Measuring Short Text Semantic Similarity with Deep Learning Models

Jun Ge

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEM AND TECHNOLOGY

YORK UNIVERSITY

TORONTO, ONTARIO

April 2018

Abstract

Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken, which is a subfield of artificial intelligence (AI). In other words, NLP is a way for computers to analyse, understand, and derive meaning from human language in a smart and automatic way. The development of NLP applications is challenging because computers traditionally require humans to "speak" to them in a programming language that is precise, unambiguous and highly structured, or through a limited number of clearly enunciated voice commands. In many NLP tasks, text semantic similarity or semantic matching, i.e., identifying the semantic meaning and inferring the semantic relations between two pieces of text, plays an important role in natural language processing and in text-related applications. The most widely used approach to identify semantic similarity is to use complex lexical, syntactic and semantic features to model text and then compare similarity between text pairs. However, there are still problems remaining. Traditional method bag-of-word/tf-idf model dominates in NLP, but it relies on word overlap to find similarities, this inherent term-specificity has limitation. This kind of methods are considered to efficient and easy to implement. However, the lack of common terms does not necessarily mean that two documents/sentences are not similar. Also, the relative word position is not considered in these models, which are important to the meaning of sentences. It requires a lot of feature engineering, which driven by intuition, experience and domain expertise. However, handcrafted features can be time-consuming, incomplete and over-specified. Deep neural networks, as a representation learning method, are able to discover from the training data the hidden structures and features at different levels of abstraction that are useful for the tasks. Motivated by this, in this thesis, we study different deep learning models and propose a new deep model for classifying semantic similarity between sentences.

In this thesis, we study the use of deep learning models, the state-of-the-art artificial intelligence (AI) method, for the problem of measuring short text semantic similarity. In particular, we propose a novel deep neural network architecture to identify semantic similarity for pairs of question sentence. In the proposed network, multiple channels of knowledge for pairs of question text can be utilized to improve the representation of text. We first model each sentence by two tied weights sub-networks in parallel, which is a siamese structure. Then a dense layer is used to learn a classifier for classifying duplicated question pairs. Through extensive experiments on the Quora test collection, our proposed approach has shown remarkable and significant improvement over strong baselines, which verifies the effectiveness of the deep models as well as the proposed deep multi-channel framework.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Jimmy Huang for the support of my study and research in the field of deep learning. He is always available, thoughtful and enthusiasm for the guidance on my study and research. I also would like to thank Professor Marina Erechtchoukova, who is my supervisor for a research project with the Shore Consulting Group Inc. and my supervisory committee member. Her attitude at work impressed and inspired me.

I would like to extend my gratitude to all faculty members who have offered support and assistance to my study: Prof. Aijun An, Prof. Xiaohui Yu, Prof. Zijiang Yang, Prof. Marin Litoiu, Prof. Peter Khaiter and so on.

I also would like to thank my friends Xing Tan, Stanley Liang and others in my laboratory and in York University for their help and friendship.

Last but not the least, I would like to thank my families: my parents who support me in my whole life; my husband Jeff Ye, who always understands and supports me, and encourages me to explore various potential; and my daughter, who makes me happy and thankful no matter how hard situation it is. Furthermore, I would welcome my new baby to this world who gives me more courage to finish this thesis.

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada, an NSERC CREATE award in ADERSIM, the York Research Chairs (YRC) program, an ORF-RE (Ontario Research Fund - Research

Excellence) award in BRAIN Alliance, which provides me with the training and financial support since 2015.

Contents

Chapter 1

Introduction

1.1 Short Text Semantic Similarity in NLP

Natural language processing (NLP) [1, 2] is the ability of a computer program to understand human language as it is spoken, which is a subfield of artificial intelligence (AI). More specifically, NLP is a way for computers to analyze, understand, and derive meaning from human language in a smart and automatic way. By using NLP algorithms or tools, developers can organize and structure knowledge to perform tasks such as automatic summarization [3–6], machine translation [7–9], named entity recognition [10, 11], relationship extraction [12], sentiment analysis [11, 13, 14], speech recognition [15–17], and topic segmentation [18], etc.

With the development of NLP applications, it is challenging for computers to perform well. The main reason is that computers do not have intelligence as human beings. They highly depend on the precise, unambiguous and structured programming language to perform various tasks, for example Java and C++; while human language is always imprecise or ambiguous. Also, the linguistic structure for human language is complex, which depends on many variables such as morphology, syntax, semantics, lexicon, pragmatics [19]. It is hard for program to model and simulate natural language. In addition, understanding human language may require external knowledge or facts which is trivial for human but demanding for computers.

Many problems in understanding natural language can be converted to comparing two pieces of text whether they are similar/dissimilar semantically or in some other ways, which has shown of promising in many NLP applications. In reality, text can be similar in two ways lexically and semantically. The main idea of lexical similarity is to determine whether two documents/sentences that contain common terms, while text semantic similarity or semantic matching is identifying the semantic meaning and inferring the semantic relations between two pieces of text. And the later plays an important role in natural language processing and in text-related applications. For example, in information retrieval area [20–33], it has been developed as taking input a query and retrieving relevant documents. Typically, the task is formalized as a matching problem between a query and a candidate document in existing work, in which the similarity between two pieces of text is computed by a ranking function derived from heuristic functions like tf-idf or deep neural network models; in duplicated question detection application, it converts into a classification problem which classifying whether two questions are similar or not; and in question answering system [34, 35], it is to identify semantic meaning and infer the semantic relations between two pieces of text – questions and answers, even though they have totally different lexical representations. Also other applications such as paraphrase identification [36–38], automatic conversation [39–41], etc. can be modeled as comparing two pieces of text.

The common challenge of these applications is that to understand the meaning of text is now transformed by comparing text semantic similarity of two pieces of text. The generic notation of such text comparing problem can be described as follow: given two pieces of text T_1 and T_2, we compute the degree of matching by a scoring function based on the representation of each text as follows:

$$match(T_1, T_2) = F(\phi(T_1), \phi(T_2)) \tag{1.1}$$

where ϕ is a function to map each text from the original space of symbolic words into some numeric space encoding various aspects of their relatedness, e.g. lexical, syntactic and semantic. F is the scoring function based on the interactions between them.

1.2 Traditional Methods to Text Semantic Similarity

Modeling the underlying semantic similarity especially between sentences has generated a great deal of research interest [42], many methods are developed in the literature[43, 44].

Traditional method such as bag-of-word/tf-idf model dominates in NLP, which relies on word overlap to find similarities. However, this method has its own inherent limitations [45]. This simple lexical similarity method is far from sufficient to compute text similarity since the lack of common terms does not necessarily mean that two documents/sentences are not similar. Also, the relative word position is not considered in these models, which are important to the meaning of sentences. The easy solution to this problem is to consider both lexically and semantically, which encode text pairs using many complex lexical, syntactic and semantic features and then compute various similarity measures [46] between the obtained representations. But this can result in new problems [47], such as time-consuming, domain specific, difficult to obtain and computational expensive, etc.

1.3 Deep Learning for Text Semantic Similarity

People have long dreamed whether machines might think and become intelligent. Nowadays, artifical intelligence(AI), which was first coined by John McCarthy in 1956 for an academic conference, is a thriving field with many practical applications and active research topics, such as speech recognition and natural language applications [16, 38], autonomous vehicles, medical diagnoses, computer vision [48], classification and so on. In early days, the field mostly solved problems that are difficult for human beings but relatively straight forward for computers – problems that can be described by a list of formal, mathematical rules. However, the fact is that the true challenge to AI is solving tasks that are easy for people to perform but hard for people describe formally – problems that human beings can solve intuitively and feel automatic [49].

To solve AI problems, people tried to hard-code knowledge about the world in formal languages, so that a computer can reason automatically by using logical inference rules. However, it is not easy to devise formal rules with enough complexity to accurately describe the world. The possible solution to this problem is that AI systems can extract patterns from raw data and acquire their own knowledge, which is similar to the working of human beings [49].

The capability of learning from raw data is known as machine learning. Simple machine learning algorithms, such as logistic regression, naive Bayes, SVM, etc., have been applied in many AI tasks [50–52] for decision-making, human-like deductive reasoning, inference and so on. However, their performance depends heavily on the representation of the data they are given. And this phenomenon is quite common in computer science even in daily life. Many AI tasks can be solved by designing the right set of features to a simple machine learning algorithm, even though it involves intensive labour work and time-consuming. But for many tasks, it is difficult to know what features should be extracted. Machine learning algorithms furthermore can be used to discover the representation of raw data itself, which goal is to separate the factors of variation that explain the observed data [49]. The quintessential example is the autoencoder algorithm. Still, this is not an easy work in many real-world applications since that many of the factors of variation influence every single piece of data, thus extracting high-level, abstract features can be very difficult.

Deep learning solves this problem well, which enables the computer to build complex concepts out of simpler concepts [49]. It is a subfield of machine learning, which is composed of a set of algorithms, inspired by the structure and learning approach of the human brain. A deep learning model is composed of several layers of neural networks. The two key elements are the concept of neurons and the networks of neurons. A single neuron receives the signal can process it and then signal neurons connected to it. It combines inputs with a set of weights, the input-weight products are summed and then passed through an activation function to determine whether and to what extent that signal progresses further through the network. The diagram for a single neuron is shown in figure 1.1. Another advantage of deep learning is the depth of networks enables the computer to learn a multistep computer program. This offers great power because networks can execute more instructions in sequence, and the representation in each layer helps to execute a program that can make sense of the input.

Representative architectures of deep learning models such as deep belief networks (DBN), convolutional neural network (CNN) and recurrent neural networks (RNN) have been applied to many research fields including computer vision [48], speech recognition [16], natural language processing [38], audio recognition, social network filtering, machine translation, bioinformatics, the effectiveness of which has been proven to be comparable to and in many cases superior to previous state-of-the-art or even human experts.

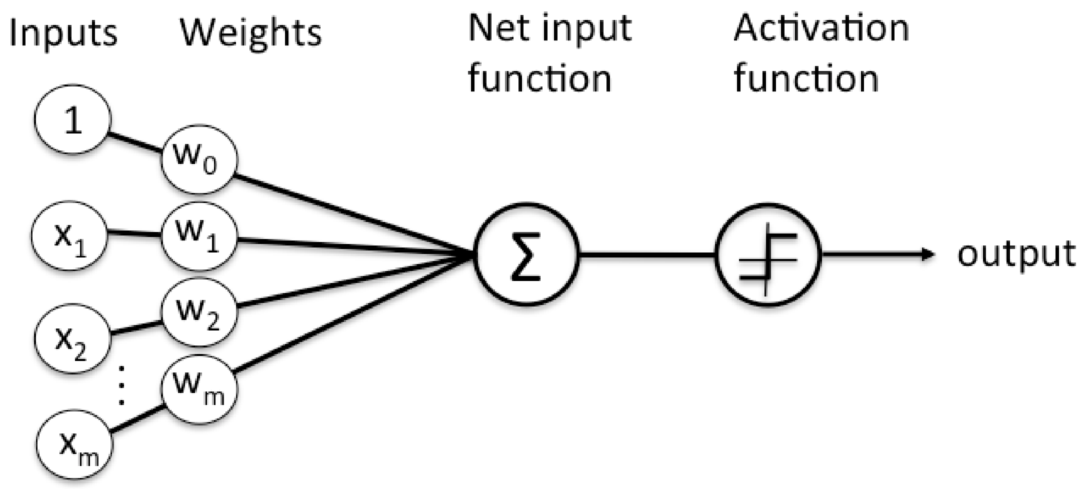

Figure 1.1: Diagram of A Neuron

Given the success of deep learning in these domains, it seems that deep learning should have a major impact on the problem text semantic similarity. Recent research shows that deep learning methods have achieved state-of-the-art results in many NLP areas. Motivated by this, in this thesis, we will investigates the effectiveness of several deep models for text semantic similarity. More specifically, we will study different deep learning models and propose a Multi-Channel Siamese BLSTM model for classifying semantic similarity. We then will study the performance of these models on Quora dataset for duplicate question pair detection, in which questions are sentences.

1.4 Research Background

Text similarity poses numerous challenges on many text related applications such as machine translation, paraphrase identification and question answering and so on. The aim of this thesis is to study the fundamental requirements and effective models for text similarity classification which can be easily applied to real-life applications. Specifically, a new architecture Multi-Channel Siamese architecture is proposed and applied for duplicate question detection for Quora dataset. This research work is funded by the Discovery grant from the Natural Sciences & Engineering Research Council (NSERC) of Canada and an NSERC CREATE award in ADERSIM.

1.5 Thesis Organization

In this thesis, we study the problem of short text semantic similarity from the perspective of deep learning models with word embedding. We explore the connection and difference of the traditional models as well as deep learning based models, and compare the effectiveness of them. Then, a Multi-Channel Siamese BLSTM model for duplicate question pair detection is proposed, in which vector representation for a question pair is learnt and used to automatically learn a deep scoring function.

The remainder of this thesis is organized as follows.

- In Chapter 2, we briefly introduce the problem of text semantic similarity and its application, and review the related literature including traditional methods to this problem as well as the recent research of deep learning methods that have achieved state-of-the-art results in this NLP area.

- In Chapter 3, we first introduce some basic concepts of deep learning and compare the recent state-of-the-art deep models for text semantic similarity in details. Based on the strength and the drawbacks of these deep models, we propose a Multi-Channel Siamese BLSTM model that is verified on duplicate question pair detection dataset.
- In Chapter 4, we detail the experimental dataset, settings and also the evaluation metrics.
- In Chapter 5, we present and analyse the experimental results of different deep models. And compare the performance of our model with some existing the state-of-the-art approaches.
- In Chapter 6, we conclude the paper with a discussion of our findings and future work.

Chapter 2

Literature Review

2.1 Research Problem

In most NLP tasks, e.g. sentiment classification, machine translation, question answer, the input of these systems is a natural language sentence. Formally, a typical NLP machine learning task involves classifying a sequence of tokens such as a sentence or a document, i.e. approximating a function $f_1(s) \in [0, 1]$ (where f_1 may determine a domain, sentiment, etc.). In addition, there is a large class of problems that are often harder and involve classifying a pair of sentences: $f_2(s_1, s_2) \in [0, 1] * c$ (where s_1, s_2 are sequences of tokens and c is a rescaling factor like $c = 5$).

Typically, the function f_2 denotes some sort of semantic similarity, that is whether (or how much) the two parameters "say the same thing". Essentially, this is a binary classification problem, in which 1 means s_1 and s_2 are similar while 0 means dissimilar. (However, the framework could be used to model different problems – like classify entailment or contradiction, topic relatedness, or answer selection in question answer system). The only difference is that we deal with different text pairs in different task. For example, in information retrieval tasks we typically deal with query-document pairs, while in question answering they are question-answer pairs. Being able to do so successfully is beneficial in many settings like in information retrieval, query suggestion, automatic summarization and image finding.

In order to find the scoring function f to compute the text similarity, a starting point is to find a way to represent the input (the input sentence), based on which f can operate. In the following, we will review related work on text semantic similarity according to the way of input representation, which includes traditional unsupervised model (e.g. *tf-idf* based model), supervised models, and we will treat the recent developed deep models separately.

2.2 Unsupervised Models

Being able to compute semantic similarity for short texts is beneficial in many applications. A large number of approaches have been proposed that use lexical matching and linguistic analysis. Methods for lexical matching aim to determine whether the words in two short texts look alike, e.g., in terms of edit distance [45], lexical overlap [53] or largest common substring [54]. These approaches tend to work for trivial cases.

Tf-idf [55] based method is a traditional and very popular representation to compare texts, such as query and document pair in information retrieval, with each other [21]. *Tf-idf* is short for term frequency-inverse document frequency, it is a statistical weight scheme used to assign weights to each term for a document in a collection or corpus, the formula is as shown in equation 2.1 [56].

$$\textit{tf-idf}_{t,d} = tf_{t,d} \times \textit{idf}_t = \frac{tf_{t,d}}{\log \frac{N}{\textit{df}_t}} \tag{2.1}$$

where $tf_{t,d}$ is referred as term frequency t for a document d; N is the total number of documents in a collection; $\textit{df}_t$ is the document frequency, which defined as the number of documents in the collection that contain a term t.

This weight increases partially to the number of times a term appears in a document but is offset by the frequency of the term in the collection, and for terms that do not occur in the document, its weight is zero. Thus this weight scheme can lead to different discriminating power of a term to the document.

After assigning each term a weight, a document can be seen as a fixed dimensional vector with one dimension corresponding to each term in the collection, together with a weight for each dimension which is given by *tf-idf*. Normally, a cosine function is then used to compute the similarity between two pieces of text. In this kind of method, it relies on word overlap to find similarities, but in very short texts, in which word overlap is rare, *tf-idf* based method often fails.

In order to enhance the basic *tf-idf* method, many complex lexical, syntactic and semantic features are also used to compute various similarity measures between the obtained representations. For example, in answer passage reranking [57] employs complex linguistic features, modelling syntactic and semantic information as bags of syntactic and semantic role dependencies and builds similarity and translation models over these representations. One of the major problems for this kind of methods is that the complex features are derived from different NLP tools/components which pretrained for other purposes. Some approaches depending on parse trees are restricted to syntactically well-formed texts, typically of one sentence in length. In addition, the error from these tools could also propagate to the text semantic similarity models that is very hard to resolve individually.

External resources like Wikipedia [1] or WordNet [2] have also been used as semantic feature to compare texts [58, 59]. Wikipedia is a free encyclopedia, written collaboratively by people who use it. It is structured around entities like well-known persons and organizations. A drawback of using dictionaries or WordNet is that high quality resources like these are not available for all languages, and proper names, domain-specific technical terms and slang tend to be underrepresented [60].

Another line of research is to use word embeddings, which are vector representations of terms computed from unlabeled data. Word embeddings represent terms/-words in a semantic space in which proximity of vectors can be interpreted as semantic similarity, thus attract a surge of interest. In [61], Mikolov et al. propose an algorithm called word2vec, in which two architectures to word2vec are developed, namely continuous bag-of-words (CBOW) and Skip-gram. Both are a variation of

[1] $https://en.wikipedia.org/wiki/Main_Page$

[2] $https://wordnet.princeton.edu/$

a neural network language model [62]. GloVe is an alternative way of getting word embeddings, which is proposed in [63]. Rather than being based on language models it is based on global matrix factorization. Some researchers are more focusing on how to use word embeddings. Kenter and Rijke [64] studied combining insights from methods based on external sources of semantic knowledge with word embeddings, in which an arbitrary number of word embedding sets can be incorporated. An important implication of the results is that distributional semantics has come to a level where it can be employed by itself in a generic approach for producing features that can be used to yield state-of-the-art performance on the short text similarity task, even if no manually tuned features are added that optimize for a specific test set or domain. Boom and Canneyt et al. [65] investigated several text representations as a combination of word embeddings and contributed a first step towards a hybrid method that combines the strength of dense distributed representations – as opposed to sparse term matching – with the strength of *tf-idf* based methods to automatically reduce the impact of less informative terms. In their toy experiment, they conclude that the hybrid approach outperforms the existing techniques in a toy experimental set-up, leading to the conclusion that the combination of word embeddings and tf-idf information might lead to a better model for semantic content within very short text fragments.

2.3 Supervised Models

Sanborn and Skryzalin [66] turn semantic similarity task into classification, and compare several machine learning methods like Nearest Neighbours, SVM, etc. as well as Recurrent NN and Recursive NN. The results show that the best performing model is a recurrent neural network with 100-dimensional word vectors.

Severyn and Moschitti [67] propose a framework for automatically engineering features for two important tasks of question answering: answer sentence selection and answer extraction, which applied SVM with tree kernels to shallow syntactic representation. An important feature of this approach is that it can effectively combine together different types of syntactic and semantic information, also generated additional automatic classifiers, e.g., focus and question classifiers.

There are some drawbacks of these supervised models. The first one is that the amount of labeled data is not enough. The second one is that handcrafted patterns and external sources of structured semantic knowledge cannot be assumed to be available in all circumstances and for all domains, and also these features are sometimes derived from external tools, the accuracy of which cannot be guaranteed.

2.4 Deep Learning Models

Recently, deep learning approaches have gained a lot of attention from the research community and industry for their ability to automatically learn optimal feature representation for a given task, while claiming state-of-the-art performance in many tasks in computer vision, speech recognition and natural language processing.

For the text semantic similarity problem under deep learning framework, normally word embedding technique is firstly used to represent each word/phrase of a input sentence. Then a deep learning model (e.g. CNN, Recursive NN and Recurrent NN) can be used to effectively capture the compositional process of mapping the meaning of individual words in a sentence and derive a continuous vector representation of the input sentence. With this vector representation, the similarity score of two input sentences can be computed. It has been shown that with large amount of training data, deep neural network are able to efficiently map the raw input of sentences to a low-dimensional vector representation, which preserves important syntactic and semantic aspects of the input sentence [68–71].

Depending on how you derive the sentence representation, existing deep matching models can be categorized into two types. The first one is siamese based neural network, in which the sentence vector representations of the two inputs are learnt respectively and then the sentence similarity of them is computed based on the sentence vectors. The other one is that it first builds the local interactions between two pieces of text based on some basic representations, and then uses deep neural networks to learn the hierarchical interaction patterns for matching. In the following of this section, we review relate work in these two streams and analyze their strength and drawbacks.

We first review examples of the first type of deep learning models, which include DSSM [72], C-DSSM [73] and ARC-I [35]. In DSSM, the authors develop a series of new latent semantic models with a feed forward neural network structure that project queries and documents into a common low-dimensional space where the relevance of a document given a query is readily computed as the distance between them. Cosine similarity function is used to compute the similarity score. In addition, word hashing technique is used to effectively scale up the semantic models to handle large vocabularies which are common in such tasks. Their results show that their best model significantly outperforms other latent semantic models on a real-world web data. In C-DSSM [73], a major difference to DSSM is that a convolutional neural network instead of feed forward neural network is used to learn a low-dimensional representation. ARC-I [35] also uses CNN to derive the text representation, but a multi-layer perceptron (MLP) is employed to compute the similarity score.

In addition, Severyn and Moschitti [47] propose a deep learning architecture for reranking short texts, where they learn the optimal representation of text pairs based on convolutional neural network architecture, and learn a similarity function based on simple neural network. Their architecture learns to extract and compose n-grams of higher degrees, thus allowing for capturing longer range dependencies. Their experimental findings show that this deep learning model greatly improves on the previous state-of-the-art systems, but requires no external parsers or resources.

Mueller and Thyagarajan [74] propose a siamese recurrent architectures for learning sentence similarity. Their purpose is that learning a semantically structured representation space then applying simple metrics to capture sentence similarity.

Neculoiu, Versteegh and Rotaru [75] present a siamese deep architecture for learning a similarity metric on variable-length character sequences.

For the second type of deep learning models, DeepMatch [76], ARC-II [35] and MatchPyramid [77] are representative. These models first build the local interactions between two texts based on some basic representations, and then uses deep neural networks to learn the hierarchical interaction patterns for matching. A novel deep architecture is proposed for this kind of matching problem in DeepMatch [76], which can sufficiently explore the nonlinearity and hierarchy in the matching process. In their experiments, it has been empirically shown to be superior to various inner product based matching models on real-world data sets (Baidu Zhidao dataset and Weibo-Comments).

MatchPyramid is inspired by the success of convolutional neural network in image recognition, where neurons can capture many complicated patterns based on the extracted elementary visual patterns. MatchPyramid models text matching as the problem of image recognition. In particular, a matching matrix whose entries represent the similarities between words is constructed and viewed as an image, and then a convolutional neural network is utilized to capture rich matching patterns in a layer-by-layer way.

One of the main problems of these existing models is that only the word information in the original sentence is used, regardless of external knowledge or fact of words in the sentences. However, understanding a natural sentence is hard and it always requires syntactic information, background knowledge or fact, and so on. Inspired by this, we propose a multi-channel siamese framework for the problem of text semantic similarity, whose architecture is mostly similar to Neculoiu's [75], but we apply it on word sequences and utilize different types of information under a multi-channel framework. In particular, we explore the use of part-of-speech (POS) information to enhance the problem of semantic text matching as starting point. We present the details of our model in Chapter 3.

2.5 Challenges to Deep Models

The performance of a deep neural network models could be affected by a number of key factors. Deep learning currently works best when there are large data sets. This is because deep learning lacks a mechanism for learning abstractions through explicit, verbal definition [78]. To get powerful abstraction from a deep learning model, a complex neural networks may be needed, which is usually expressed through the number of parameters. Thus, the more parameters need to be tuned, the more data are required which can be ranged from thousands to billions of labeled examples. Overfiting is also a major problem for deep learning model. As aforementioned, deep learning model usually has large number of parameters, so when there are not enough training data or noise to some extent in training data, it is easy for a model to generate a good fit to the current available data without capturing any genuine insights into the underlying phenomenon. Other factors such as how to integrate prior knowledge into a deep learning system is still not straightforward. The dominant approach in deep learning is self-contained and isolated from other potentially usefully knowledge [78]. The learning process is conducted by learning the relations between inputs and outputs in which knowledge represented largely opaque between potential "features".

In order to solve these problems, we apply some strategies and propose a new model. Firstly, we use validation data. The total dataset is split into a training dataset, a testing dataset and a validation dataset, while we use validation dataset to choose the learnt best model. Secondly, we apply L_2 regularization method to prevent overfitting, by which a penalty is adding for model complexity or extreme parameter values. L_2 regularization tends to yield a dense solution, where the magnitude of the coefficients are evenly reduced. Lastly, a multi-channel framework is proposed, which is trying to utilize prior knowledge in deep learning model.

Chapter 3

Deep Models for Text Semantic Similarity

The advent of more sophisticated algorithms, computational powers from GPUs becoming cheaper and the availability of large amount of data led to what can be called “a renaissance for Deep Learning”. Deep Learning for NLP is relatively new as compared to its usage in computer vision which employs it to process images and videos.

In this chapter, we first introduce a concept called word embedding, which is usually considered as a starting point of deep learning for NLP. Then, the basic paradigm of using deep learning techniques and some representative deep models are discussed for better understanding of its usage in text semantic similarity. Lastly, we detail the proposed multi-channel siamese deep framework, in which prior knowledge can be incorporated as multiple type of inputs for better modelling of text similarity.

3.1 Word Embeddings

3.1.1 Brief history of word embeddings

Vector space models have been used in distributional semantics since the 1990s. The aim is to quantify and categorize semantic similarities between linguistic items based on their distributional properties in large samples of language data. However, these traditional techniques treat words as atomic symbols, and represent them as arbitrary and unique indexes for a vocabulary. These unique and discrete indexes furthermore lead to data sparsity, which means that we may need more data in order to successfully train a model. And since there is no notion of similarity between words, it is very hard for a model to leverage learned knowledge encoded implicitly.

Since then, a number of models were developed for estimating continuous representations of word, which are usually named as word embeddings. The term word embeddings was originally coined by Bengio et al. in 2003 who trained them in a neural language model [62]. However, it was Mikolov et al.(2013), who really attributes to the eventual popularization of word embeddings through the creation of word2vec, a toolkit enabling the training and use of pre-trained embeddings

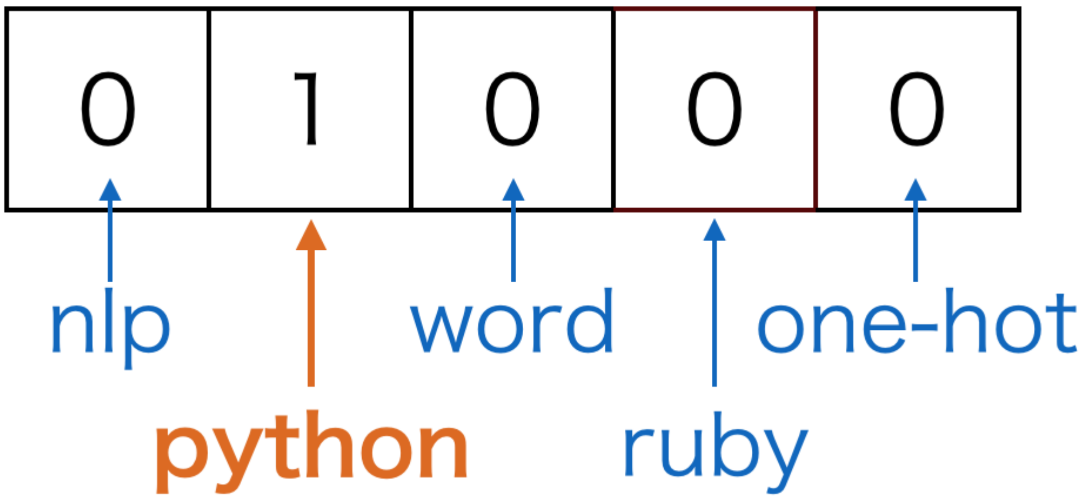

Figure 3.1: An example for one-hot representation

[61, 79]. A year later, Pennington et al. introduced us to GloVe [63], a competitive set of pre-trained embeddings. From then on, word embeddings used as underlying input representation has gained tremendous success in NLP tasks such as text classified, machine translation, syntactic parsing and sentiment analysis.

3.1.2 What is word embedding?

Before introducing word embedding, we first illustrate the one-hot representation. In particular, one-hot representation is a representation method in which only one dimension in a vector is set to 1 and the others are 0 in the vector. By setting 1 or 0 for each dimension, it means "that word or not".

For example, we show the word "python" as one-hot representation. Here, the vocabulary which is a set of words is five-words(nlp, python, word, ruby, one-hot). Then the following vector describes the word "python" [1]:

[1]http://ahogrammer.com/2017/03/22/why-is-word-embeddings-important-for-natural-language-processing/

Although one-hot representation is simple, there are several weak points: it is impossible to obtain meaningful results with arithmetic between vectors. For example, we take an inner product to calculate similarity between words. In one-hot representation, different words are 1 in different places and the other elements are 0. Thus, the result of taking the dot product between the different words is 0. This is not a useful result.

A word embedding is a learned representation for words, which are real number vectors whose relative similarities correlate with semantic similarity, which is usually a by-product of training a neural language model. Word embedding is also the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP), which maps words to a relatively lower dimension vectors as $W : words \rightarrow R^n$. Deeper linguistic theory behind this approach, namely the "distributional hypothesis" by Zellig Harris [80], is that words that have similar context will have similar meanings.

The learning process is either joint with the neural network model on some task, or is an unsupervised process, using document statistics. The learned representations can be used both as an end in itself (for computing similarities between terms), and as a basic representation for downstream NLP tasks like text classification, document clustering, sentiment analysis and so on.

3.1.3 Why word embedding?

One obvious advantage of word embedding is the aforementioned real number representation. Many state-of-the-art models are incapable of processing strings or plain text in their raw form. They require numerical inputs to perform sort of jobs such as classification, clustering regression etc. With the huge amount of text data available nowadays, it is imperative to find a way to represent symbolic data and extract knowledge out of it.

Using dense and low-dimensional vectors is computationally efficient. Word embedding involves that taking words from a space with one dimension per word to a continuous vector space with much lower dimension. Traditional sparse word representations, such as BOW, often have thousands or millions of dimensions. While the size of vector representation space is specified as part of the model and usually tens or hundreds of dimensions, such as 50, 100, or 300 dimensions.

Another advantage of dense vector representation is that it achieves a level of generalization which is not possible with classical language models such as n-gram model which works in terms of discrete units that have no inherent relationship to one another [81]. In addition, researchers find that the learned representations capture meaningful syntactic and semantic regularities in a very simple way. Specifically, the regularities are observed as constant vector offsets between pairs of words sharing a particular relationship. This allows vector math and reasoning based on the offsets between words. For example, there seems to be a constant male-female difference vector:

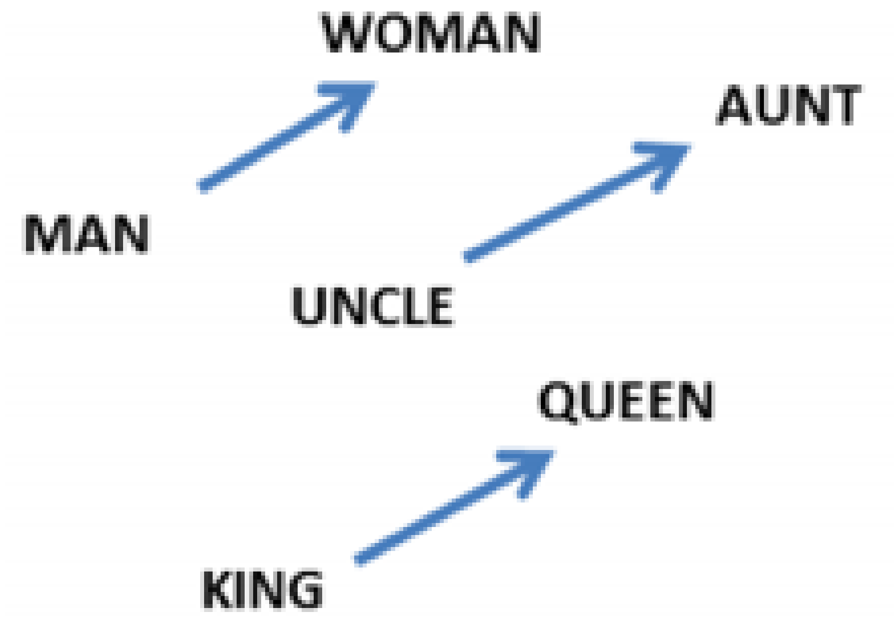

Figure 3.2: Constant vector offsets between pairs of words

$$W(\text{“}woman\text{”}) - W(\text{“}man\text{”}) \simeq W(\text{“}aunt\text{”}) - W(\text{“}uncle\text{”}) \tag{3.1}$$

$$W(\text{“}woman\text{”}) - W(\text{“}man\text{”}) \simeq W(\text{“}queen\text{”}) - W(\text{“}king\text{”}) \tag{3.2}$$

3.1.4 Word embedding models

The different types of word embeddings can be broadly classified into two categories: frequency based embedding and prediction based embedding. Frequency based embedding is more about traditional vectorization methods such as count vector model, *tf-idf* vector model and co-occurrence vector model. In this section, we primarily introduce prediction based embedding. And assumes that you have knowledge of how a neural network works and mechanisms by which weights in a neural network are updated.

Classic neural language model

The classic neural language model proposed by Bengio et al. in 2003, they are one of the first to introduce what we now refer to as a word embedding, a real-valued word vector in R. The architecture of their model is shown in figure 3.3. There are three basic layers which can still be found in current neural language and word embedding models.

- 1. Embedding layer: a layer that generates word embeddings by multiplying an index vector with a word embedding matrix;
- 2. Intermediate Layer(s): one or more layers that produce an intermediate representation of the input, e.g. a fully-connected layer that applies a non-linearity to the concatenation of word embeddings of n previous words;
- 3. Softmax Layer: the final layer that produces a probability distribution over words in V.

The training set is a sequence $w_1 \dots w_T$ of words $w_t \in V$, where the size of vocabulary V is $|V|$. The objective is to learn a good model $f(w_t, \dots, w_{t-n+1}) = P(w_t|w_1^{t-1})$, in the sense that it gives high out-of-sample likelihood. The objective of their model is to maximize :

$$J_\theta = \frac{1}{T} \sum_{t=1} log f(w_t, w_{t-1}, \dots, w_{t-n+1}) \tag{3.3}$$

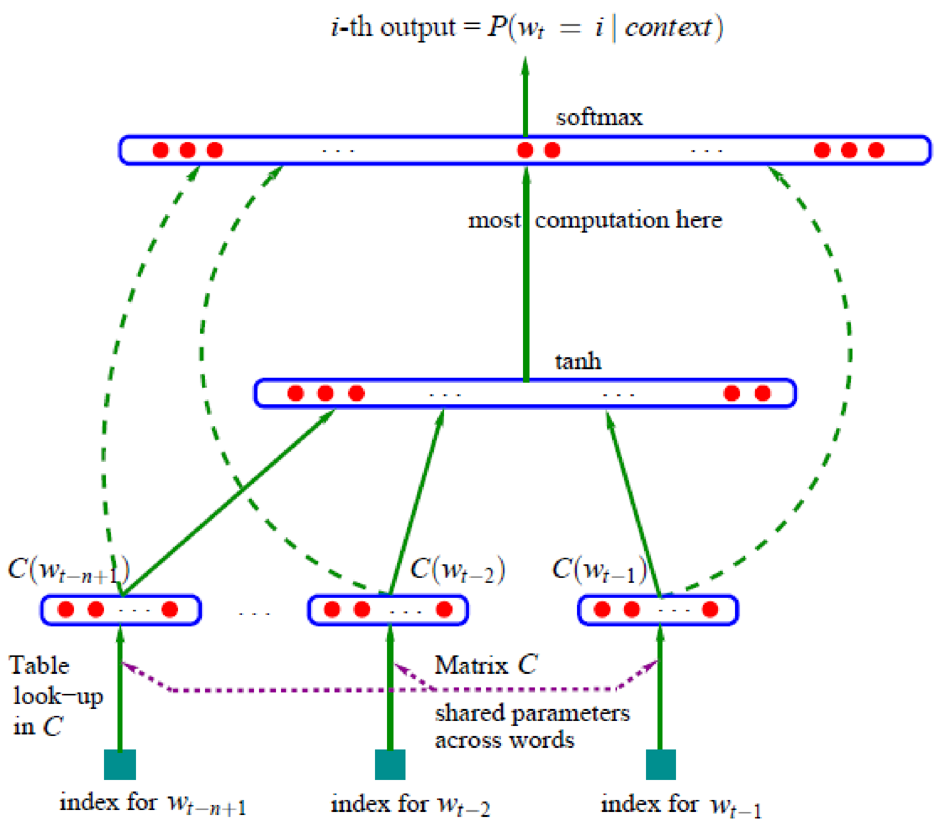

FIGURE 3.3: Classic neural language model [62]

(for simplicity, the regularization term is omitted) $f(w_t, w_{t-1}, \ldots, w_{t-n+1})$is the output of the model, i.e. the probability$P(w_t|w_{t-1}, \ldots, w_{t-n+1})$ as computed by the softmax, where n is the number of previous words fed into the model.

Word2vec

As aforementioned, it is until Mikolov et al. introduced word2vec toolkits, word embedding became popular to the NLP community. Word2vec is a combination of related models that are used to produce word embeddings, namely Continuous bag of words (CBOW) and Skip-gram model. Both of these models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words.

Continuous bag of words (CBOW)

The CBOW model predicts the current word based on the context. Its architecture is similar to classic neural network model, where the non-linear hidden layer is removed and the projection layer is shared for all words; thus, all words get projected into the same position and their vectors are averaged. The architecture is shown in figure 3.4. The objective function is as follows:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} logp(w_t|w_{t-n}, \ldots w_{t-1}, w_{t+1}, \ldots, w_{t+n}) \tag{3.4}$$

We can see that this model uses continuous distributed representation of the context and the order of words does not influence the projection. That is why they call this model a continuous bag-of-words model.

There are some major difference comparing to other models. In language model, n previous words are fed into the model, while CBOW receives a window of n words around the target word w_t at each time step t. In classic neural network model, it assumes a fixed dimensional input and extract a fixed number of features: each feature is represented as a vector and the vectors are concatenated. In this way, each region of the resulting input vector corresponds to a different feature. However, we usually do not know the number of features in advance. CBOW model can meet this requirement.

Skip-gram Model

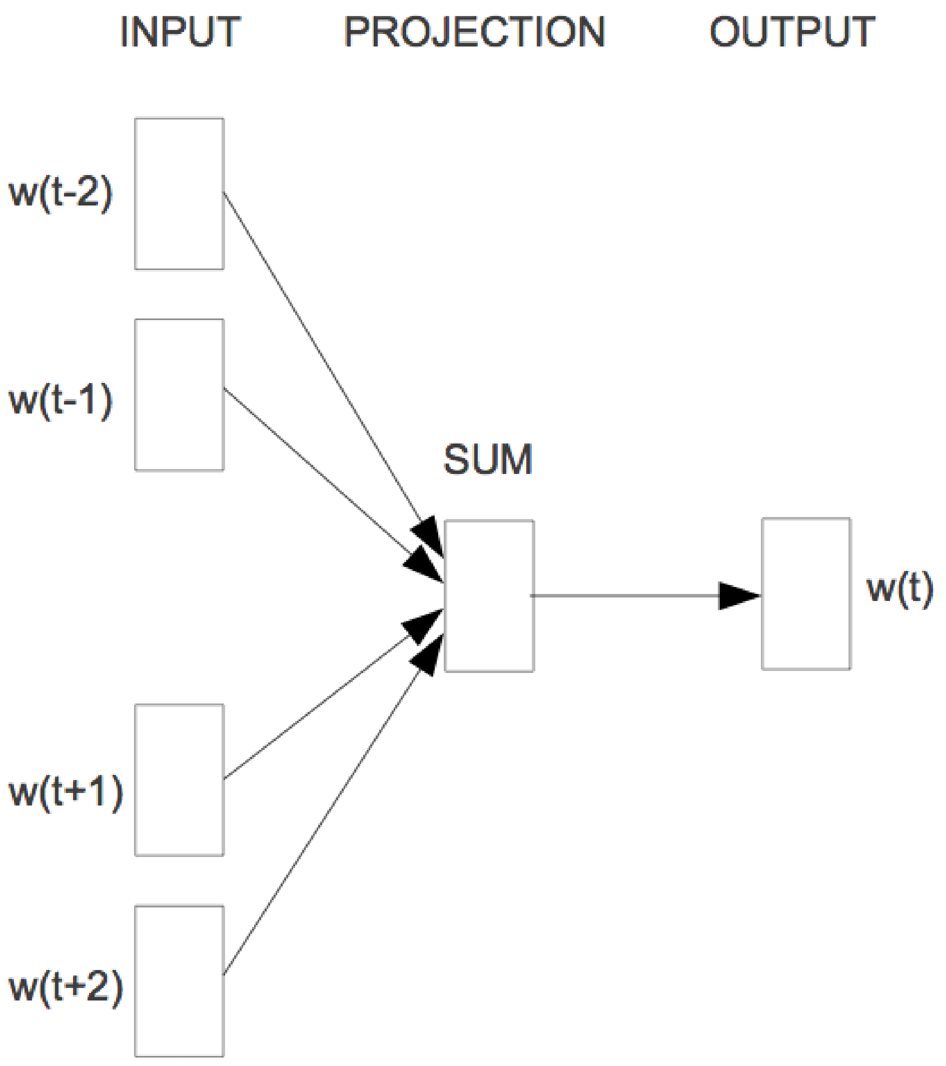

FIGURE 3.4: Continuous bag of words (CBOW) [61]

Skip-gram model follows the same topology as of CBOW in Figure 3.5, but in reverse direction of CBOW, which is using the center word to predict the surrounding words. Specifically, the neural network is trained to do the following job: given a word in the middle of a sentence, look at the words nearby and pick one at random. The network is going to tell us the probability for every word in the vocabulary of being the "nearby word". After training, the weights between the input and the hidden layer are taken as the word vector representation.

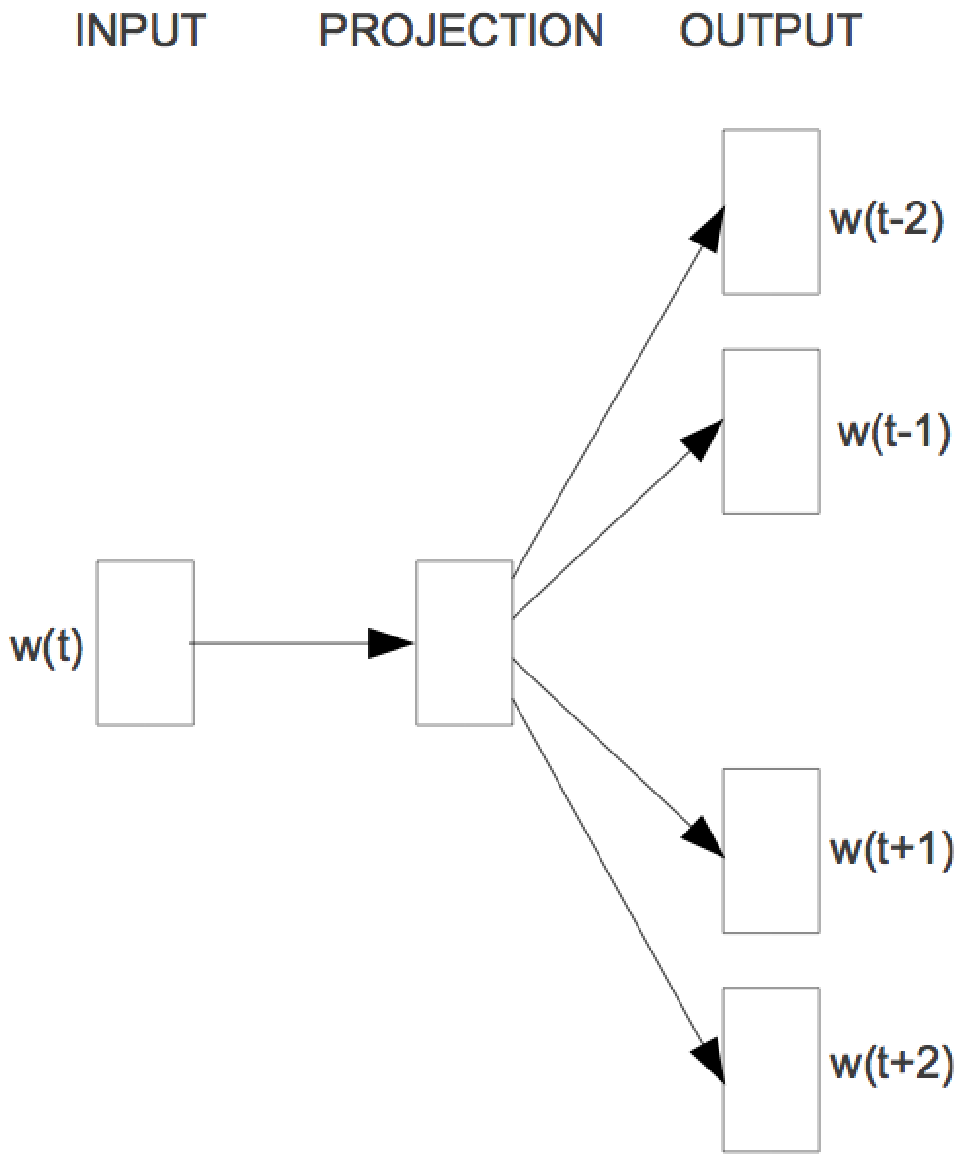

FIGURE 3.5: Skip-gram [61]

The objective of skip-gram is:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T}\sum_{-n\leqslant j\leqslant n,\neq 0} logp(w_{t+j}|w_t) \tag{3.5}$$

GloVe

In contrast to word2vec, GloVe [63] seeks to make explicit what word2vec does

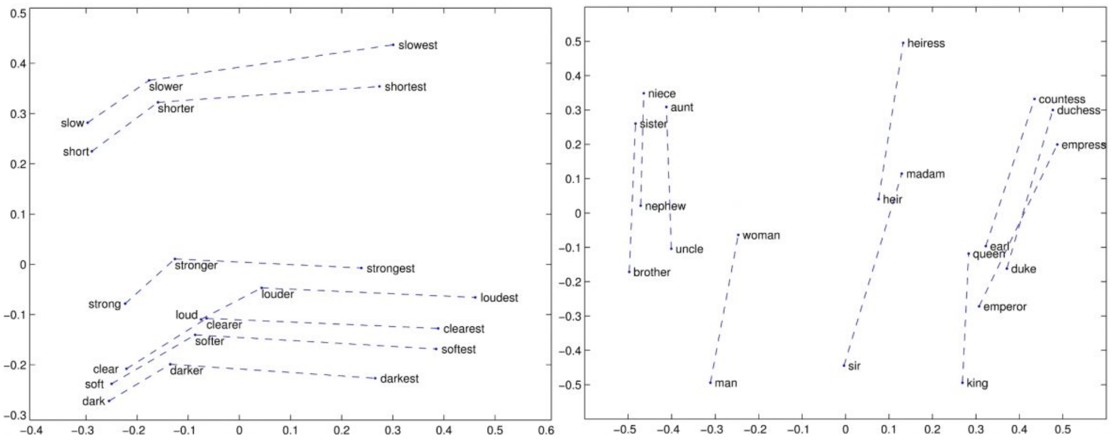

FIGURE 3.6: Vector relations captured by GloVe

implicitly: encoding meaning as vector offsets in an embedding space – seemingly only a serendipitous by-product of word2vec – is the specified goal of GloVe.

To be specific, the creators of GloVe illustrate that the ratio of the co-occurrence probabilities of two words (rather than their co-occurrence probabilities themselves) is what contains information and so look to encode this information as vector differences. GloVe is designed in order that such vector differences capture as much as possible the meaning specified by the juxtaposition of two words as shown in 3.6.

For this to be accomplished, they propose a weighted least squares objective J that directly aims to reduce the difference between the dot product of the vectors of two words and the logarithm of their number of co-occurrences:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - logX_{ij})^2 \tag{3.6}$$

where w_i and b_i are the word vector and bias respectively of word i, w_j and b_j are the context word vector and bias respectively of word j, X_{ij} is the number of times word i occurs in the context of word j, and f is a weighting function that assigns relatively lower weight to rare and frequent co-occurrences.

As co-occurrence counts can be directly encoded in a word-context co-occurrence matrix, GloVe takes such a matrix rather than the entire corpus as input.

These results are in contrast to the general consensus that word embeddings are superior to traditional methods. Rather, they indicate that it typically makes no difference whatsoever whether word embeddings or distributional methods are used. What really matters is that the hyperparameters are tuned and that the appropriate pre-processing and post-processing steps utilized. In this thesis, we mainly use the GloVe embedding pre-trained from its official website [2].

3.2 Basic Deep Learning Concepts and Models

Deep learning has various definitions. One of the definitions is that it is a class of machine learning techniques which exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification. There are two key aspects in this definition: 1) models consisting of multiple layers and 2) methods for learning features at higher, more abstract layers [82]. To understand deep learning well, we will introduce the basic concepts of machine learning first.

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

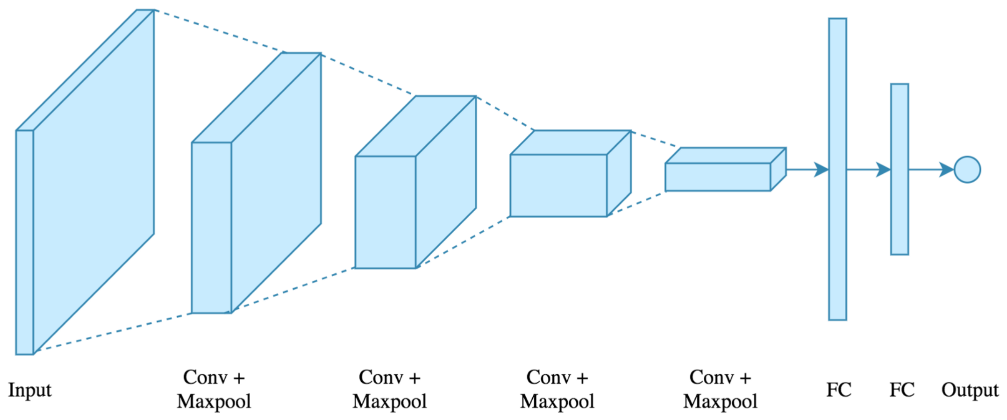

FIGURE 3.7: Architecture of CNN

In neural networks, a neuron forms the basic structure. It receives an input, processes it and generates an output which is either sent to other neurons for further processing or it is the final output.

3.2.1 Convolutional Neural Network(CNN)

Convolutional Neural Network (CNN) is a class of deep, feed-forward artificial neural networks which were inspired by biological processes in that individual neurons respond to stimuli only in a restricted region. CNN is now commonly used model on image related problem. It is also successfully applied to recommender systems, natural language processing and more.

CNN often has a series convolutional and pooling layers and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture is shown in figure 3.7.

There are four core building blocks in CNN.

1. Convolutional Layer

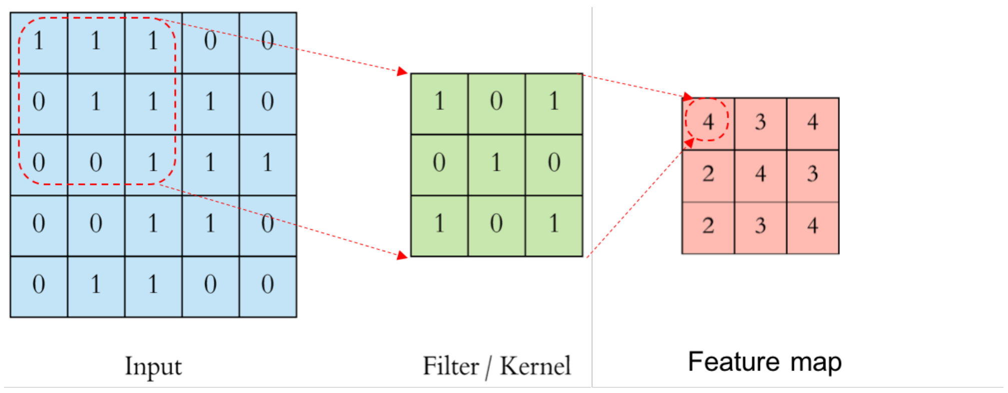

FIGURE 3.8: Convolution operation

Convolution is a mathematical operation on two functions to produce a third function. Here in convolutional layer, it is applied on the input data using a convolution filter to produce a feature map. The convolution operation is performed by sliding convolution filter over the input. At each location, do element-wise multiplication and sum the result which goes into the feature map.

2. Non-linearity

Non-linear activation function can make any kind of neural network to be powerful, which is achieved by passing the weighted sum of its inputs through an activation function. So does CNN. This is because linear combination of linear functions will create nothing qualitatively different. No matter how many layers applied in architecture, just act like one single layer.

There are three most common activation function used, sigmoid, softmax and rectifier (relu). However, according to research [83], relu has been found to perform better in most situations. The formulas for these activation functions are as follow:

Sigmoid: takes a real-valued input and squashes it to range between 0 and 1.

$$\sigma(Z) = \frac{1}{1+e^{-Z}} \tag{3.7}$$

Softmax: takes a real-valued input and squashes it to range between 0 and 1 that add up to 1.

$$a_j^L = \frac{e^{Z_j^L}}{\sum_k e^{Z_k^L}} \tag{3.8}$$

Relu: takes a real-valued input and thresholds it at zero (replaces negative values with zero).

$$f(x) = max(0, x) \tag{3.9}$$

3. Pooling or sub sampling

After convolutional operation, a pooling layer (also called sub sampling) is performed. It can be used to reduce the dimensionality of each feature map but still retains the most important information. There are several benefits by pooling. The most obvious is to make the input dimension smaller. Further, it enables to reduce the number of parameters computation in network, which furthermore reduce and control overfitting.

Pooling can be applied separately on each feature map, reducing the size of height and width, but keeping the depth intact. There are also different types of pooling, Max, Average and Sum. In practice, Max pooling has been

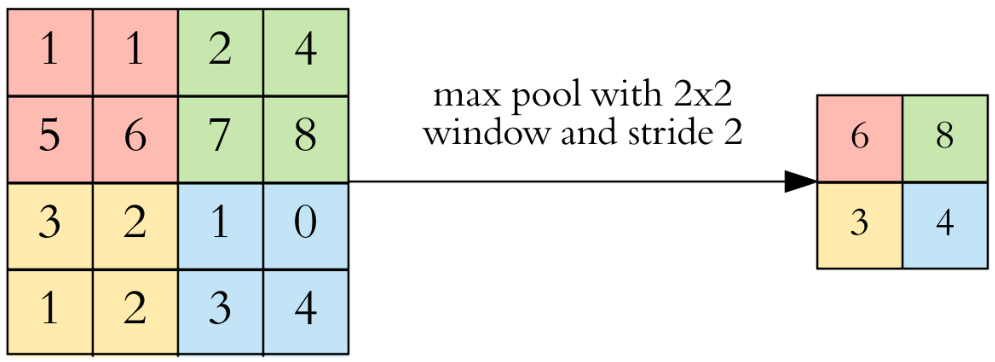

FIGURE 3.9: Example for Max Pooling in CNN

shown to work better. An example of Max pooling with 2*2 window and stride 2 is shown in figure 3.9.

4. Fully connected layer

   Traditionally, a fully connected layer is a Multi Layer Perceptron. Just as its name implies, neurons have full connections to all activations in the previous layer. It is worth noting the differences between a fully connected layer and a convolutional and pooling layer. From connectivity perspective, they are different that, in convolution layer neurons are connected only to a local region in the input. Another difference is their purposes. Essentially, the convolutional and pooling layers are providing a meaningful, low-dimensional and high-level features of the input, while fully connected layer is learning a function based on training data and then classifying unseen data.

3.2.2 Recurrent Neural Networks(RNN)

In a traditional neural network, it has no notion of order in time, and the only input it considers is the current example it has been exposed to and all inputs are assumed to be independent of each other. So it is unclear that a traditional neural network could leverage some prior knowledge about the data, for example, predicting the next word in a sentence which would be better to know the words before it. Recurrent Neural Networks (RNN) address this issue which makes use of sequential information.

3.2.3 Long-Short Term Model(LSTM)

Like standard RNNs, LSTM model naturally suits for variable lengthy input like sentences. LSTM sequentially updates a hidden-state representation by introducing memory cell units which can store information across input and three gates which control the flow of information through time steps. A typical repeated module in LSTM is shown in figure 3.10.

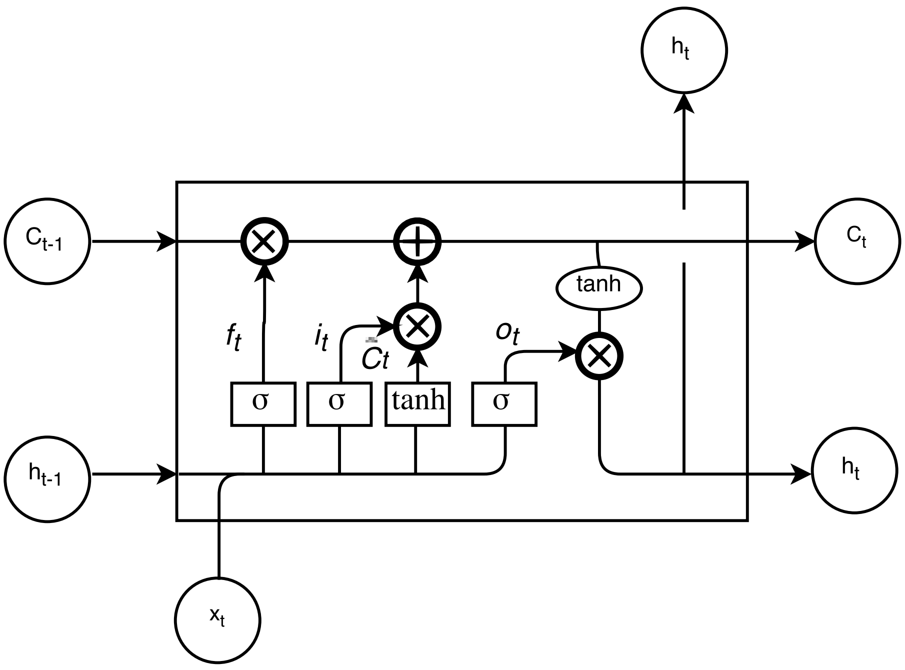

FIGURE 3.10: Typical LSTM Architecture

Long-short Term Memory (LSTM) is a recurrent neural networks (RNN) architecture that has been designed to address the vanishing and exploding gradient problems of conventional RNNs. Unlike feedforward neural networks, RNNs have cyclic connections making them powerful for modeling sequences. They are applied for sequence labeling and sequence prediction tasks and gained promising results [84].

All recurrent neural networks have the form of a chain of repeating modules of neural network, but this repeating module has a simple structure. As the gap between the relevant information and the point where it is needed to become very large, RNNs becomes unable to learn to connect the information. However, in a standard LSTM, the structure is more complex. Except the repeating module, there is a special hidden units called memory blocks(in figure 3.10). Each memory block is composed of four elements: (1) a memory cell C_t (e.g. a neuron) with a self-connection, (2) an input gate i_t to control the flow of input signal into the neuron, (3) an output gate o_t to control the effect of the neuron activation on other neurons, and (4) a forget gate f_t to allow the neuron to adaptively reset its current state through the self-connection[85]. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates[84].

The computations for each updates at $t \in \{1, ..., T\}$ in an LSTM are as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.10}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.11}$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{3.12}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.13}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.14}$$

$$h_t = o_t * \tanh(C_t) \tag{3.15}$$

Where W_i, W_f, W_c, and W_o are weight matrices, and b_i, b_f, b_c and b_o are bias-vectors.

3.2.4 Bidirectional LSTM (BLSTM)

The disadvantage of LSTM is that it is only able to make use of previous elements. However, the output may not only depend on previous elements in a sequence, but also future elements. BLSTM utilize both previous and future elements by stacking two LSTMs on top of each other.

The principle of BLSTM is to split the neurons of a regular LSTM into two directions, one for positive time direction (forward states), and another for negative time direction (backward states). Those two states' output are not connected to inputs of the opposite direction states. Figure 3.11 shows the basic structure of BLSTMs. By using two time directions, input information from the past and future of the current time frame can be used unlike standard LSTM which requires

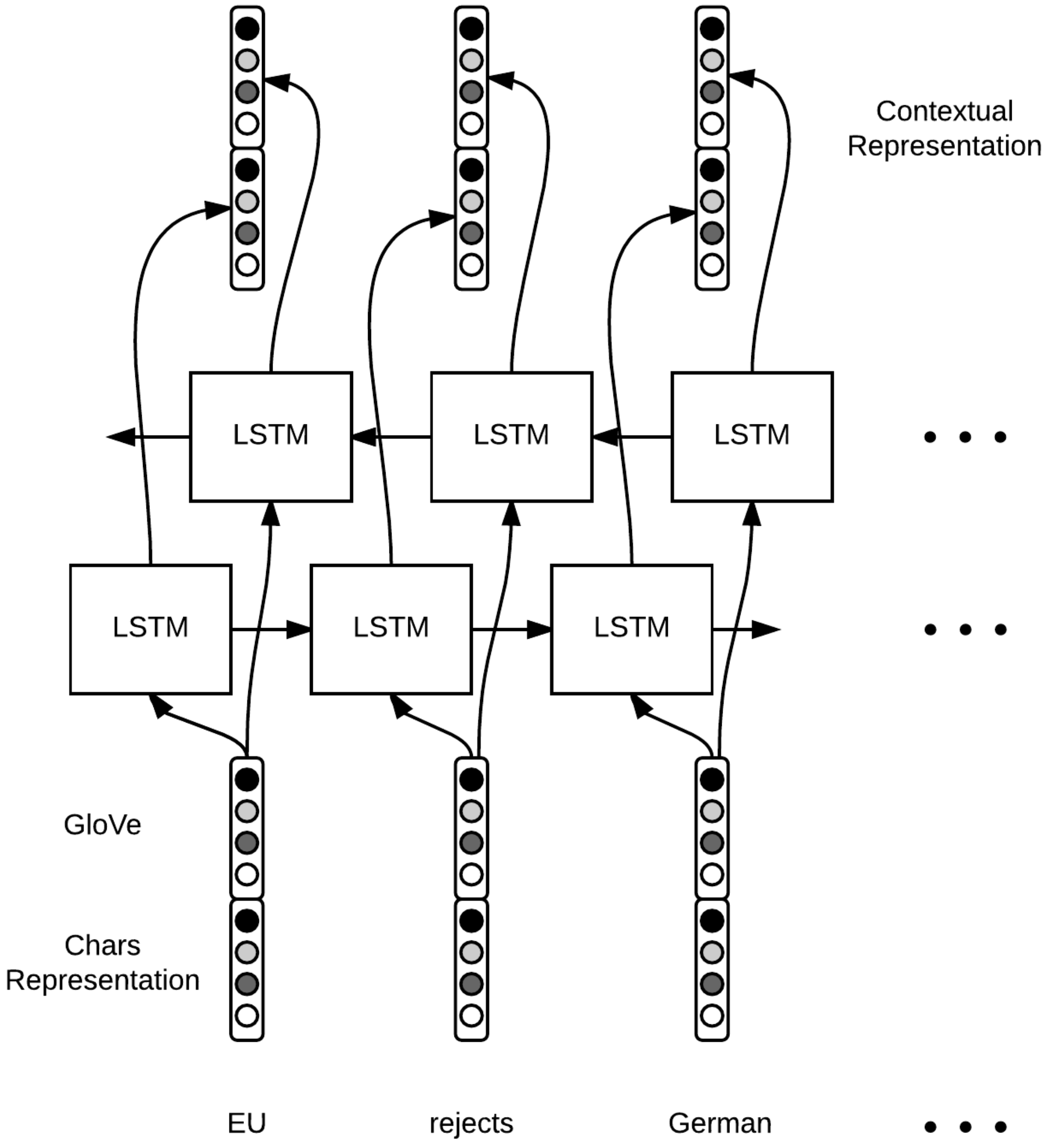

FIGURE 3.11: Structure of BLSTM

the delays for including future information. The output is then computed based on the hidden state of both LSTMs.

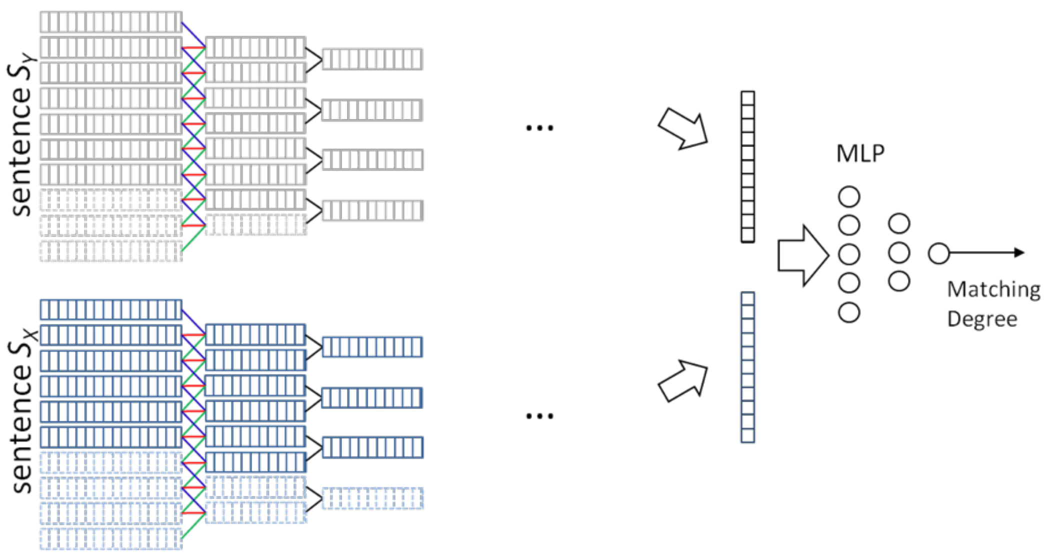

FIGURE 3.12: Architecture for ARC-I Model

3.3 Advanced Deep Learning Models

In this section, we will introduce some state-of-the-art deep learning models, which have gained significant performance in NLP tasks.

3.3.1 Architecture-I(ARC-I)

ARC-I is a convolutional based deep model proposed by Baotian Hu etc. [35]. They believe that natural language sentences are complicated both sequential and hierarchical. It is important to have an algorithm that can capture the internal structures. ARC-I adopts a siamese convolutional based architecture, which is shown in figure 3.12 . It takes word embeddings as input and summarizes meaning of a sentence by the layers of convolution and pooling. With the large feature map design, this model can adequately model the rich structures in the composition of words. After finding the representation for each sentence, then the representations for the sentence pair are compared by a multi-layer perceptron(MLP).

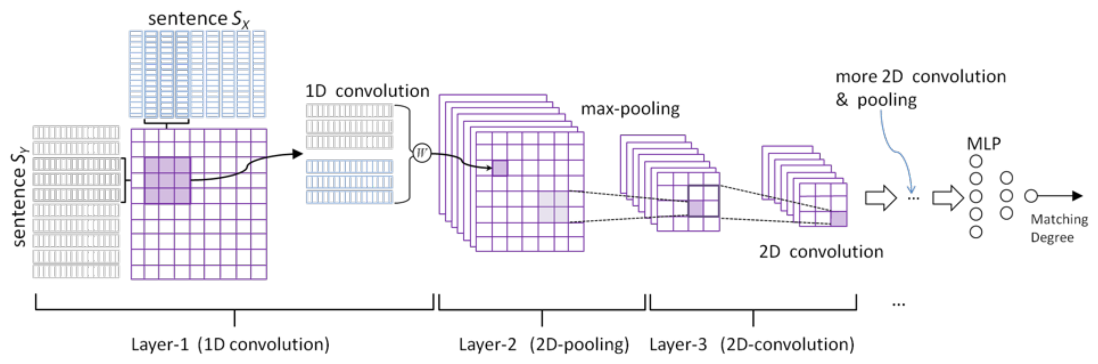

FIGURE 3.13: Architecture for ARC-II Model

3.3.2 Architecture-II(ARC-II)

Baotian Hu etc. also proposed Architecture-II(ARC-II) in [35]. The architecture is shown in figure 3.13.

The main idea is that letting the two sentences meet before their own high-level representations mature while still retaining the space for the individual development of abstraction of each sentence. In layer-1, the 1D convolution preserves all possible combinations of segments in two sentences, which is forming the interaction space between two sentence. After that more layers of 2D convolution and 2D max pooling can be employed to get the higher level abstract of representation. Finally, MLP is used to predict matching degree.

3.3.3 MatchPyramid

Inspired by the success of convolutional neural network in image recognition, Liang Pang, etc. [77] proposed an architecture, as illustrated in figure 3.14 , which viewed text matching as image recognition. This is an interaction based model. It takes

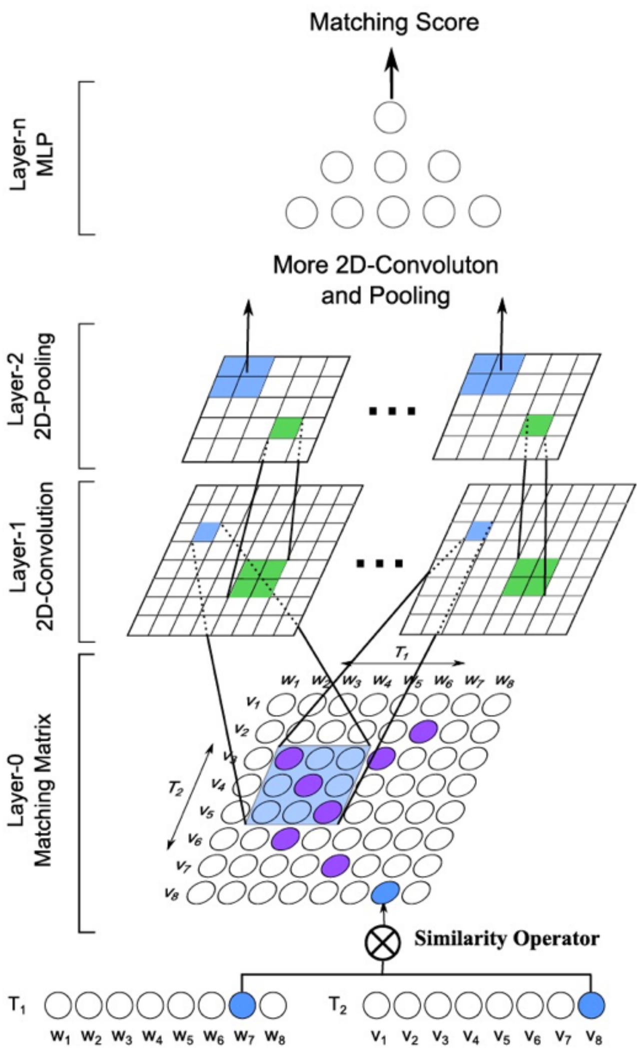
Matching Score
Layer-n MLP
More 2D-Convoluton and Pooling
Layer-2 2D-Pooling
Layer-1 2D-Convolution
Layer-0 Matching Matrix
T_1
w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8
T_2
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8
Similarity Operator
T_1 w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8
T_2 v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8

FIGURE 3.14: Architecture for MatchPyramid Model

into account rich interaction structure in the text matching process such as interactions between words, phrases or the whole sentences correspondently. The basics of matching signal is the word level matching signal, including exact word matching and semantic similar matching, while the phrase and sentence level matching signals are obtained by composing lower level ones. This is similar to image recognition, in which raw pixels provide basic units and higher abstract visual patterns can be extracted by layer-by-layer composition [86].

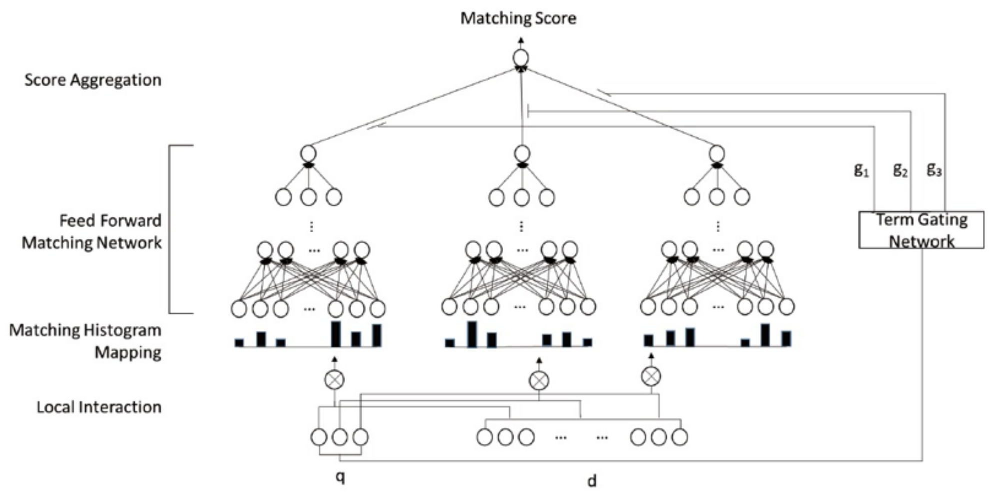

FIGURE 3.15: Architecture for DRMM Model

3.3.4 Deep Relevance Matching Model(DRMM)

DRMM is a state-of-the-art interaction focused deep matching model and has been tested on ad-hoc retrieval task, as depicted in 3.15. The authors [22] argue that the ad-hoc retrieval task which is mainly about relevance matching, is different from most NLP matching tasks which concern semantic matching.

Compared to semantic matching, relevance matching is focused on three factors. Firstly, exact matching of terms between documents and queries is still the most important signals. This is due to the indexing and search paradigm in modern search engines. Secondly, query term importance should be considered. Since queries are mainly keywords based, and the compositional relation among query terms is usually simple, it is important to take into account term discrimination. Lastly, relevance matching could happen in any part of a relevant document, matching whole document or part of document. This is because of the length of document varies and the document may not be topic concentrated.

According to the factors mentioned above, in DRMM model, the variable-length local interaction between query and document terms is first transformed into a fixed-length matching histogram. Based on this fixed-length matching histogram, a feed forward matching network is employed to learn hierarchical matching patterns Finally, generate the overall matching score for each query term.

3.4 Multi-Channel Siamese BLSTM(MC-BLSTM)

In this thesis, we propose a Multi-Channel Siamese BLSTM framework, which could incorporate different types of knowledge or fact for text semantic similarity. In the following of this paragraph, we first formulate the problem of text semantic matching. Then, we present the basic of siamese network, which is the basis of the our proposed model. Lastly, we detail our proposed Multi-Channel Siamese BLSTM framework.

3.4.1 Problem formulation

The setup of text semantic similarity is as follows: we are given two texts: (S_1, S_2), where each sentence can be represented as a word sequence, $(X_1^{(1)}, X_2^{(1)}, \ldots X_n^{(1)})$ and $(X_1^{(2)}, X_2^{(2)}, \ldots X_m^{(2)})$ for S_1 and S_2 respectively. A function F is usually needed to compute the similarity between these two texts: (S_1, S_2), which indicates how much of the two text is similar semantically to each other. More formally, this problem can be described as follows:

$$F(S_1, S_2) \in [0, 1] \tag{3.16}$$

where the function F denotes some sort of semantic similarity. The output is usually in the range of [0, 1], where 1 means S_1 and S_2 are similar and 0 means dissimilar. Under the deep learning paradigm, the sentences (S_1, S_2) are usually mapped to vector representations with a sub network, such that F can be used for comparison.

$$F(\varPhi_1(S_1), \varPhi_2(S_2)) \in [0, 1] \tag{3.17}$$

where $\varPhi$ is a sub network.

3.4.2 Siamese networks

Siamese nearal networks [87] are dual-branch networks with tied weights, in which the two sub-networks are constrained to have identical weight. The general untied version of this model may be more useful for applications with asymmetric domains such as information retrieval [74].

Siamese nearal networks are popular among tasks that involve finding similarity or a relationship between two comparable things. Some examples are paraphrase scoring, where the inputs are two sentences and the output is a score of how similar they are; or signature verification, where figure out whether two signatures are from the same person. Generally, in such tasks, two identical subnetworks are used to process the two inputs, and another module will take their outputs and produce the final output. The picture below is from Bromley et al. (1993) [88]. They proposed a siamese architecture for the signature verification task [3].

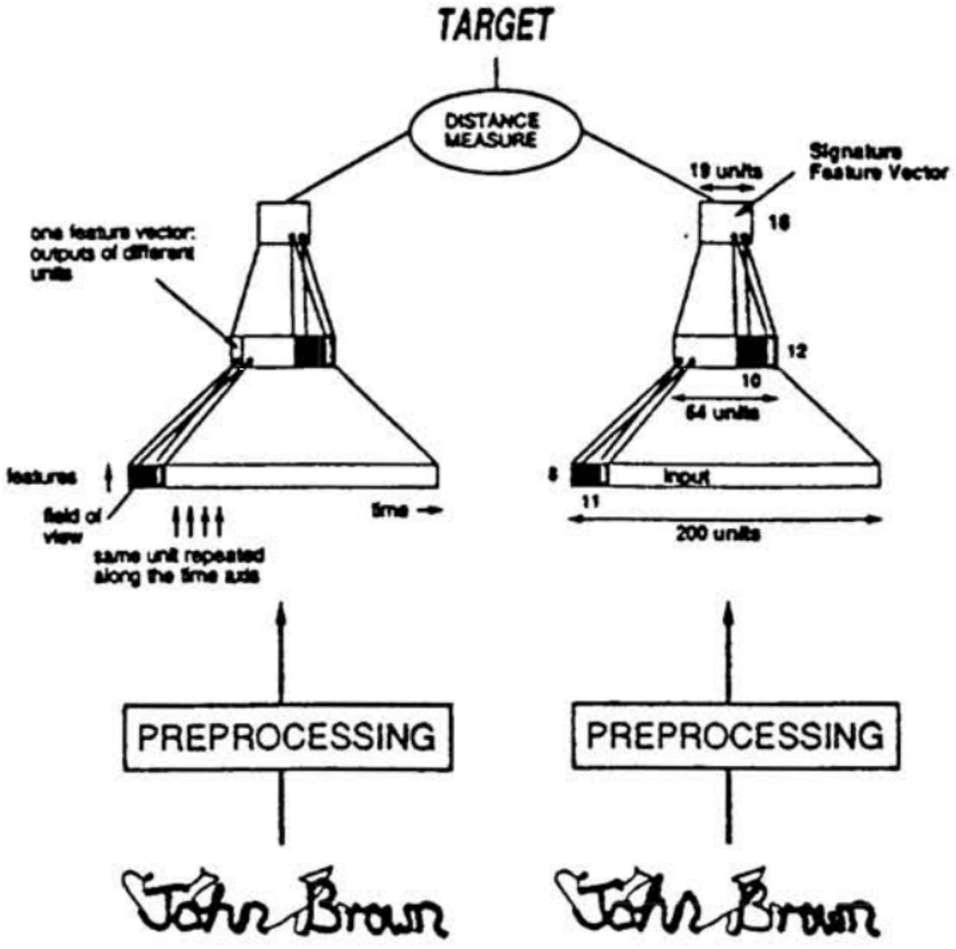

FIGURE 3.16: Architecture of Siamese Network

Siamese architectures are good in these tasks because:

- Sharing weights across subnetworks means fewer parameters to train for, which in turn means less data required and less tendency to overfit.

[3]https://www.quora.com/What-are-Siamese-neural-networks-what-applications-are-they-good-for-and-why

- Each subnetwork essentially produces a representation of its input. ("Signature Feature Vector" in the picture.) If your inputs are of the same kind, like matching two sentences or matching two pictures, it makes sense to use similar model to process similar inputs. This way you have representation vectors with the same semantics, making them easier to compare.

3.4.3 the Proposed Framework

In text semantic similarity, some recent studies have used siamese architectures to score relevance between two texts. Such as in question answering system, one input is a question, the other input is another answer, and the output is how relevant it is between them. Two sentences do not look exactly the same, but if the goal is to extract the similarity or a connection between each other, a siamese architecture can work well with considering the mutual influence of two sentences [89].

In our proposed multi-channel siamese BLSTM model, we use bidirectional LSTM layers to encode the sentence pair, and then fully connected by a dense layer to learn a classifier. As we discussed in previous sections, we argue the only word information for understanding a natural language sentence may be not enough. The understanding may also require external knowledge or facts which should be utilized by computers. We expand the basic siamese framework by adding extra channels to incorporate external knowledge for better understanding. In particular, except words information we also take its Part-Of-Speech tagging as input, $(embedding(X_j^{(i)}) + embedding(POS(X_j^{(i)})))$, which is viewed as multi-channel in this paper. Part-Of-Speech tagging (or POS tagging, for short) is one of the main components of almost any NLP analysis. The task of POS-tagging simply implies labeling words with their appropriate Part-Of-Speech (Noun, Verb, Adjective, Adverb, Pronoun, . . .). We believe this will benefit our task. In our proposed framework, other kind of knowledge could also be easily incorporated. Our model is shown in figure 3.17.

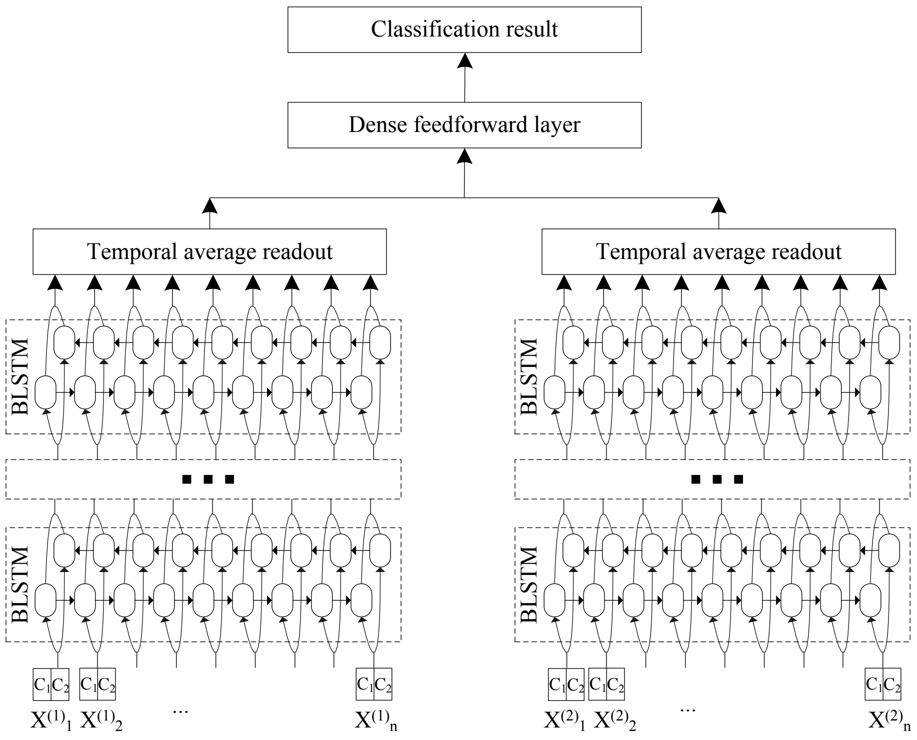

FIGURE 3.17: Architecture of Multi-Channel Siamese BLSTM

The difference for Multi-Channel model is that the input is word representation concatenating its POS representation as input, while bidirectional model runs an extra reversed LSTM model and sum the results for next layer. We use a binary cross-entropy cost function for this classification task. The benefit of using cross-entropy is that, unlike quadratic cost function, it avoids the problem of learning slowing down. The training objective is to minimize the following cost function.

L_2 regularization is applied to prevent overfitting problem. Adam optimizer [90] is used. After learning, we use validation set to choose the best model for testing. The related formulas are as follows:

$$C = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)] \tag{3.18}$$

$$a = \sigma(z) \tag{3.19}$$

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{3.20}$$

$$z = \sum_{j} \omega_j x_j + b \tag{3.21}$$

Chapter 4

Experimental Settings

4.1 Dataset

4.1.1 Overview

Quora is a platform for sharing knowledge about anything. People ask and discuss questions, others may provide unique insights and quality answers, which enable people to learn from each other. However, as there are more than 100 million people visit every month, it is not surprising that many people repeatedly ask semantically similar questions. This may cause troubles for both seekers and writers that it may take more time to find the best answer and increase workload to answer questions.

In order to make it easier to find high quality answers and improve experience for Quora writers, seekers and readers, in 2017, Quora held a competition to identify duplicate questions, which required to classify whether question pairs are duplicates or not by applying advanced techniques.

4.1.2 Data description

The dataset is released and preprocessed by Quora [91]. There are three important features about this dataset. The original sampling method returned an imbalanced dataset with more true examples of duplicate pairs than non-duplicates. Thus negative examples are supplemented from a source of “related questions”, in order to pertain to similar topics but truly not semantically equivalent.

Due to the combination of sampling procedures and some sanitization measures that have been applied to the final dataset (e.g., removal of questions with extremely long question details), the distribution of questions in the dataset should not be taken to be representative of the distribution of questions asked on Quora.

In summary, the ground-truth labels are not guaranteed to be perfect and they may contain some noise.

4.1.3 Statistics of dataset

The final dataset consists of 404,351 lines of potential question duplicate pairs. A sample from the dataset is shown in Appendix E. Each line contains IDs for each question in the pair, the full text for each question, and a binary value that indicates whether the line truly contains a duplicate pair.

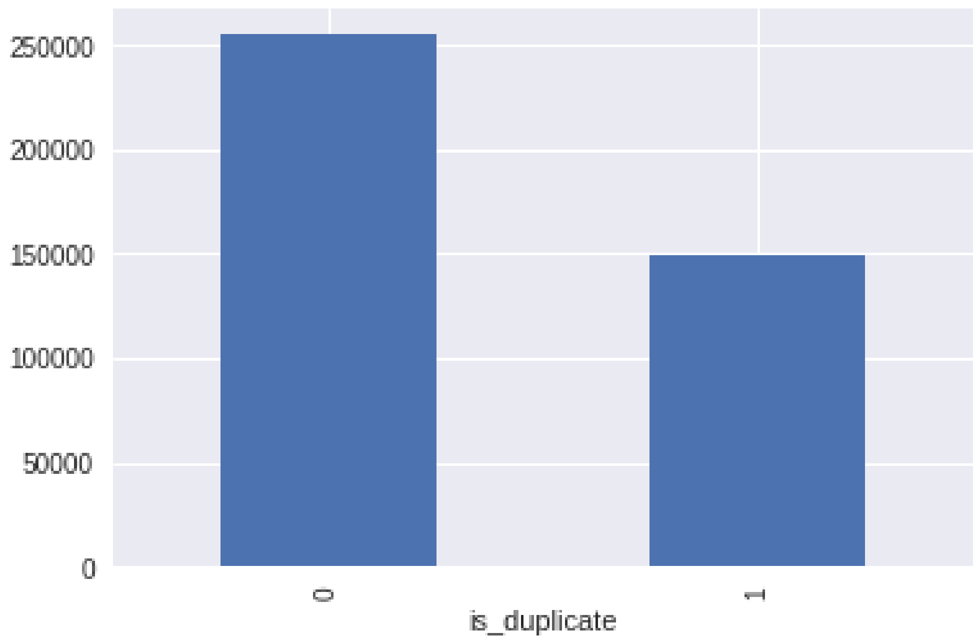

Figure 4.1: Ratios for Classes

There are 255045 negative (non-duplicate) and 149306 positive (duplicate) pairs. The ratio between negative and positive class is about 1.7:1. This indicates that the dataset is slightly imbalanced as shown in figure 4.1. Since the class balance can influence some classifiers, this fact should be considered when training machin learning models.

The length of questions also varies greatly. The shortest question is 1 character long (which is useless for the task) and the longest question is 1169 characters (which is a long, complicated question). The pairs which are shorter than 10 characters are removed. Thus the average length is 59 and std is 32.

We divide the total dataset into a training set D of 323,481 pairs and a test set T of 40,435 pairs and a validation set V of 40,435 pairs.

4.2 Experimental Environment

All the deep learning models are trained on a server computer with NVIDIA GeForce GTX 1080 Ti GPU with 8 GPU RAM. For the development of deep learning models, keras is used that a high-level neural networks API, written in Python. Keras [1] can run on top of TensorFlow, CNTK, or Theano seamlessly on both CPU and GPU.

In our experiments, we use the backend of TensorFlow, which is an open source software library for numerical computation using data flow graphs. TensorFlow [2] was originally developed by researchers and engineers working on the Google Brain team within Google's Machine Intelligence Research organization for the purposes of conducting machine learning and deep neural networks research. The system is general enough to be applicable in a wide variety of other domains, as well.

4.3 Data Preprocessing

Converting Questions into Vectors

We use the GloVe word embedding to convert each question into a semantic vector then we stack a siamese network to detect if the pair is duplicate. In particular, we embed words into a vector space with 300 dimensions in general. These vectors capture semantics and even analogies between different words.

[1]https://github.com/keras-team/keras
[2]https://github.com/tensorflow/tensorflow

For this particular problem, we train the GloVe model by using a tool of Gensim [3]. The implement can be found in Appendix A.

4.4 Model Training and Evaluation

4.4.1 Dataset Split

In machine learning, data used to build the final model usually come from multiple datasets. In particular, three data sets are commonly used in different stages of the creation of the model. The split is illustrated in Figure 4.2.

- Training Dataset: the sample of data used to fit the model
- Validation Dataset: the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as a skill on the validation dataset is incorporated into the model configuration.
- Test Dataset: the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

In order to train and evaluate our models, we split the data set into 3 parts, where 80% of the dataset are used for training, 10% are used for validation and the rest 10% for test. For the parameters of the models, we do optimization based on standard grid search.

[3] https://radimrehurek.com/gensim/

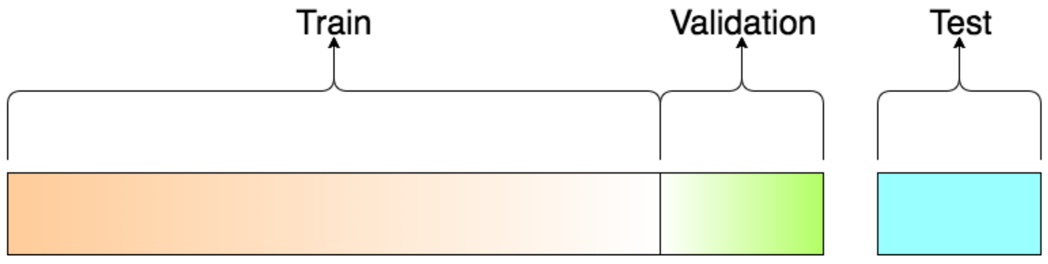

FIGURE 4.2: A Visualization of Dataset Split

4.4.2 Backpropagation: Adam

The backpropagation algorithm is usually used for training a deep neural network. It was originally introduced in the 1970s, but its importance was not fully appreciated until 1986 a famous paper publish by David Rumelhart, etc. [92]. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize the weights in neural networks.

The overall goal of the algorithm is to attempt to find weights such that, for every input vector in the training set, the neural network yields an output vector closely matching the prescribed target vector. The procedure can be described as follows:

- Initialize the weights and set the learning rate and the stopping criteria.
- Randomly choose an input and the corresponding target.
- Compute the input to each layer and the output of the final layer.
- Compute the sensitivity components.
- Compute the gradient components and update the weights.
- Check against the stopping criteria. Exit and return the weights or loop back to step 2.

In all our experiments, the Adam algorithm is used. Adaptive Moment Estimation (Adam) [90] is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. In this method, in addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum:

$$m_t = \beta_1 m_{t_1} + (1 - \beta_1) g_t ls \tag{4.1}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.2}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and when the decay rates are small (i.e. β_1 and β_2 are close to 1).

4.4.3 Model Training

We implement the following learning procedure. Firstly, we randomly split the Quora dataset into 3 parts, 80% for training, 10% for validation and 10% for test. We ensure that each of part of the datasets contains an equal proportion of the labels from the original dataset. For a question pair (S_1, S_2, y)in the training set, each word $X_j^{(i)}$is mapped to a vector space by using 300d reference GloVe vectors[63] as $embedding(X_j^{(i)})$.

Secondly, we use a binary cross-entropy cost function for this classification task. The benefit of using cross-entropy is that, unlike quadratic cost function, it avoids the problem of learning slowing down. L_2 regularization is applied to prevent over-fitting problem. For the training of the models, the Adam optimizer is used to choose the best model for testing. The learning rate is set to 0.001, and the batch size is set to 64.

For POS-tagger, there are some tools available in NLTK for building use-specified POS-tagger, however, because this is not the focus of this thesis, we use the default NLTK tagger [4] in our experiments.

In addition, after deleting the stop words and infrequent words, the vocabulary size of the dataset is 90908. The max length of the input sentences is set to 20. Longer sentences will be truncated to 20 while short ones will be padded to the same length.

4.4.4 Evaluation Metrics: Accuracy

In our experiments, accuracy is used for evaluation. Specifically, accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions made. Figure 4.3 illustrate the computation of accuracy [5].

In the numerator, are our correct predictions (True Positives and True Negatives)(Marked as red in the figure 4.3) and in the denominator, are the kind of all

[4]http://www.nltk.org/book/ch05.html

[5]https://medium.com/greyatom/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b

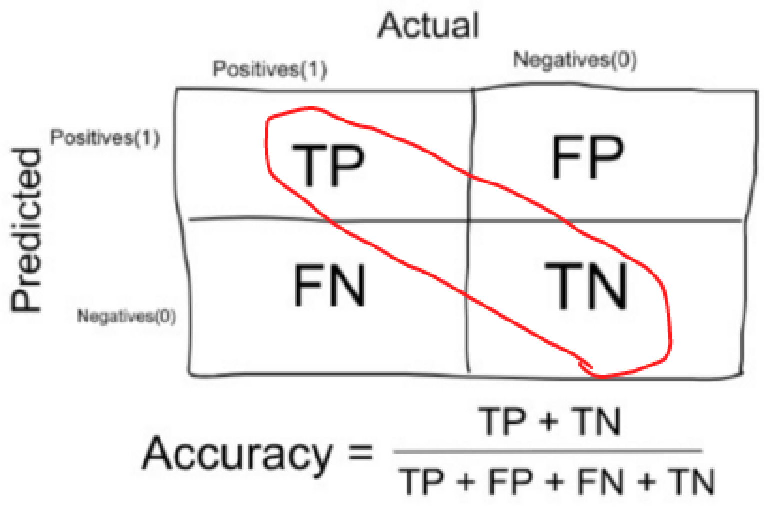

FIGURE 4.3: Computation of Accuracy

predictions made by the algorithm(Right as well as Wrong ones). In particular, a true positive is an outcome where the model correctly predicts the positive class. Similarly, a true negative is an outcome where the model correctly predicts the negative class. A false positive is an outcome where the model incorrectly predicts the positive class. And a false negative is an outcome where the model incorrectly predicts the negative class.

Accuracy is a good measure when the target variable classes in the data are nearly balanced. Accuracy should never be used as a measure when the target variable classes in the data are a majority of one class.

Chapter 5

Experimental Results and Discussion

5.1 Baselines

In our experiments, we compare our proposed model with some basic LSTM modes (including bidirectional LSTM), as well as some recently proposed deep neural network based models. The models are described as follows:

- **LSTM**: Basic siamese network with LSTM as described in Section 3.2.3 to compute the sentence representation, which is the base of our proposed model.
- **BLSTM**: Bidirectional LSTM, which is similar with the above LSTM except that two directions of LSTM described in Section 3.2.4 are used.

- **ARC-I**: ARC-I [35] is a general representation-focused deep matching model that has been tested on a set of NLP tasks including sentence completion, response matching, and paraphrase identification.

- **ARC-II**: ARC-II [35] was proposed by the authors of the model ARC-I, but focuses on learning hierarchical matching patterns from local interactions using a CNN.

- **MatchPyramid**: MatchPyramid [77] is another state-of-the-art deep matching model and has been tested on two NLP tasks including paraphrase identification and paper citation matching.

- **DRMM**: Deep relevance matching model (DRMM) is specially proposed for ad-hoc retrieval. This model employs a joint deep architecture at the query term level for relevance matching. By using matching histogram mapping, a feed forward matching network, and a term gating network, it can effectively deals with special relevance matching factors. We adapt this model for our sentence similarity classification.

- **MC-BLSTM**: MC-BLSTM denotes our proposed Multi-Channel Siamese BLSTM model.

5.2 Performance and Analysis

This section presents the performance of different baselines over the quora dataset, the results are shown in Table 5.1. We compare our proposed model with both

TABLE 5.1: Comparison of Performance for Each Model

Model	Train Accuracy	Validation Accuracy	Test Accuracy
LSTM*	0.9015	0.7866	0.7754
BLSTM	0.9218	0.7906	0.7912 (2.0%)
DRMM	0.8806	0.6402	0.6278 (-19.0%)
ARCI	0.9218	0.7906	0.7616 (-1.78.0%)
ARCII	0.9116	0.7806	0.7400 (-4.44%)
MatchPyramid	0.9188	0.7896	0.7937 (2.36%)
MC-BLSTM	0.9298	0.8126	0.8120 (4.70%)

basic LSTM and the bidirectional siamese LSTM model under optimized settings. We also present some of the state-of-the-art deep neural network based models for reference.

As we can see, Table 5.1 demonstrates that the DRMM model performs the worst among all models. This is maybe because that DRMM model was specifically proposed for the adhoc information retrieval problem. It is good at capturing the relevance between a query and a document instead of the semantic similarity between two text, which demonstrates the significant differences between semantic matching and relevance matching.

The ARC-I model, although trained on the corresponding corpus, was outperformed by basic LSTM and BLSTM models. A possible reason is that ARC-I concatenates the two text representation for computing the matching score, which may be less effective than the cosine function in the siamese framework. ARC-II performed even worst than ARC-I. These models were originally evaluated on other tasks like paraphrase other than the duplicated question detection on the quora dataset.

The MatchPyramid model provides 2.36% better than the basic LSTM, which is probably due to the indirect local interactions (i.e., local interaction is based on the weighted sum of query and document term vectors rather than cosine similarity or dot product).

Lastly, we have shown that our proposed approach is able to significantly outperform other strong algorithms, including the state-of-the-art MatchPyramid model, which verifies the effectiveness of our proposed multi-channel framework. Indeed, additional knowledge of words are useful to model the semantic meaning of pairs of text, which can be modeled properly under our model. This also demonstrates the potential of our model in capturing the short text semantic similarity.

5.3 Effect of Term Embeddings

Since we leverage a priori learned term embeddings in our model, we further study the effect of embedding dimensionality on the performance. Here we report the performance results on the quora dataset using term embeddings trained by the GloVe model with 50, 100, and 300 dimensions, respectively. As shown in Table 5.2, the performance increases with the increase of dimensionality. Term embeddings of different dimensionality provide different granularity of semantic similarity; they may also require different amounts of training data. With lower dimensionality, the similarity between term embeddings might be coarse and hurt the text semantic matching performance. However, with larger dimensionality, one may need more data to train reliable term embeddings. Our results suggest

Table 5.2: Effect of Embedding Dimensionality

Model	Embedding	Test Accuracy
MC-BLSTM	GloVe-50d	0.7410
MC-BLSTM	GloVe-100d	0.8080
MC-BLSTM	GloVe-300d	0.8120

that 300 dimensions is sufficient for learning term embeddings effective for text semantic similarity.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, We study the problem of text semantic similarity in natural language processing.

- Firstly, we briefly introduce the problem of text semantic similarity and its application, and review the related literature including traditional methods used to solve this problem as well as the recent research of deep learning methods that have achieved state-of-the-art results in NLP area.
- Secondly, we introduce some basic concepts of deep learning and compare the recent state-of-the-art deep models for text semantic similarity in details. Based on the strength and the drawbacks of these deep models, we propose a Multi-Channel Siamese BLSTM model that is verified on duplicate question pair detection dataset.

- In particular, we have proposed a novel, effective and practical approach to the text semantic similarity problem in natural language processing.

Through extensive experiments on the Quora test collection, our proposed approach has shown remarkable and significant improvement over strong algorithms, including the basic siamese bidirectional LSTM model, which verifies the effectiveness of our multi-channel framework.

6.2 Future Work

In the future, we plan to investigate further applications of our proposed approach.

- For example, firstly we plan to extend the knowledge sources (vertical knowledge graph database) of the words to further improve the performance.
- Secondly, we may also include phrase embeddings so that phrases can be treated as a whole rather than separate terms. In this way, we expect the local interactions can better reflect the meaning by using the proper semantic units in language, leading to better matching performance.
- Thirdly, our proposed framework is generic and can be applied to many different task. So we also plan to investigate its performance in different tasks like information retrieval, paraphrase, etc.
- Lastly, we in this thesis only use POS information to verify our framework on a single dataset. We will explore more types of information like syntactic

parsers and knowledge graph to further improve the performance of text semantic similarity comparison.

Appendix A

Training of the GloVe Word Embedding

A.1 Instructions

Let's load the training data first.

```

import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
df =
    pd.read_csv("./Quora/DuplicateQuestion/quora_duplicate_questions.tsv",delim

# convert questions into unicode.
```

```
df['question1'] = df['question1'].apply(lambda x:
    unicode(str(x),"utf-8"))
df['question2'] = df['question2'].apply(lambda x:
    unicode(str(x),"utf-8"))
```

A.2 GloVe Training with Gensim

```

import gensim

questions = list(df['question1']) + list(df['question2'])

# tokenize the questions
c = 0
for question in tqdm(questions):
    questions[c] = list(gensim.utils.tokenize(question,
    deacc=True, lower=True))
    c += 1

# train a model
model = gensim.models.Word2Vec(questions, size=300, workers=8,
    iter=15, negative=25)

# trim memory
model.init_sims(replace=True)

# creta a dict
```

```
w2v = dict(zip(model.index2word, model.syn0))
print "Number of tokens in Word2Vec:", len(w2v.keys())

# save the model
model.save('data/word2vec.mdl')
model.save_word2vec_format('data/word2vec.bin', binary=True)
```

The above code trains a GLOVE model and saves it. It generates 300 dimensional vectors for words. Hyper parameters would be chosen better but it is just a baseline to see a initial performance. However, as I'll show this model gives performance below than my expectation. I believe, this is because our questions are short and does not induce a semantic structure that GLOVE is able to learn a salient model.

A.3 Data Preprocessing

```

from __future__ import print_function
import numpy as np
import csv, datetime, time, json
from zipfile import ZipFile
from os.path import expanduser, exists
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Dense, Dropout, Reshape,
    Merge, BatchNormalization, TimeDistributed, Lambda
from keras.regularizers import l2
from keras.callbacks import Callback, ModelCheckpoint
from keras.utils.data_utils import get_file
from keras.engine import Input, Merge, Model
from keras.layers import Embedding, Bidirectional, LSTM, Dense,
    BatchNormalization, Activation, Lambda
from keras.optimizers import Adam, Adadelta, Adagrad, RMSprop, SGD
from keras import backend as K
from sklearn.model_selection import train_test_split
import nltk


# KERAS_DATASETS_DIR = expanduser('~/.keras/datasets/')
KERAS_DATASETS_DIR = expanduser('./')
QUESTION_PAIRS_FILE_URL =
    'http://qim.ec.quoracdn.net/quora_duplicate_questions.tsv'
QUESTION_PAIRS_FILE = 'quora_duplicate_questions.tsv'
GLOVE_ZIP_FILE_URL =
    'http://nlp.stanford.edu/data/glove.840B.300d.zip'
GLOVE_ZIP_FILE = 'glove.840B.300d.zip'
GLOVE_FILE = 'glove.840B.300d.txt'
Q1_TRAINING_DATA_FILE = 'q1_train.npy'
Q2_TRAINING_DATA_FILE = 'q2_train.npy'
Q1_TAG_TRAINING_DATA_FILE = 'q1_train_tag.npy'
Q2_TAG_TRAINING_DATA_FILE = 'q2_train_tag.npy'
LABEL_TRAINING_DATA_FILE = 'label_train.npy'
WORD_EMBEDDING_MATRIX_FILE = 'word_embedding_matrix.npy'
NB_WORDS_DATA_FILE = 'nb_words.json'
MAX_NB_WORDS = 200000
```

```
MAX_NB_WORDS_TAGS = 40
MAX_SEQUENCE_LENGTH = 20
EMBEDDING_DIM = 300
TAG_EMBEDDING_DIM = 30
MODEL_WEIGHTS_FILE = 'question_pairs_weights.h5'
VALIDATION_SPLIT = 0.1
TEST_SPLIT = 0.1
RNG_SEED = 13371447
NB_EPOCHS = 10

if exists(Q1_TRAINING_DATA_FILE) and
   exists(Q2_TRAINING_DATA_FILE) and
   exists(LABEL_TRAINING_DATA_FILE) and exists(
        NB_WORDS_DATA_FILE) and
   exists(WORD_EMBEDDING_MATRIX_FILE) \
        and exists(Q1_TAG_TRAINING_DATA_FILE) and
   exists(Q2_TAG_TRAINING_DATA_FILE):
    q1_data = np.load(open(Q1_TRAINING_DATA_FILE, 'rb'))
    q2_data = np.load(open(Q2_TRAINING_DATA_FILE, 'rb'))
    q1_data_tag = np.load(open(Q1_TAG_TRAINING_DATA_FILE, 'rb'))
    q2_data_tag = np.load(open(Q2_TAG_TRAINING_DATA_FILE, 'rb'))
    labels = np.load(open(LABEL_TRAINING_DATA_FILE, 'rb'))
    word_embedding_matrix =
   np.load(open(WORD_EMBEDDING_MATRIX_FILE, 'rb'))
    with open(NB_WORDS_DATA_FILE, 'r') as f:
        d = json.load(f)
        nb_words = d['nb_words']
        nb_words_tags = d['nb_words_tags']
```

```
else:
    if not exists(KERAS_DATASETS_DIR + QUESTION_PAIRS_FILE):
        get_file(QUESTION_PAIRS_FILE, QUESTION_PAIRS_FILE_URL)

    print("Processing", QUESTION_PAIRS_FILE)

    question1 = []
    question2 = []
    is_duplicate = []
    with open(KERAS_DATASETS_DIR + QUESTION_PAIRS_FILE) as
   csvfile:
    # with open(KERAS_DATASETS_DIR + QUESTION_PAIRS_FILE,
   encoding='utf-8') as csvfile:
        reader = csv.DictReader(csvfile, delimiter='\t')
        for row in reader:
            # question1.append(row['text1'])
            # question2.append(row['text2'])
            # is_duplicate.append(row['duplicate'])
            question1.append(row['question1'])
            question2.append(row['question2'])
            is_duplicate.append(row['is_duplicate'])

    print('Question pairs: %d' % len(question1))

    questions = question1 + question2
    tokenizer = Tokenizer(nb_words=MAX_NB_WORDS)
    tokenizer.fit_on_texts(questions)
```

```
question1_word_sequences =
tokenizer.texts_to_sequences(question1)
question2_word_sequences =
tokenizer.texts_to_sequences(question2)
word_index = tokenizer.word_index
print('word_index size: ', len(word_index),
max(word_index.values()))
print(question1[0:2])
index_word = [0] * (len(word_index) + 1)
for word, i in word_index.items():
    index_word[i] = word

question1_word_sequences_tags = []
question2_word_sequences_tags = []
for sentence in question1_word_sequences:
    sentence = [index_word[i] for i in sentence]
    tags = ' '.join([x[1] for x in nltk.pos_tag(sentence)])
    question1_word_sequences_tags.append(tags)
for sentence in question2_word_sequences:
    sentence = [index_word[i] for i in sentence]
    tags = ' '.join([x[1] for x in nltk.pos_tag(sentence)])
    question2_word_sequences_tags.append(tags)
tokenizer_tag = Tokenizer(nb_words=MAX_NB_WORDS_TAGS)
tokenizer_tag.fit_on_texts(question1_word_sequences_tags +
question2_word_sequences_tags)
question1_word_sequences_tags =
tokenizer_tag.texts_to_sequences(question1_word_sequences_tags)
```

```
question2_word_sequences_tags = tokenizer_tag.texts_to_sequences(question2_word_sequences_tags)

print("Words in index: %d" % len(word_index))

if not exists(KERAS_DATASETS_DIR + GLOVE_ZIP_FILE):
    zipfile = ZipFile(get_file(GLOVE_ZIP_FILE, GLOVE_ZIP_FILE_URL))
    zipfile.extract(GLOVE_FILE, path=KERAS_DATASETS_DIR)

print("Processing", GLOVE_FILE)

embeddings_index = {}
with open(KERAS_DATASETS_DIR + GLOVE_FILE) as f:
# with open(KERAS_DATASETS_DIR + GLOVE_FILE, encoding='utf-8') as f:
    for line in f:
        values = line.split(' ')
        word = values[0]
        embedding = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = embedding

print('Word embeddings: %d' % len(embeddings_index))

nb_words = min(MAX_NB_WORDS, len(word_index))
nb_words_tags = min(MAX_NB_WORDS_TAGS, len(tokenizer_tag.word_index))
```

```
    word_embedding_matrix = np.zeros((nb_words + 1,
EMBEDDING_DIM))
    for word, i in word_index.items():
        if i > MAX_NB_WORDS:
            continue
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            word_embedding_matrix[i] = embedding_vector

    print('Null word embeddings: %d' %
np.sum(np.sum(word_embedding_matrix, axis=1) == 0))

    q1_data = pad_sequences(question1_word_sequences,
maxlen=MAX_SEQUENCE_LENGTH)
    q2_data = pad_sequences(question2_word_sequences,
maxlen=MAX_SEQUENCE_LENGTH)
    q1_data_tag = pad_sequences(question1_word_sequences_tags,
maxlen=MAX_SEQUENCE_LENGTH)
    q2_data_tag = pad_sequences(question2_word_sequences_tags,
maxlen=MAX_SEQUENCE_LENGTH)

    labels = np.array(is_duplicate, dtype=int)
    print('Shape of question1 data tensor:', q1_data.shape)
    print('Shape of question2 data tensor:', q2_data.shape)
    print('Shape of label tensor:', labels.shape)

    np.save(open(Q1_TRAINING_DATA_FILE, 'wb'), q1_data)
    np.save(open(Q2_TRAINING_DATA_FILE, 'wb'), q2_data)
    np.save(open(Q1_TAG_TRAINING_DATA_FILE, 'wb'), q1_data_tag)
    np.save(open(Q2_TAG_TRAINING_DATA_FILE, 'wb'), q2_data_tag)

    np.save(open(LABEL_TRAINING_DATA_FILE, 'wb'), labels)
    np.save(open(WORD_EMBEDDING_MATRIX_FILE, 'wb'),
word_embedding_matrix)
    with open(NB_WORDS_DATA_FILE, 'w') as f:
        json.dump({'nb_words': nb_words, 'nb_words_tags':
nb_words_tags}, f)
```

Appendix B

Implementation of Basic Siamese LSTM

```

from __future__ import print_function
import numpy as np
import csv, datetime, time, json
from zipfile import ZipFile
from os.path import expanduser, exists
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Dense, Dropout, Reshape,
    Merge, BatchNormalization, TimeDistributed, Lambda
from keras.regularizers import l2
from keras.callbacks import Callback, ModelCheckpoint
from keras.utils.data_utils import get_file
from keras.engine import Input, Merge, Model
from keras.layers import Embedding, Bidirectional, LSTM, Dense,
    BatchNormalization, Activation, Lambda
from keras.optimizers import Adam, Adadelta, Adagrad, RMSprop, SGD
from keras import backend as K
from sklearn.model_selection import train_test_split
import nltk

embedding = Embedding(nb_words + 1, EMBEDDING_DIM,
    weights=[word_embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
                        trainable=False)
tag_embedding = Embedding(nb_words_tags + 1, TAG_EMBEDDING_DIM,
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=True)


def build_rnn():
    input_query_a = Input(shape=(None,), name='input_query_a',
    dtype='int32')
    input_query_b = Input(shape=(None,), name='input_query_b',
    dtype='int32')

    embedding_query_a = embedding(input_query_a)  # output
    dimension is 100
    embedding_query_b = embedding(input_query_b)  # output
    dimension is 100

    lstm_query_a = LSTM(output_dim=200, init='glorot_uniform',
    inner_init='orthogonal')(
        embedding_query_a)
    lstm_query_b = LSTM(output_dim=200, init='glorot_uniform',
    inner_init='orthogonal')(
        embedding_query_b)
    merged = Merge(mode='concat')([lstm_query_a, lstm_query_b])
    output = Dense(output_dim=1, activation='sigmoid',
    W_regularizer='l2')(merged)

    model = Model(input=[input_query_a, input_query_b],
    output=output)
    return model


model = build_rnn()

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy', 'precision', 'recall',
    'fbeta_score'])

callbacks = [ModelCheckpoint(MODEL_WEIGHTS_FILE,
    monitor='val_acc', save_best_only=True)]

print("Starting training at", datetime.datetime.now())

t0 = time.time()
```

```
history = model.fit([Q1_train, Q2_train, Q1_TAG_train,
    Q2_TAG_train],
                    y_train,
                    nb_epoch=NB_EPOCHS,
                    validation_split=VALIDATION_SPLIT,
                    verbose=1,
                    callbacks=callbacks)
t1 = time.time()

print("Training ended at", datetime.datetime.now())

print("Minutes elapsed: %f" % ((t1 - t0) / 60.))

model.load_weights(MODEL_WEIGHTS_FILE)

loss, accuracy, precision, recall, fbeta_score =
    model.evaluate([Q1_test, Q2_test, Q1_tag_test, Q2_tag_test],
    y_test)
print('')
print('loss      = {0:.4f}'.format(loss))
print('accuracy  = {0:.4f}'.format(accuracy))
print('precision = {0:.4f}'.format(precision))
print('recall    = {0:.4f}'.format(recall))
print('F         = {0:.4f}'.format(fbeta_score))
```

Appendix C

Implementation of Siamese Bidirectionarl LSTM

```

from __future__ import print_function
import numpy as np
import csv, datetime, time, json
from zipfile import ZipFile
from os.path import expanduser, exists
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Dense, Dropout, Reshape,
    Merge, BatchNormalization, TimeDistributed, Lambda
from keras.regularizers import l2
from keras.callbacks import Callback, ModelCheckpoint
from keras.utils.data_utils import get_file
from keras.engine import Input, Merge, Model
from keras.layers import Embedding, Bidirectional, LSTM, Dense, BatchNormalization, Activation, Lambda
from keras.optimizers import Adam, Adadelta, Adagrad, RMSprop, SGD
from keras import backend as K
from sklearn.model_selection import train_test_split
import nltk

embedding = Embedding(nb_words + 1, EMBEDDING_DIM, weights=[word_embedding_matrix], input_length=MAX_SEQUENCE_LENGTH,
                      trainable=False)
tag_embedding = Embedding(nb_words_tags + 1, TAG_EMBEDDING_DIM,
                          input_length=MAX_SEQUENCE_LENGTH,
                          trainable=True)


def build_bi_rnn():
    input_query_a = Input(shape=(None,), name='input_query_a', dtype='int32')
    input_query_b = Input(shape=(None,), name='input_query_b', dtype='int32')

    embedding_query_a = embedding(input_query_a)  # output dimension is 100
    embedding_query_b = embedding(input_query_b)  # output dimension is 100

    lstm_query_a = Bidirectional(LSTM(output_dim=50,
    init='glorot_uniform', inner_init='orthogonal'),
    merge_mode='sum')(
        embedding_query_a)
    lstm_query_b = Bidirectional(LSTM(output_dim=50,
    init='glorot_uniform', inner_init='orthogonal'),
    merge_mode='sum')(
        embedding_query_b)
    merged = Merge(mode='concat')([lstm_query_a, lstm_query_b])
    output = Dense(output_dim=1, activation='sigmoid',
    W_regularizer='l2')(merged)

    model = Model(input=[input_query_a, input_query_b],
    output=output)
    return model



model = build_bi_rnn()

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy', 'precision', 'recall',
    'fbeta_score'])

callbacks = [ModelCheckpoint(MODEL_WEIGHTS_FILE,
    monitor='val_acc', save_best_only=True)]

```

```
print("Starting training at", datetime.datetime.now())

t0 = time.time()
history = model.fit([Q1_train, Q2_train, Q1_TAG_train,
    Q2_TAG_train],
                    y_train,
                    nb_epoch=NB_EPOCHS,
                    validation_split=VALIDATION_SPLIT,
                    verbose=1,
                    callbacks=callbacks)
t1 = time.time()

print("Training ended at", datetime.datetime.now())

print("Minutes elapsed: %f" % ((t1 - t0) / 60.))

model.load_weights(MODEL_WEIGHTS_FILE)

loss, accuracy, precision, recall, fbeta_score =
    model.evaluate([Q1_test, Q2_test, Q1_tag_test, Q2_tag_test],
    y_test)
print('')
print('loss      = {0:.4f}'.format(loss))
print('accuracy  = {0:.4f}'.format(accuracy))
print('precision = {0:.4f}'.format(precision))
print('recall    = {0:.4f}'.format(recall))
print('F         = {0:.4f}'.format(fbeta_score))
```

Appendix D

Implementation of Multi-Channel Siamese BLSTM(MC-BLSTM)

```

from __future__ import print_function
import numpy as np
import csv, datetime, time, json
from zipfile import ZipFile
from os.path import expanduser, exists
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Dense, Dropout, Reshape,
    Merge, BatchNormalization, TimeDistributed, Lambda
from keras.regularizers import l2
from keras.callbacks import Callback, ModelCheckpoint
from keras.utils.data_utils import get_file
from keras.engine import Input, Merge, Model
from keras.layers import Embedding, Bidirectional, LSTM, Dense,
    BatchNormalization, Activation, Lambda
from keras.optimizers import Adam, Adadelta, Adagrad, RMSprop, SGD
from keras import backend as K
from sklearn.model_selection import train_test_split
import nltk

embedding = Embedding(nb_words + 1, EMBEDDING_DIM,
    weights=[word_embedding_matrix],
    input_length=MAX_SEQUENCE_LENGTH,
                        trainable=False)
tag_embedding = Embedding(nb_words_tags + 1, TAG_EMBEDDING_DIM,
                             input_length=MAX_SEQUENCE_LENGTH,
                             trainable=True)


def build_bi_rnn():
     input_query_a = Input(shape=(None,), name='input_query_a',
    dtype='int32')
     input_query_b = Input(shape=(None,), name='input_query_b',
    dtype='int32')

     input_query_tag_a = Input(shape=(None,),
    name='input_query_tag_a', dtype='int32')
     input_query_tag_b = Input(shape=(None,),
    name='input_query_tag_b', dtype='int32')

     embedding_query_a = embedding(input_query_a)  # output dimension is 100
     embedding_query_b = embedding(input_query_b)  # output dimension is 100

     embedding_query_tag_a = tag_embedding(input_query_tag_a)
     embedding_query_tag_b = tag_embedding(input_query_tag_b)
     merged_a = Merge(mode='concat')([embedding_query_a, embedding_query_tag_a])
     merged_b = Merge(mode='concat')([embedding_query_b, embedding_query_tag_b])

     lstm_query_a = Bidirectional(LSTM(output_dim=100, init='glorot_uniform', inner_init='orthogonal'), merge_mode='sum')(
         merged_a)
     lstm_query_b = Bidirectional(LSTM(output_dim=100, init='glorot_uniform', inner_init='orthogonal'), merge_mode='sum')(
         merged_b)
     merged = Merge(mode='concat')([lstm_query_a, lstm_query_b])
     output = Dense(output_dim=1, activation='sigmoid', W_regularizer='l2')(merged)

     model = Model(input=[input_query_a, input_query_b, input_query_tag_a, input_query_tag_b], output=output)
     return model

```

```

model = build_bi_rnn()

model.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy', 'precision', 'recall',
    'fbeta_score'])

callbacks = [ModelCheckpoint(MODEL_WEIGHTS_FILE,
    monitor='val_acc', save_best_only=True)]

print("Starting training at", datetime.datetime.now())

t0 = time.time()
history = model.fit([Q1_train, Q2_train, Q1_TAG_train,
    Q2_TAG_train],
                        y_train,
                        nb_epoch=NB_EPOCHS,
                        validation_split=VALIDATION_SPLIT,
                        verbose=1,
                        callbacks=callbacks)
t1 = time.time()

print("Training ended at", datetime.datetime.now())

print("Minutes elapsed: %f" % ((t1 - t0) / 60.))

model.load_weights(MODEL_WEIGHTS_FILE)
```

```

loss, accuracy, precision, recall, fbeta_score =
    model.evaluate([Q1_test, Q2_test, Q1_tag_test, Q2_tag_test],
    y_test)
print('')
print('loss       = {0:.4f}'.format(loss))
print('accuracy   = {0:.4f}'.format(accuracy))
print('precision = {0:.4f}'.format(precision))
print('recall     = {0:.4f}'.format(recall))
print('F          = {0:.4f}'.format(fbeta_score))
```

Appendix E

Sample data in Quora

id	qid1	qid2	question1	question2	is_duplicate
0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0
1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0
2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0
3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when math 23 24 math is divided by 24,23?	0

4	9	10	Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?	Which fish would survive in salt water?	0
5	11	12	Astrology: I am a Capricorn Sun Cap moon and cap rising...what does that say about me?	I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What does this say about me?	1
6	13	14	Should I buy tiago?	What keeps childern active and far from phone and video games?	0
7	15	16	How can I be a good geologist?	What should I do to be a great geologist?	1
9	19	20	Motorola (company): Can I hack my Charter Motorolla DCX3400?	How do I hack Motorola DCX3400 for free internet?	0
10	21	22	Method to find separation of slits using fresnel biprism?	What are some of the things technicians can tell about the durability and reliability of Laptops and its components?	0
11	23	24	How do I read and find my YouTube comments?	How can I see all my Youtube comments?	1

447	895	896	What are natural numbers?	What is a least natural number?	0
1518	3037	3038	Which pizzas are the most popularly ordered pizzas on Domino's menu?	How many calories does a Dominos pizza have?	0
3272	6542	6543	How do you start a bakery?	How can one start a bakery business?	1

Table E.1: Sample data in Quora

Bibliography

[1] Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003.

[2] Elizabeth D Liddy. Natural language processing. 2001.

[3] Kam-Fai Wong, Mingli Wu, and Wenjie Li. Extractive summarization using supervised and semi-supervised learning. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 985–992. Association for Computational Linguistics, 2008.

[4] Ani Nenkova and Kathleen McKeown. A survey of text summarization techniques. In *Mining text data*, pages 43–76. Springer, 2012.

[5] Udo Hahn and Inderjeet Mani. The challenges of automatic summarization. *Computer*, 33 (11):29–36, 2000.

[6] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[7] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 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. Association for Computational Linguistics, 2007.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[10] Alan Ritter, Sam Clark, Oren Etzioni, et al. Named entity recognition in tweets: an experimental study. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1524–1534. Association for Computational Linguistics, 2011.

[11] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *arXiv preprint arXiv:1511.08308*, 2015.

[12] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[13] Bing Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[15] Frederick Jelinek. *Statistical methods for speech recognition.* MIT press, 1997.

[16] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[17] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

[18] Freddy YY Choi. Advances in domain independent linear text segmentation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 26–33. Association for Computational Linguistics, 2000.

[19] Nick C Ellis and Diane Larsen-Freeman. *Language as a complex adaptive system*, volume 3. John Wiley & Sons, 2009.

[20] ChengXiang Zhai. Statistical language models for information retrieval. *Synthesis Lectures on Human Language Technologies*, 1(1):1–141, 2008.

[21] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[22] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 55–64. ACM, 2016.

[23] X Huang and S Robertson. A probabilistic approach to chinese information retrieval: theory and experiments. In *Proceedings of the BCS-IRSG*, pages 178–193, 2000.

[24] Zheng Ye and Jimmy Xiangji Huang. A simple term frequency transformation model for effective pseudo relevance feedback. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 323–332. ACM, 2014.

[25] Jiashu Zhao, Jimmy Xiangji Huang, and Shicheng Wu. Rewarding term location information to enhance probabilistic information retrieval. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1137–1138. ACM, 2012.

[26] Qin Chen, Qinmin Hu, Jimmy Huang, and Liang He. TAKer: Fine-grained time-aware microblog search with kernel density estimation. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[27] M Beaulieu, M Gatford, Xiangji Huang, S Robertson, S Walker, and P Williams. Okapi at TREC-5. *NIST SPECIAL PUBLICATION SP*, pages 143–166, 1997.

[28] Yang Liu, Xiangji Huang, Aijun An, and Xiaohui Yu. ARSA: a sentiment-aware model for predicting sales performance using blogs. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 607–614. ACM, 2007.

[29] Xiangji Huang, Fuchun Peng, Dale Schuurmans, Nick Cercone, and Stephen E Robertson. Applying machine learning to text segmentation for information retrieval. *Information Retrieval*, 6(3-4):333–362, 2003.

[30] Ben He, Jimmy Xiangji Huang, and Xiaofeng Zhou. Modeling term proximity for probabilistic information retrieval models. *Information Sciences*, 181(14):3017–3031, 2011.

[31] Xiangji Huang and Qinmin Hu. A bayesian learning approach to promoting diversity in ranking for biomedical information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 307–314. ACM, 2009.

[32] Xiaoshi Yin, Jimmy Xiangji Huang, Zhoujun Li, and Xiaofeng Zhou. A survival modeling approach to biomedical search result diversification using wikipedia. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1201–1212, 2013.

[33] Jiashu Zhao, Jimmy Xiangji Huang, and Ben He. CRTER: using cross terms to enhance probabilistic information retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 155–164. ACM, 2011.

[34] Robert F Simmons. Answering english questions by computer: a survey. *Communications of the ACM*, 8(1):53–70, 1965.

[35] 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.

[36] 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 ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 468–476. Association for Computational Linguistics, 2009.

[37] Zornitsa Kozareva and Andrés Montoyo. Paraphrase identification on the basis of supervised machine learning techniques. In *Advances in natural language processing*, pages 524–533. Springer, 2006.

[38] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*, pages 801–809, 2011.

[39] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[40] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[41] Minghui Qiu, Feng-Lin Li, Siyu Wang, Xing Gao, Yan Chen, Weipeng Zhao, Haiqing Chen, Jun Huang, and Wei Chu. Alime chat: A sequence to sequence and rerank based chatbot engine. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 498–503, 2017.

[42] 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. In *SemEval@ COLING*, pages 1–8, 2014.

[43] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJcAI*, volume 7, pages 1606–1611, 2007.

[44] Michael Mohler and Rada Mihalcea. Text-to-text semantic similarity for automatic short answer grading. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 567–575. Association for Computational Linguistics, 2009.

[45] Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, pages 775–780, 2006.

[46] Courtney Corley and Rada Mihalcea. Measuring the semantic similarity of texts. In *Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment*, pages 13–18. Association for Computational Linguistics, 2005.

[47] 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. ACM, 2015.

[48] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[50] Shlomo Mor-Yosef, Arnon Samueloff, Baruch Modan, Daniel Navot, and Joseph G Schenker. Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study. *Obstetrics and gynecology*, 75(6):944–947, 1990.

[51] Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D Spyropoulos, and Panagiotis Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *arXiv preprint cs/0009009*, 2000.

[52] William Stafford Noble et al. Support vector machine applications in computational biology. *Kernel methods in computational biology*, pages 71–92, 2004.

[53] Valentin Jijkoun, Maarten de Rijke, et al. Recognizing textual entailment using lexical similarity. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, pages 73–76. Citeseer, 2005.

[54] Aminul Islam and Diana Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2 (2):10, 2008.

[55] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142, 2003.

[56] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press, 2008.

[57] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Computational linguistics*, 37(2):351–383, 2011.

[58] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection. In *Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics*, pages 45–52, 2008.

[59] Rafael Ferreira, Rafael Dueire Lins, Fred Freitas, Steven J Simske, and Marcelo Riss. A new sentence similarity assessment measure based on a three-layer sentence representation. In *Proceedings of the 2014 ACM symposium on Document engineering*, pages 25–34. ACM, 2014.

[60] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. * sem 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, volume 1, pages 32–43, 2013.

[61] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[62] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

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

[64] Tom Kenter and Maarten De Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1411–1420. ACM, 2015.

[65] Cedric De Boom, Steven Van Canneyt, Steven Bohez, Thomas Demeester, and Bart Dhoedt. Learning semantic similarity for very short texts. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, pages 1229–1234. IEEE, 2015.

[66] Adrian Sanborn and Jacek Skryzalin. Deep learning for semantic similarity. *CS224d: Deep Learning for Natural Language Processing. Stanford, CA, USA: Stanford University.*

[67] 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.

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

[69] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[70] Qin Chen, Qinmin Hu, Jimmy Xiangji Huang, and Liang He. CA-RNN: Using context-aligned recurrent neural networks for modeling sentence similarity. In *AAAI*, pages 265–273, 2018.

[71] Qin Chen, Qinmin Hu, Jimmy Xiangji Huang, and Liang He. CAN: Enhancing sentence similarity modeling with collaborative and adversarial network. In *Proceedings of the 41st international ACM SIGIR conference on Research and development in information retrieval.* ACM, 2018.

[72] 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 Conference on information & knowledge management*, pages 2333–2338. ACM, 2013.

[73] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 373–374. ACM, 2014.

[74] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *AAAI*, pages 2786–2792, 2016.

[75] Paul Neculoiu, Maarten Versteegh, Mihai Rotaru, and Textkernel BV Amsterdam. Learning text similarity with siamese recurrent networks. *ACL 2016*, page 148, 2016.

[76] Zhengdong Lu and Hang Li. A deep architecture for matching short texts. In *Advances in Neural Information Processing Systems*, pages 1367–1375, 2013.

[77] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. Text matching as image recognition. In *AAAI*, pages 2793–2799, 2016.

[78] Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

[79] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[80] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[81] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[82] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[83] Thomas M Breuel. The effects of hyperparameters on sgd training of neural networks. *arXiv preprint arXiv:1508.02788*, 2015.

[84] Christopher Olah. Understanding lstm networks. *GITHUB blog, posted on August*, 27:2015, 2015.

[85] Pengfei Liu, Shafiq R Joty, and Helen M Meng. Fine-grained opinion mining with recurrent neural networks and word embeddings. In *EMNLP*, pages 1433–1443, 2015.

[86] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

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

[88] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994.

[89] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *arXiv preprint arXiv:1512.05193*, 2015.

[90] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[91] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs, 2017.

[92] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.