3. We also compared our GPU algorithms against Moses [77], representing a standard phrase-based SMT baseline. Phrase tables generated by Moses are essentially the same as the GPU implementation of on-

[3] http://hohocode.github.io/cgx/

demand extraction for phrase-based translation by our work [58] as described earlier in Section 7.4.

7.5.6 Results

We show the speed performance of our hierarchical grammar extraction component in following sections.

Results on Translation Quality

We first verified that our GPU implementation achieves the same translation quality as the corresponding CPU baseline. This is accomplished by comparing system output against the baseline systems, training on Xinhua, tuning on NIST03, and testing on NIST05. In all cases, we used MIRA [26] to tune parameters. We ran experiments three times and report the average as recommended by Clark et al. [28]. Hierarchical grammars were extracted with sampling at a rate of 300; we also bound source patterns at a length of 5 and matches at a length of 15. For Moses we used default parameters.

Our BLEU scores, shown in Table 7.6, replicate well-known results where hierarchical models outperform pure phrase-based models on this task. The difference in quality is partly because the phrase-based baseline system does not use lexicalized reordering, which provides similar improvements to hierarchical translation [89]. Such lexicalized reordering models cannot be produced by the GPU-based system in the previous section of He et al. [58]. This establishes a clear translation quality improvement between our work in this section and that of the previous section.

System	BLEU
Moses phrase-based baseline	31.11
Hierarchical with online CPU extraction	33.37
Hierarchical with online GPU extraction	33.46

Table 7.6: Comparison of translation quality. The hierarchical system is cdec. Online CPU extraction is the baseline, part of the standard cdec package. Online GPU extraction is this work.

We see that the BLEU score obtained by our GPU implementation of hierarchical grammar extraction is nearly identical to cdec's, evidence that our implementation is correct. The minor differences in score are due to non-determinism in tuning and the difference in sampling algorithms (Sec. 7.5.4).

Results on Extraction Speed

Next, we focus on the performance of the hierarchical grammar extraction component, comparing the CPU and GPU implementations. For both implementations, our timings include preparation of queries, pattern matching, extraction, and feature computation. For the GPU implementation, we include the time required to move data to and from the GPU. We do not include time for construction of static data structures (suffix arrays and indexes) and initial loading of the parallel corpus with alignment data, as those represent one-time costs. Note that the CPU implementation includes indexes for frequent patterns in the form $u \star v$ and $u \star v \star w$, while our GPU implementation indexes only the former.

We compared performance varying the number of queries, and following Lopez [88], we compared sampling at a rate of 300 against runs without sampling. Our primary evaluation metric is throughput: the average number of processed words per second (i.e.,

threads	+Sampling		−Sampling	
	X	X+U	X	X+U
1	12.7	4.8	0.32	0.05
16	190.7	65.3	4.81	0.70
32	248.1	76.0	6.23	0.89

Table 7.7: CPU extraction performance (throughput in words/second) using different numbers of threads under different data conditions (X: Xinhua, X+U: Xinhua+UN), with and without sampling.

batch size in words divided by total time).

We first establish throughput baselines on the CPU, shown in Table 7.7. Experiments used different numbers of threads under different data conditions (Xinhua or Xinhua + UN), with and without sampling. Our server has a total of 16 physical cores, but supports 32 logical cores via hyperthreading. We obtained the CPU sampling results by running cdec over 16k query sentences. For the non-sampling runs, since the throughput is so low, we measured performance over 2.6k sentences for Xinhua and 500 sentences for Xinhua + UN. We see that throughput scaling is slightly less than linear: with sampling, using 16 threads increases throughput by 15× on the Xinhua data (compared to a single thread) and 13.6× on Xinhua + UN data. Going from 16 to 32 threads further increase throughput by 15%-30% and saturates the processing capacity of our server. The 32 thread condition provides a fair baseline for comparing the performance of the GPU implementation.

Table 7.8 shows GPU hierarchical grammar extraction performance in terms of throughput (words/second); these results are averaged over three trials. We varied the number of query sentences, and in each case, also report the speedup with respect to the CPU condition with 32 threads. GPU throughput increases with larger batch sizes be-

	Batch Size	Number of Sentences Number of Tokens	100 2.8k	500 14.5k	1k 28.8k	2k 57.9k	4k 117.9k	6k 161.9k	8k 214.2k	16k 436.5k	32k 893.9k
+Sampling	Xinhua	Throughput (words/s)	236	667	914	1356	1613	1794	2001	2793	3998
		Speedup	1.0×	2.7×	3.7×	5.5×	6.5×	7.2×	8.1×	11.3×	16.1×
	Xinhua+UN	Throughput (words/s)	106	223	287	400	454	514	571	709	1016
		Speedup	1.4×	2.9×	3.8×	5.3×	6.0×	6.8×	7.5×	9.3×	13.4×
−Sampling	Xinhua	Throughput (words/s)	84	280	405	690	929	1200	1414	2135	3240
		Speedup	13×	45×	65×	111×	149×	193×	227×	343×	520×
	Xinhua+UN	Throughput (words/s)	19	62	99	172	248	304	357	509	793
		Speedup	22×	69×	112×	193×	279×	342×	401×	572×	891×

Table 7.8: GPU grammar extraction throughput (words/second) under different batch sizes, data conditions, with and without sampling. Speedup is computed with respect to the CPU baseline running on 32 threads.

cause we are increasingly able to saturate the GPU and take full advantage of the massive parallelism it offers. We do not observe this effect on the CPU since we saturate the processors early.

With a batch size of 100 sentences, the GPU is slightly slower than the 32-thread CPU implementation on Xinhua and faster on Xinhua + UN, both with sampling. Without sampling, the GPU is already an order of magnitude faster at a batch size of 100 sentences. At a batch size of 500 sentences, the GPU implementation is substantially faster than the 32-thread CPU version across all conditions. With the largest batch size in our experiments of 32k sentences, the GPU is over an order of magnitude faster than the fully-saturated CPU with sampling, and over two orders of magnitude faster without sampling. Although previous work does not show decreased translation quality due to sampling [20, 89], these results illustrate the raw computational potential of GPUs, showing that we can eliminate heuristics that make CPU processing tractable. We believe future work can exploit these untapped processing cycles to improve translation quality.

How does the GPU fare for translation tasks that demand low latency, such as

	+Sampling		−Sampling	
	X	X+U	X	X+U
GPU one-by-one	7.23	4.7	2.39	0.71
CPU single-thread	12.7	4.8	0.32	0.05

Table 7.9: Sentence-by-sentence GPU grammar extraction throughput (words/second) vs. a single thread on the CPU (X: Xinhua, X+U: Xinhua + UN).

sentence-by-sentence translation on the web? To find out, we conducted experiments where the sentences are fed, one by one, to the GPU grammar extraction algorithm. Results are shown in Table 7.9, with a comparison to a single-threaded CPU baseline. To be consistent with the other results, we also measure speed in terms of throughput here. Note that we are not aware of any freely available multi-threaded CPU algorithm to process an *individual* sentence in parallel, so the single-thread CPU comparison is reasonable.[4] We observe that the GPU is slower only with sampling on the smaller Xinhua data. In all other cases, sentence-by-sentence processing on the GPU achieves a similar level of performance or is faster.

Next, we compare the performance of hierarchical grammar extraction to phrase-based extraction on the GPU with sampling. We replicated the test data condition of [58] so that our first 8k query sentences are the same as those used in their experiments. The results are shown in Table 7.10, where we report grammar extraction time for batches of different sizes; the bottom row shows the slowdown of the hierarchical vs. non-hierarchical grammar conditions. This quantifies the performance penalty to achieve the translation quality gains reported in Table 7.6. Hierarchical grammar extraction is about three times slower, primarily due to the computational costs of the new passes.

[4]Parallel sub-sentential parsing has been known for many years [23] although we don't know of an implementation in any major open-source MT systems.

Queries	2k	4k	6k	8k
GPU PBMT	15s	25s	29s	33s
GPU Hiero	43s	73s	90s	107s
Slowdown	2.8×	2.9×	3.1×	3.2×

Table 7.10: Grammar extraction time comparing this work (GPU Hiero) and the work of He et al. (2013) (GPU PBMT).

	CPU	GPU
PBMT	1.6 GB	2.3 GB
Hiero	1.9 GB	8.0 GB

Table 7.11: Memory consumption (CPU RAM) for different experimental conditions.

Another aspect of performance is memory footprint. We report the memory use (CPU RAM) of all four conditions in Table 7.11. The values reported for the CPU implementation use a single thread only. At runtime, our hierarchical GPU system exhibits peak CPU memory use of 8 GB on the host machine. Most of this memory is consumed by batching extracted phrases before scoring in passes 10 through 14. Since the phrase-based GPU implementation processes far fewer phrases, the memory footprint is much smaller. The CPU implementations process extracted phrases in small batches grouped by source phrase, and thus exhibit less memory usage. However, these levels of memory consumption are modest considering modern hardware. In all other respects, memory usage is similar for all systems, since the suffix array and associated data structures are all linear in the size of the indexed parallel text.

Results on Per-Pass Speed

To obtain a detailed picture of where the GPU speedups and bottlenecks are, we collected per-pass timing statistics. Table 7.12 shows results for grammar extraction on

6k queries using the Xinhua data with no sampling and default length constraints (passes in gray occur on the GPU; all others on the CPU). These numbers explain the decreased speed of hierarchical extraction compared to He et al. [58] as described earlier, with the new passes (shown in *italics*) accounting for more than 75% of the total computation time. However, even passes that are nominally the same actually require more time in the hierarchical case: in extracting and scoring phrases associated with a contiguous pattern u, we must now also extract and score patterns $\star\ u$, $u\ \star$, and $\star\ u\ \star$.

Interestingly, the CPU portions of our algorithm account for around half of the total grammar extraction time. One way to interpret this observation is that the massive parallelization provided by the GPU is so effective that we are bottlenecked by the CPU. In our current design, the CPU portions are those that cannot be easily parallelized on the GPU or those that require too much memory to fit on the GPU. The former is a possible target for optimization in future work, though the latter will likely be solved by hardware advances alone: for example, the Tesla K80 has 24 GB of memory.

Results on Translation Quality Improvement

Due to efficiency issues with *cdec* grammar extractor, users have to add grammar rule length constraints, sampling, and search limits during the extraction process. Now given a faster GPU-based grammar extractor, we would like to relax all the constraints and investigate whether longer grammar rules would be beneficial for final translation quality of the hierarchical SMT system. Therefore we conduct SMT experiments comparing grammar rules with default length constraints as in *cdec* vs longer grammar rules from

Pass	Time	%
Contig. pattern pass 1	0.03	0.02%
Contig. pattern pass 2	0.02	0.02%
One-gap pattern generation	1.14	0.86%
One-gap pattern matching	19.24	14.54%
Two-gap pattern generation	0.37	0.28%
Two-gap pattern matching	15.91	12.02%
Gappy pattern processing	1.21	0.91%
Contig. pattern extraction	13.94	10.53%
Two-gap pattern extraction	0.52	0.39%
One-gap pattern extraction	11.07	8.36%
One gap translation features	23.72	17.91%
Two gap translation features	30.90	23.34%
Contig. translation features	5.00	3.77%
Lexical weight feature	3.09	2.33%
Data transfer and control	6.23	4.71%
Total	132.39	100.0%

Table 7.12: Detailed timings (in seconds) for 6k queries. Passes in gray occur on the GPU; all others on the CPU. Passes needed for hierarchical grammars are in *italics*.

our GPU grammar extractor.

We use the 2003 NIST SMT Chinese-to-English evaluation data (919 sentence pairs) as the development set, and the 2002, 2005, 2006 and 2008 NIST SMT Chinese-to-English evaluation data (878, 1,082, 616 and 1,357 sentence pairs, respectively) as the test set. We use a reasonably competitive cdec system which already incorporates sparse and syntax features. We explore longer grammar rules in the following four aspects:

1. We relax source side grammar pattern length restrictions, from default length 15 to 35.

2. We use non-sampling grammar rule extraction, instead of sampling with the default value of 300.

3. We relax target side grammar rule searching span length from default value 15 to

	NIST02	NIST05	NIST06	NIST08
cdec+Default Grammar Rules	42.85	34.66	32.0	24.7
cdec+Longer Grammar Rules	**43.57**	**35.28**	**32.79**	**25.39**
BLEU Improvement	+0.72	+0.62	+0.79	+0.69

Table 7.13: Translation quality comparison given longer grammar rules vs normal grammar rules.

150.

4. We increase the number of symbols (terminal+non-terminal) in grammar patterns from default value 5 to 7.

We show the NIST BLEU score differences in Table 7.13. We found that there is a consistent BLEU score improvement around +0.7 when the longer grammar rules are in use, this is in contradictory to what previous work have found. We hypothesis the translation quality improvement is due to the fact a significantly larger number of grammar rules is generated by our GPU extractor than ones of any previous work.

Results on End-to-end Speed

What is the benefit of using GPUs in an end-to-end translation task? Since we have shown that both the CPU and GPU implementations achieve near-identical translation quality, the difference lies in speed. But speed is difficult to measure fairly: translation involves not only grammar extraction but also decoding and associated I/O. We have focused on grammar extraction on the GPU, but our research vision involves eventually moving all components of the machine translation pipeline onto the GPU. The experiments we describe below capture the performance advantages of our implementation that

Grammar Extraction	Disk I/O	Decoding
GPU: 30.8s CPU: 101.1s	13.4s	CPU: 59s

Table 7.14: Running times for an end-to-end translation pipeline over NIST03 test data. Grammar extraction is either performed on the GPU or the CPU (32 threads); other stages are the same for both conditions (decoding uses 32 threads).

are achievable today, using cdec for decoding (on the CPU, using 32 threads).

To measure end-to-end translation speed using pycdec, per-sentence grammars are first extracted for a batch of sentences and written to disk, then read from disk during decoding. Therefore, we report times separately for grammar extraction, disk I/O, and decoding (which includes time for reading the grammar files from disk back into memory). Grammar extraction is either performed on the GPU or on the CPU (using 32 threads), same as the experiments described in the previous section.

Results are shown in Table 7.14 using the Xinhua training data and NIST03 test data (919 sentences, 27,045 words). All experiment settings are exactly the same as in the previous section. We observe an end-to-end translation throughput of 262 words/second with GPU grammar extraction and 156 words/second on the CPU (32 threads), for a speedup of 1.68×.

Despite the speedup, we note that this experiment favors the CPU for several reasons. First, the GPU is idle during decoding, but it could be used to process grammars for a subsequent batch of sentences in a pipelined fashion. Second, NIST03 is a small batch that doesn't fully saturate the GPU—throughput keeps increasing by a large margin with larger batch sizes (see results in Table 7.8). Third, in comparison to the 32-thread CPU baseline, our GPU extraction only uses a single thread on the CPU, thus the CPU portion

of the performance can be further improved, especially in the feature generation passes (see Sec. 7.5.6).

Of course, the GPU/CPU combination requires a server equipped with a GPU, incurring additional hardware costs. We estimate that in Q4 2014 dollars, our base system would cost roughly $7500 USD, and the GPU would cost another $2600 USD. However, the server-grade GPU used in this work is not the only choice: a typical high-end consumer GPU, such as the NVIDIA GTX Titan Black (around $1100), costs considerably less but has even higher memory bandwidth and with similarly impressive floating point performance. This price difference is due to extra functionalities (e.g., error-correcting code memory) for specific applications (e.g., scientific computing), and is not directly related to differences in raw computational power. This means that we could speed up overall translation by 68% if we spend an additional 35% (server-grade GPU) or 15% (consumer-grade GPU) on hardware. From an economic perspective, this is an attractive proposition. Of course, the advantages of using GPUs for high-throughput translation go up further with larger batch sizes.

Results on One-time Construction Costs

Construction of static data structures for on-demand grammar extraction is a one-time cost given a corpus T. However, under a streaming scenario where we might receive incremental changes to T as new training data become available, we need to update the data structures appropriately.

Updating static data structures involves two costs: the suffix array with its LCP

array and the precomputation indexes. We do not consider the alignment construction cost as it is external to cdec (and is a necessary step for all implementations). For the Xinhua data, building the suffix array using a single CPU thread takes 29.2 seconds and building the precomputation indexes on the GPU takes 5.7 seconds. Compared to Table 7.12, these one-time costs represent approximately 27% of the GPU grammar extraction time.

It is possible to lower the construction costs given recent advances. Levenberg et al. [82] describe novel algorithms that allow efficient in-place updates of the suffix array when new training data arrive. That work directly tackles on-demand SMT architectures in the streaming data scenario. Alternatively, the speed of suffix array construction can be improved by the CUDA Data Parallel Primitives Library,[5] which provides sorting algorithms to efficiently construct suffix arrays on the GPU. Minimizing data preparation costs has not been a focus of this work, but we believe the massive parallelism from the GPU represents promising future work.

7.6 Summary

In this chapter we present our novel massively parallel algorithms on GPUs for the high performance cross-lingual pattern matching task. Our work consists of three major components: the parallel phrase lookup with suffix array on GPUs; the contiguous pattern matching and extraction for phrase-based SMT; and lastly the gappy pattern matching for hierarchical translation grammars of SMT.

We focus on increasing the utilization of GPU's raw processing power to improve

[5] http://cudpp.github.io/cudpp/2.2/

the performance and developed algorithms with higher level of massive parallelism; we also design our GPU algorithms with smaller memory footprint by using compact data structure. Our GPU-based grammar extraction of SMT tool achieves orders of magnitude efficiency improvement over previous state-of-the-art CPU baseline.

Chapter 8: Conclusion and Future Work

Identifying similar pieces of texts remains one of the fundamental problems in computational linguistics. In this dissertation, we focus on a variety of textual similarity tasks that share common properties, including, paraphrase identification, answer selection for question answering, cross-lingual and monolingual textual similarity measurement, pairwise learning to rank, insight extraction on biomedical literature and high-performance textual similarity identification task, and present our efforts to address these tasks given over 20 public benchmarks.

In contrast to the standard neural network choice of the Siamese architecture [19], which utilizes single representation view and straightforward similarity comparison, we develop novel interaction-based neural networks that encourage multi-perspective input representation and closer input interactions, with which our multiple neural network models show highly competitive performance on most of the public benchmarks we used.

Our first multi-perspective convolutional neural networks (MPCNN) as introduced in Chapter 3 uses a multiplicity of perspectives to process input sentences with multiple parallel convolutional neural networks, extracts salient sentence-level features automatically at multiple granularity with different types of pooling. Our novel structured similarity layer encourages the input interaction modeling by comparing local regions of both

sentence representations.

In addition, in Chapter 3.7 we provide an attention-based input interaction layer on top of our MPCNN model. The input interaction layer models a closer relationship of input words by converting two separate sentences into an inter-related sentence pair. This layer utilizes the attention mechanism in an effective way without introducing extra parameters and is another example of our interaction-based neural modeling.

In Chapter 4 we provide our *pairwise word interaction model with very deep neural networks* (PWI). We directly encode input word interactions with novel pairwise word interaction modeling and our similarity focus layer. The use of very deep architecture in this model is also the first example to our best knowledge in the NLP domain for better textual similarity modeling. Our pairwise word interaction model significantly outperforms the Siamese architecture and feature engineering approach on multiple tasks, and is another example of our interaction-based neural modeling.

In Chapter 5 we focus on the question answering task with a pairwise ranking approach. Unlike traditional pointwise approach of the task, our pairwise ranking approach uses Noise-Contrastive Estimation, focuses on modeling interactions between the pairs and learns the relative order of the pairs to predict which answer is more relevant to the question. This is also a direct extension which utilizes both our MPCNN and PWI models. We demonstrate the high effectiveness of our interaction-based approach against competitive pointwise baselines.

In Chapter 6 for insight extraction on biomedical literature task, we develop neural networks with novel similarity modeling for better relation extraction, as we convert the extraction task into a similarity measurement task. Our approach innovates in that it

explicitly models the interactions among the trio: named entities, entity relations and contexts, and then measures both relational and contextual similarity among them, finally integrate both similarity evaluations into considerations for insight extraction. We also build an end-to-end system to extract the insights, with human evaluations we show our system is able to extract insights with high human acceptance accuracy.

Lastly in Chapter 7 we focus on the textual similarity identification task, and introduce novel contiguous and approximate cross-lingual pattern matching algorithms on GPUs that can run on GPUs efficiently for machine translation. In order to get high efficiency for the similarity identification task on GPUs, we show developing massively parallel algorithms on GPUs is the most important approach to fully utilize GPU's raw processing power, and developing compact data structures on GPUs is also helpful to lower GPU's memory latency. As a result our massively parallel approach is orders of magnitude faster than the previous state-of-the-art CPU baseline.

We finally discuss directions for future work. First of all, in order to achieve improved performance, better modeling should be critical. Since we have shown the interaction-based neural modeling can achieve competitive performance on textual similarity measurement tasks, we believe one promising and very clear direction is toward higher level of interaction-based modeling, e.g. finer-grained interactions should be modeled in order to achieve further improvement. Interaction-based modeling essentially represents the prior knowledge injection process to better solve the textual similarity measurement problems, and there are indeed many possibilities to model closer interactions between inputs. And in the end different ways of interactions should definitely be closely embedded as the core of deep neural network models, not as extra feature sets or auxiliary

components.

Besides better modeling, the other interesting direction is the automatic loss learning or metric learning for better textual similarity measurement, as there is usually not a clear borderline for the similarity as a concept, the loss function or metric learning does not require much human annotation efforts, but learn the similarity meaning by itself. Metric learning ultimately allows learning not just from human-annotated datasets but also from things that exist already in nature. But doing so actually essentially mimics the process of automatic annotated data collection but on the learning side, since high quality annotated datasets could be very expensive to obtain.

Lastly, in terms of resources and to support open science, releasing source codes of existing approaches is helpful for future research (e.g. our work[1]). Also, high quality annotated datasets for textual similarity measurement still plays a major role in the area. Collecting large amount of data and annotating them is labor intensive, but highly useful to advance related research (e.g. TwitterPPDB [80]).

[1] https://github.com/hohoCode

Bibliography

[1] ACL. Question answering (state of the art), http://aclweb.org/aclwiki/index.php?title=Question_Answering_(State_of_the_art), accessed Aug., 18, 2016.

[2] Eneko Agirre, Mona Diab, Daniel Cer, and Aitor Gonzalez-Agirre. SemEval-2012 task 6: a pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, pages 385–393, 2012.

[3] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. SemEval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation*, pages 81–91, 2014. URL http://www.aclweb.org/anthology/S14-2010.

[4] Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, German Rigau, Larraitz Uria, and Janyce Wiebe. Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 252–263, Denver, Colorado, June 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/S15-2045.

[5] Yaser Al-onaizan, Jan Curin, Michael Jahr, Kevin Knight, John Lafferty, Dan Melamed, Franz-Josef Och, David Purdy, Noah A. Smith, and David Yarowsky. Statistical machine translation. Final report, JHU Summer Workshop on Language Engineering, 1999.

[6] Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1247–1255, 2013.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Paul Baltescu and Phil Blunsom. A fast and simple online synchronous context free grammar extractor. *The Prague Bulletin of Mathematical Linguistics*, 102(1): 17–26, 2014.

[10] Daniel Bär, Chris Biemann, Iryna Gurevych, and Torsten Zesch. UKP: computing semantic textual similarity by combining multiple content similarity measures. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, pages 435–440, 2012.

[11] Islam Beltagy, Katrin Erk, and Raymond Mooney. Probabilistic soft logic for semantic textual similarity. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics*, pages 1210–1219, 2014.

[12] Shane Bergsma, Aditya Bhargava, Hua He, and Grzegorz Kondrak. Predicting the semantic compositionality of prefix verbs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 293–303, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1870658.1870687.

[13] Johannes Bjerva, Johan Bos, Rob van der Goot, and Malvina Nissim. The meaning factory: formal semantics for recognizing textual entailment and determining semantic similarity. *International Workshop on Semantic Evaluation*, 2014.

[14] William Blacoe and Mirella Lapata. A comparison of vector-based representations for semantic composition. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 546–556, 2012. URL http://dl.acm.org/citation.cfm?id=2390948.2391011.

[15] Olivier Bodenreider. The unified medical language system (UMLS): integrating biomedical terminology. *Nucleic Acids Research*, 32:D267, 2004. URL http://dx.doi.org/10.1093/nar/gkh061.

[16] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2787–2795. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf.

[17] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

[18] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):669–688, 1993.

[19] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. *IJPRAI*, 1993.

[20] Chris Callison-Burch, Colin Bannard, and Josh Schroeder. Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL 2005)*, pages 255–262, 2005.

[21] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14. Association for Computational Linguistics, 2017. doi: 10.18653/v1/S17-2001. URL http://aclanthology.coli.uni-saarland.de/pdf/S/S17/S17-2001.pdf.

[22] Victor Chahuneau, Noah A. Smith, and Chris Dyer. pycdec: A Python interface to cdec. In *Proceedings of the 7th Machine Translation Marathon (MTM 2012)*, 2012.

[23] M. Chandwani, M. Puranik, and N. S. Chaudhari. On CKY-parsing of context-free grammars in parallel. In *Proceedings of Technology Enabling Tomorrow: Computers, Communications and Automation towards the 21st Century*, pages 141–145, 1992.

[24] Elizabeth S. Chen, George Hripcsak, Hua Xu, Marianthi Markatou, and Carol Friedman. Automated acquisition of diseasedrug knowledge from biomedical and clinical documents: An initial study. *Journal of the American Medical Informatics Association*, 15(1):87, 2008. URL http://dx.doi.org/10.1197/jamia.M2401.

[25] David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.

[26] David Chiang. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13:1159–1187, 2012.

[27] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1179.

[28] Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*, pages 176–181, 2011.

[29] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine learning*, pages 160–167, 2008.

[30] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2078186.

[31] Mark Craven. Learning to extract relations from medline. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 25–30, 1999.

[32] Fabien Cromieres and Sadao Kurohashi. Efficient retrieval of tree translation examples for syntax-based machine translation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011*, pages 508–518, 2011.

[33] Hang Cui, Renxu Sun, Keya Li, Min-Yen Kan, and Tat-Seng Chua. Question answering passage retrieval using dependency relations. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 400–407, 2005. ISBN 1-59593-034-5. doi: 10.1145/1076034.1076103. URL http://doi.acm.org/10.1145/1076034.1076103.

[34] Dipanjan Das and Noah A. Smith. Paraphrase identification as probabilistic quasi-synchronous recognition. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 468–476, 2009. ISBN 978-1-932432-45-9. URL http://dl.acm.org/citation.cfm?id=1687878.1687944.

[35] Rezarta Islamaj Dogan, Aurélie Névéol, and Zhiyong Lu. A context-blocks model for identifying clinical relationships in patient records. *BMC Bioinformatics*, 12(S-3):S3, 2011. doi: 10.1186/1471-2105-12-S3-S3. URL http://dx.doi.org/10.1186/1471-2105-12-S3-S3.

[36] Bill Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: exploiting massively parallel news sources. In *Proceedings of the 20th International Conference on Computational Linguistics*, 2004.

[37] Chris Dyer, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, pages 7–12, 2010.

[38] Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering. In *ACL*, 2003.

[39] Miles Efron, Jimmy Lin, Jiyin He, and Arjen de Vries. Temporal feedback for tweet search with non-parametric density estimation. In *SIGIR*, pages 33–42, 2014.

[40] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[41] Samuel Fern and Mark Stevenson. A semantic similarity approach to paraphrase detection. In *Computational Linguistics UK 11th Annual Research Colloquium*, 2008.

[42] Katja Filippova, Enrique Alfonseca, Carlos A. Colmenares, Lukasz Kaiser, and Oriol Vinyals. Sentence compression by deletion with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 360–368, 2015.

[43] Andrew Finch. Using machine translation evaluation techniques to determine sentence-level semantic equivalence. In *Proceedings of the International Workshop on Paraphrasing*, 2005.

[44] Jenny Finkel, Shipra Dingare, Christopher D. Manning, Malvina Nissim, Beatrice Alex, and Claire Grover. Exploring the boundaries: gene and protein identification in biomedical text. *BMC Bioinformatics*, 2005. doi: 10.1186/1471-2105-6-S1-S5. URL https://doi.org/10.1186/1471-2105-6-S1-S5.

[45] Carol Friedman, Lyudmila Shagina, Yves A. Lussier, and George Hripcsak. Automated encoding of clinical documents based on natural language processing. *Journal of the American Medical Informatics Association*, 11:392–402, 2004.

[46] Michel Galley and Christopher D. Manning. Accurate non-hierarchical phrase-based translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL 2010)*, pages 966–974, 2010. ISBN 1-932432-65-5. URL http://dl.acm.org/citation.cfm?id=1857999.1858138.

[47] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. PPDB: the Paraphrase Database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013.

[48] Abdullah Gharaibeh and Matei Ripeanu. Size matters: Space/time tradeoffs to improve GPGPU applications performance. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2010)*, pages 1–12, 2010. ISBN 978-1-4244-7559-9. doi: 10.1109/SC.2010.51. URL http://dx.doi.org/10.1109/SC.2010.51.

[49] Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of the International Conference on Neural Networks*, pages 347–352, 1996.

[50] Jesús González-Rubio, Daniel Ortiz-Martínez, and Francisco Casacuberta. Active learning for interactive machine translation. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2012)*, pages 245–254, 2012. URL http://www.aclweb.org/anthology/E12-1025.

[51] Usha Goswami. *Analogical reasoning in children*. Psychology Press, 1992.

[52] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Bidirectional LSTM networks for improved phoneme classification and recognition. In *Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and Their Applications - Part II*, pages 799–804, 2005.

[53] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 369–376, 2006. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143891. URL http://doi.acm.org/10.1145/1143844.1143891.

[54] Weiwei Guo and Mona Diab. Modeling sentences in the latent space. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 864–872, 2012. URL http://dl.acm.org/citation.cfm?id=2390524.2390644.

[55] Samer Hassan. *Measuring Semantic Relatedness Using Salient Encyclopedic Concepts*. PhD thesis, University of North Texas, Denton, Texas, USA, 2011.

[56] Hua He and Jimmy 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*, pages 937–948, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1108.

[57] Hua He, Denilson Barbosa, and Grzegorz Kondrak. Identification of speakers in novels. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1312–1320, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P13-1129.

[58] Hua He, Jimmy Lin, and Adam Lopez. Massively parallel suffix array queries and on-demand phrase extraction for statistical machine translation using GPUs. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 325–334, 2013. URL http://www.aclweb.org/anthology/N13-1033.

[59] Hua He, Kevin Gimpel, and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1576–1586, 2015. URL http://aclweb.org/anthology/D15-1181.

[60] Hua He, Jimmy Lin, and Adam Lopez. Gappy pattern matching on gpus for on-demand extraction of hierarchical translation grammars. *TACL*, 3:87–100, 2015. URL https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/555.

[61] Hua He, John Wieting, Kevin Gimpel, Jinfeng Rao, and Jimmy Lin. UMD-TTIC-UW at SemEval-2016 task 1: Attention-based multi-perspective convolutional neural networks for textual similarity measurement. In *SemEval*, 2016.

[62] Hua He, Kris Ganjam, Navendu Jain, Jessica Lundin, Ryen White, and Jimmy Lin. An insight extraction system on biomedical literature with deep neural networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2681–2691, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D17-1284.

[63] Michael Heilman and Noah A. Smith. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1011–1019, 2010. ISBN 1-932432-65-5. URL http://dl.acm.org/citation.cfm?id=1857999.1858143.

[64] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. SemEval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, pages 94–99. Association for Computational Linguistics, 2009.

[65] Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 33–38, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/S10-1006.

[66] Jared Hoberock and Nathan Bell. Thrust: A parallel template library, 2010. URL http://thrust.github.io/. Version 1.8.0.

[67] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[68] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*, pages 2042–2050, 2014.

[69] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, pages 2333–2338, 2013.

[70] Abhyuday N. Jagannatha and Hong Yu. Bidirectional RNN for medical event detection in electronic health records. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 473–482, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1056.

[71] Yangfeng Ji and Jacob Eisenstein. Discriminative improvements to distributional sentence similarity. In *Proceedings of the 2013 Conference on Empirical Methods for Natural Language Processing*, pages 891–896, 2013. URL http://www.cc.gatech.edu/~jeisenst/papers/ji-emnlp-2013.pdf.

[72] Sergio Jimenez, George Duenas, Julia Baquero, Alexander Gelbukh, Av Juan Dios Bátiz, and Av Mendizábal. UNAL-NLP: combining soft cardinality features for semantic textual similarity, relatedness and entailment. *International Workshop on Semantic Evaluation*, 2014.

[73] Renata Kabiljo, Andrew B. Clegg, and Adrian J. Shepherd. A realistic assessment of methods for extracting gene/protein interactions from free text. *BMC Bioinformatics*, 2009. doi: 10.1186/1471-2105-10-233. URL http://dx.doi.org/10.1186/1471-2105-10-233.

[74] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[75] Abhay Kashyap, Lushan Han, Roberto Yus, Jennifer Sleeman, Taneeya Satyapanich, Sunil Gandhi, and Tim Finin. Meerkat mafia: Multilingual and cross-level semantic textual similarity systems. In *Proceedings of the 8th International Workshop on Semantic Evaluation*, pages 416–423, 2014. URL http://www.aclweb.org/anthology/S14-2072.

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

[77] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses:

Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180, 2007. URL http://dl.acm.org/citation.cfm?id=1557769.1557821.

[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[79] Alice Lai and Julia Hockenmaier. Illinois-LH: a denotational and distributional approach to semantics. *International Workshop on Semantic Evaluation*, 2014.

[80] Wuwei Lan, Siyu Qiu, Hua He, and Wei Xu. A continuously growing dataset of sentential paraphrases. In *Proceedings of The 2017 Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2017.

[81] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning*, pages 1188–1196, 2014.

[82] Abby Levenberg, Chris Callison-Burch, and Miles Osborne. Stream-based translation models for statistical machine translation. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL 2010)*, pages 394–402, 2010. ISBN 1-932432-65-5. URL http://dl.acm.org/citation.cfm?id=1857999.1858061.

[83] Yuxi Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017. URL http://arxiv.org/abs/1701.07274.

[84] Jimmy Lin. An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems*, 25(2):1–55, 2007.

[85] Yifeng Liu. Question answering for biomedicine. PhD Dissertation, University of Alberta, 2016. doi: 10.7939/r3wh2dk7t.

[86] Yifeng Liu, Yongjie Liang, and David Wishart. PolySearch2: a significantly improved text-mining system for discovering associations between human diseases, genes, drugs, metabolites, toxins and more. *Nucleic Acids Research*, 43(W1): W535–W542, 2015. doi: 10.1093/nar/gkv383. URL +http://dx.doi.org/10.1093/nar/gkv383.

[87] Adam Lopez. Hierarchical phrase-based translation with suffix arrays. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*, pages 976–985, 2007.

[88] Adam Lopez. *Machine translation by pattern matching*. Ph.D. dissertation, University of Maryland, College Park, Maryland, USA, 2008.

[89] Adam Lopez. Tera-scale translation models via pattern matching. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, pages 505–512, 2008.

[90] André Lynum, Partha Pakray, Björn Gambäck, and Sergio Jimenez. NTNU: Measuring semantic similarity with sublexical feature representations and soft cardinality. In *Proceedings of the 8th International Workshop on Semantic Evaluation*, pages 448–453, 2014. URL http://www.aclweb.org/anthology/S14-2078.

[91] Nitin Madnani, Joel Tetreault, and Martin Chodorow. Re-examining machine translation metrics for paraphrase identification. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 182–190, 2012. ISBN 978-1-937284-20-6. URL http://dl.acm.org/citation.cfm?id=2382029.2382055.

[92] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 319–327, 1990. URL http://dl.acm.org/citation.cfm?id=320176.320218.

[93] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014. URL http://www.aclweb.org/anthology/P/P14/P14-5010.

[94] Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. SemEval-2014 task 1: evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. *International Workshop on Semantic Evaluation*, 2014.

[95] Yishu Miao, Lei Yu, and Phil Blunsom. Neural variational inference for text processing. *arXiv:1511.06038*, 2015.

[96] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at International Conference on Learning Representations*, 2013. URL http://arxiv.org/abs/1301.3781.

[97] Makoto Miwa and Mohit Bansal. End-to-end relation extraction using LSTMs on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1116, Berlin, Germany, August 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1105.

[98] Raymond J. Mooney. Semantic parsing: Past, present, and future. In *ACL Workshop on Semantic Parsing. Presentation slides.*, 2014.

[99] Franz Josef Och, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical machine translation. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP 1999)*, pages 20–28, 1999.

[100] John Paparrizos, Ryen W. White, and Eric Horvitz. Detecting devastating diseases in search logs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 559–568, 2016. URL http://doi.acm.org/10.1145/2939672.2939722.

[101] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014. URL http://aclweb.org/anthology/D/D14/D14-1162.pdf.

[102] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. *EMNLP*, 2014.

[103] Aaron B. Philips. *Modeling Relevance in Statistical Machine Translation: Scoring Alignment, Context, and Annotations of Translation Instances*. PhD thesis, Carnegie Mellon University, 2012.

[104] George Polya. *How to Solve It: A New Aspect of the Mathematical Method*. Princeton University Press, 1957/2004.

[105] Jay M. Ponte and W. Croft. A language modeling approach to information retrieval. In *SIGIR*, pages 275–281, 1998.

[106] Hoifung Poon and Lucy Vanderwende. Joint inference for knowledge extraction from biomedical literature. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 813–821, Los Angeles, California, June 2010. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N10-1123.

[107] Hoifung Poon, Chris Quirk, Charlie DeZiel, and David Heckerman. Literome: PubMed-scale genomic knowledge base in the cloud. *Bioinformatics*, 30(19): 2840, 2014. doi: 10.1093/bioinformatics/btu383. URL +http://dx.doi.org/10.1093/bioinformatics/btu383.

[108] Long Qiu, Min-Yen Kan, and Tat-Seng Chua. Paraphrase recognition via dissimilarity significance classification. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 18–26, 2006. ISBN 1-932432-73-6. URL http://dl.acm.org/citation.cfm?id=1610075.1610079.

[109] Jinfeng Rao, Jimmy Lin, and Miles Efron. Reproducible experiments on lexical and temporal feedback for tweet search. In *ECIR*, pages 755–767. 2015.

[110] Jinfeng Rao, Hua He, and Jimmy Lin. Noise-contrastive estimation for answer selection with deep neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1913–1916, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983872. URL http://doi.acm.org/10.1145/2983323.2983872.

[111] Jinfeng Rao, Hua He, and Jimmy Lin. Experiments with convolutional neural network models for answer selection. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1217–1220, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5022-8. doi: 10.1145/3077136.3080648. URL http://doi.acm.org/10.1145/3077136.3080648.

[112] Jinfeng Rao, Hua He, H. Zhang, Ferhan Ture, R. Sequiera, Salman Mohammed, and Jimmy Lin. Integrating Lexical and Temporal Signals in Neural Ranking Models for Searching Social Media Streams. *ArXiv e-prints*, July 2017.

[113] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press. ISBN 978-0-9749039-5-8. URL http://dl.acm.org/citation.cfm?id=1795114.1795167.

[114] Steven Rennie, Vaibhava Goel, and Samuel Thomas. Deep order statistic networks. In *Proceedings of the IEEE Workshop on Spoken Language Technology*, 2014.

[115] Thomas C. Rindflesh and Marcelo Fiszman. The interaction of domain knowledge and linguistic structure in natural language processing: Interpreting hypernymic propositions in biomedical text. *J. of Biomedical Informatics*, 36(6):462–477, December 2003. ISSN 1532-0464. URL http://dx.doi.org/10.1016/j.jbi.2003.11.003.

[116] Bryan Rink and Sanda Harabagiu. UTD: Classifying semantic relations by combining lexical and semantic resources. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 256–259, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1859664.1859721.

[117] Miguel Rios, Wilker Aziz, and Lucia Specia. UOW: semantically informed text similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, pages 673–678, 2012.

[118] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1044.

[119] Cicero dos Santos, Ming Tan, Bing Xiang, and Bowen Zhou. Attentive pooling networks. *arXiv:1602.03609*, 2016.

[120] Frane Šarić, Goran Glavaš, Mladen Karan, Jan Šnajder, and Bojana Dalbelo Bašić. TakeLab: systems for measuring semantic text similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, pages 441–448, 2012.

[121] Michael Schatz, Cole Trapnell, Arthur Delcher, and Amitabh Varshney. High-throughput sequence alignment using graphics processing units. *BMC Bioinformatics*, 8(1):474, 2007. ISSN 1471-2105. URL http://www.biomedcentral.com/1471-2105/8/474.

[122] Aliaksei Severyn and Alessandro Moschitti. Automatic feature engineering for answer selection and extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 458–467, 2013.

[123] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 373–382, 2015. ISBN 978-1-4503-3621-5. doi: 10.1145/2766462.2767738. URL http://doi.acm.org/10.1145/2766462.2767738.

[124] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *SIGIR*, 2015.

[125] Michel Simard, Nicola Cancedda, Bruno Cavestro, Marc Dymetman, Eric Gaussier, Cyril Goutte, and Kenji Yamada. Translating with non-contiguous phrases. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (EMNLP 2005)*, pages 755–762, 2005.

[126] Patrick Simianer, Stefan Riezler, and Chris Dyer. Joint feature selection in distributed stochastic learning for large-scale discriminative training in SMT. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, pages 11–21, 2012.

[127] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

[128] Matthew S. Simpson and Dina Demner-Fushman. Biomedical text mining: A survey of recent progress. In *Mining Text Data*, pages 465–517, Boston, MA, 2012. Springer US.

[129] Samuel L. Smith, David H. P. Turban, Steven Hamblin, and Nils Y. Hammerla. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *CoRR*, abs/1702.03859, 2017.

[130] Mark D. Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *CIKM*, pages 623–632, 2007.

[131] Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, 2011.

[132] Richard Socher, Andrej Karpathy, Quoc V. Le, Christopher D. Manning, and Andrew Y. Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2: 207–218, 2014.

[133] Md Arafat Sultan, Steven Bethard, and Tamara Sumner. Dls@cu: Sentence similarity from word alignment. In *Proceedings of the 8th International Workshop on Semantic Evaluation*, pages 241–246, 2014. URL http://www.aclweb.org/anthology/S14-2039.

[134] Md Arafat Sultan, Steven Bethard, and Tamara Sumner. Dls@cu: Sentence similarity from word alignment and semantic vector composition. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 148–153, Denver, Colorado, June 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/S15-2027.

[135] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112, 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

[136] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. URL http://arxiv.org/abs/1409.4842.

[137] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, 2015.

[138] Stefanie Tellex, Boris Katz, Jimmy Lin, Gregory Marton, and Aaron Fernandes. Quantitative evaluation of passage retrieval algorithms for question answering. In *SIGIR*, 2003.

[139] Junfeng Tian, Zhiheng Zhou, Man Lan, and Yuanbin Wu. Ecnu at semeval-2017 task 1: Leverage kernel-based traditional nlp features and neural networks

to build a universal model for multilingual and cross-lingual semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 191–197. Association for Computational Linguistics, 2017. doi: 10.18653/v1/S17-2028. URL http://aclanthology.coli.uni-saarland.de/pdf/S/S17/S17-2028.pdf.

[140] Tijmen Tieleman and Geoffrey E. Hinton. Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. Coursera: Neural Networks for Machine Learning, 2012.

[141] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1499–1509, 2015. URL http://aclweb.org/anthology/D/D15/D15-1174.pdf.

[142] Kristina Toutanova, Victoria Lin, Wen-tau Yih, Hoifung Poon, and Chris Quirk. Compositional learning of embeddings for relation paths in knowledge base and text. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016. URL http://aclweb.org/anthology/P/P16/P16-1136.pdf.

[143] Cole Trapnell and Michael C. Schatz. Optimizing data intensive GPGPU computations for DNA sequence alignment. *Parallel Computing*, 35(8-9):429–440, 2009.

[144] Stephen Tratz and Eduard Hovy. ISI: Automatic classification of relations between nominals using a maximum entropy classifier. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 222–225, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1859664.1859713.

[145] Kateryna Tymoshenko and Claudio Giuliano. FBK-IRST: Semantic relation extraction using cyc. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 214–217, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1859664.1859711.

[146] Zia Ul-qayyum and Wasif Altaf. Paraphrase identification using semantic heuristic features. In *Research Journal of Applied Sciences, Engineering and Technology*, 2012. URL http://maxwellsci.com/jp/abstract.php?jid=RJASET&no=232&abs=52.

[147] Patrick Verga, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum. Multilingual relation extraction using compositional universal schema. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 886–896, 2016. URL http://aclweb.org/anthology/N/N16/N16-1103.pdf.

[148] Ellen M. Voorhees. Overview of the TREC 2002 question answering track. In *TREC*, 2002.

[149] Yogarshi Vyas and Marine Carpuat. Sparse bilingual word representations for cross-lingual lexical entailment. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1187–1197, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1142.

[150] Stephen Wan, Mark Dras, Robert Dale, and Cecile Paris. Using Dependency-based Features to Take the "Para-farce" out of Paraphrase. In *Australasian Language Technology Workshop*, pages 131–138, 2006.

[151] Di Wang and Eric Nyberg. A long short-term memory model for answer sentence selection in question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 707–712, 2015.

[152] Di Wang and Eric Nyberg. CMU OAQA at TREC 2015 LiveQA: Discovering the right answer with clues. In *TREC*, 2015.

[153] Di Wang and Eric Nyberg. A long short-term memory model for answer sentence selection in question answering. In *ACL*, 2015.

[154] Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 105–114, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0757-4. doi: 10.1145/2009916.2009934. URL http://doi.acm.org/10.1145/2009916.2009934.

[155] Mengqiu Wang and Daniel Cer. Probabilistic edit distance metrics for STS. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, pages 648–654, 2012.

[156] Mengqiu Wang and Christopher D. Manning. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1164–1172, 2010. URL http://dl.acm.org/citation.cfm?id=1873781.1873912.

[157] Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. What is the Jeopardy model? A quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 22–32, 2007. URL http://www.aclweb.org/anthology/D07-1003.

[158] Mengqiu Wang, Noah A Smith, and Teruko Mitamura. What is the Jeopardy model? A quasi-synchronous grammar for QA. In *EMNLP-CoNLL*, 2007.

[159] Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. Sentence similarity learning by lexical decomposition and composition. *arXiv:1602.07019*, 2016.

[160] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: scaling up to large vocabulary image annotation. In *International Joint Conference on Artificial Intelligence*, pages 2764–2770, 2011.

[161] John Wieting, Mohit Bansal, Kevin Gimpel, Karen Livescu, and Dan Roth. From paraphrase database to compositional paraphrase model and back. *Transactions of the Association for Computational Linguistics*, 3:345–358, 2015. ISSN 2307-387X. URL https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/571.

[162] John Wieting, Mohit Bansal, Kevin Gimpel, Karen Livescu, and Dan Roth. From paraphrase database to compositional paraphrase model and back. *Transactions of the Association for Computational Linguistics*, 3:345–358, 2015.

[163] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[164] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.270. URL http://dx.doi.org/10.1162/neco.1989.1.2.270.

[165] Lengdong Wu, Hua He, and Osmar R. Zaïane. Utility enhancement for privacy preserving health data publishing. In *Part II of the Proceedings of the 9th International Conference on Advanced Data Mining and Applications - Volume 8347*, ADMA 2013, pages 311–322, New York, NY, USA, 2013. Springer-Verlag New York, Inc. ISBN 978-3-642-53916-9. doi: 10.1007/978-3-642-53917-6_28. URL http://dx.doi.org/10.1007/978-3-642-53917-6_28.

[166] Wei Xu, Alan Ritter, Chris Callison-Burch, William B. Dolan, and Yangfeng Ji. Extracting lexically divergent paraphrases from Twitter. *Transactions of the Association for Computational Linguistics*, 2:435–448, 2014. URL http://www.cis.upenn.edu/~xwe/files/tacl2014-extracting-paraphrases-from-twitter.pdf.

[167] Adam Yala, Regina Barzilay, Laura Salama, Molly Griffin, Grace Sollender, Aditya Bardia, Constance Lehman, Julliette M Buckley, Suzanne B Coopey, Fernanda Polubriaginof, Judy E Garber, Barbara L Smith, Michele A Gadd, Michelle C Specht, Thomas M Gudewicz, Anthony Guidi, Alphonse Taghian, and Kevin S Hughes. Using machine learning to parse breast pathology reports. *bioRxiv*, 2016. doi: 10.1101/079913. URL http://biorxiv.org/content/early/2016/10/10/079913.

[168] Yi Yang, Wen-tau Yih, and Christopher Meek. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, 2015.

[169] Xuchen Yao, Benjamin Van Durme, Chris Callison-burch, and Peter Clark. Answer extraction as sequence tagging with tree edit distance. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 858–867, 2013.

[170] Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. Answer extraction as sequence tagging with tree edit distance. In *HLT-NAACL*, 2013.

[171] Wentau Yih, Ming-Wei Chang, Christopher Meek, and Andrzej Pastusiak. Question answering using enhanced lexical semantic models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 1744–1753, 2013. URL http://research.microsoft.com/apps/pubs/default.aspx?id=192357.

[172] Wenpeng Yin and Hinrich Schütze. Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 901–911, 2015.

[173] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. ABCNN: attention-based convolutional neural network for modeling sentence pairs. *CoRR*, abs/1512.05193, 2015. URL http://arxiv.org/abs/1512.05193.

[174] Wenpeng Yin, Sebastian Ebert, and Hinrich Schütze. Attention-based convolutional neural network for machine comprehension. In *Proceedings of the Workshop on Human-Computer Question Answering*, pages 15–21, San Diego, California, June 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W16-0103.

[175] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. Deep Learning for Answer Sentence Selection. In *NIPS Deep Learning Workshop*, 2014. URL http://arxiv.org/abs/1412.1632.

[176] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. Deep learning for answer sentence selection. *NIPS deep learning workshop*, 2014.

[177] Ashkan Zarnani, Petr Musilek, Xiaoyu Shi, Xiaodi Ke, Hua He, and Russell Greiner. Learning to predict ice accretion on electric power lines. *Eng. Appl. Artif. Intell.*, 25(3):609–617, April 2012. ISSN 0952-1976. doi: 10.1016/j.engappai.2011.11.004. URL http://dx.doi.org/10.1016/j.engappai.2011.11.004.

[178] Ying Zhang and Stephan Vogel. An efficient phrase-to-phrase alignment model for arbitrarily long phrase and large corpora. In *Proceedings of the Tenth Conference of the European Association for Machine Translation (EAMT-05)*, 2005.

[179] Jiang Zhao, Tian Tian Zhu, and Man Lan. ECNU: one stone two birds: ensemble of heterogenous measures for semantic relatedness and textual entailment. *International Workshop on Semantic Evaluation*, 2014.

[180] Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1604–1612, 2015.