MULTIRESOLUTION GRAPH ATTENTION NETWORKS FOR RELEVANCE MATCHING

by

Ting Zhang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Various of deep learning models have been proposed for the text matching problem, which is at the core of various typical natural language processing (NLP) tasks. However, existing deep models are mainly designed for the semantic matching between a pair of short texts, such as paraphrase identification and question answering, and do not perform well on the task of relevance matching between *short-long* text pairs. This is partially due to the fact that the essential characteristics of short-long text matching have not been well considered in these deep models. More specifically, these methods fail to handle extreme length discrepancy between text pieces and neither can they fully characterize the underlying structural information in long text documents.

In this thesis, we are especially interested in relevance matching between a piece of short text and a long document, which is critical to problems like query-document matching in information retrieval and web searching. To extract the structural information of documents, an undirected graph is constructed, with each vertex representing a keyword and the weight of an edge indicating the degree of interaction between keywords. Based on the keyword graph, we further propose a *Multiresolution Graph Attention Network* to learn multi-layered representations of vertices

through a Graph Convolutional Network (GCN), and then match the short text snippet with the graphical representation of the document with an attention mechanism applied over each layer of the GCN. Moreover, we develop deeper insights into the GCN model in [1] and address its limits to weighted graphs. Experimental results on two benchmark datasets demonstrate that our graph approach outperforms other state-of-the-art deep matching models.

Acknowledgments

I would like to thank all the people who contributed in some way to the work described in this thesis. Firstly, I would like to express my sincere gratitude to my advisor, Dr. Di Niu. His guidance and supervision help me to learn how to be a good researcher and pursue an academic career. Their willingness to discussion helped me through two important years of my life. Additionally, I would like to thank my committee members Dr. Marek Reformat and Dr. Linglong Kong for their interest in my work.

Finally, I would like to acknowledge friends and family who supported me during my time here. Thanks for their friendship and unyielding support!

Table of Contents

1 Introduction **1**

2 Preliminaries and Background **5**

2.1 Relevance Matching Problem 5

2.2 Related Work 8

2.2.1 Unsupervised Algorithms 8

2.2.2 Deep Matching Models 10

3 Document as Graph **13**

3.1 Document Graph Representation 13

3.2 Keyword Graph Construction 14

3.2.1 Document Preprocessing 15

3.2.2 Keyword Extraction 16

3.2.3 Edge Construction 17

4 Multiresolution Graph Attention Network **19**

4.1 Query Embedding and Encoding 20

4.2 Vertex Encoding based on Graph Convolutional Network 21

4.2.1 Graph Convolutional Network for Weighted Graphs 21

4.2.2 Vertex Embedding 24

4.3 Rank-and-Pooling Layer 26

4.4 Attention-based Query-Graph Matching 27

4.5 Aggregation Layer 28

5 Experiments **29**

5.1 Description of Tasks and Datasets 29

5.2 Competitor Methods 31

5.3 Performance Analysis 32

5.4 Impact of Different Modules and Parameters 35

6 Conclusion **40**

List of Tables

2.1 Examples to show the differences between relevance matching and semantic matching. 7

5.1 Description of evaluation datasets. 29
5.2 Average length of text in evaluation datasets. 30
5.3 Accuracy and F1-score results of different algorithms on the Ohsumed dataset. 32
5.4 Accuracy and F1-score results of different algorithms on the NFCorpus dataset. 33
5.5 Accuracy and F1-score results of MGAN and its variants on the Ohsumed dataset. 35
5.6 Accuracy and F1-score results of MGAN and its variants on the NFCorpus dataset. 36

List of Figures

2.1 Architecture of representation-focused deep models. 11
2.2 Architecture of interaction-focused deep models. 12

3.1 An example to show a piece of document and its corresponding Keyword Graph representation. 16

4.1 An overview of the proposed *Multiresolution Graph Attention Network* (MGAN) for matching a short query and a long text document. 20
4.2 The impact of convolution on each vertex. 26

5.1 Compare the accuracies given by different λ on the Ohsumed dataset. 38
5.2 Compare the F1 scores given by different λ on the Ohsumed dataset. 39

List of Abbreviations

Acronyms	Definition
GCN	Graph Convolutional Network
DGCNN	Deep Graph Convolutional Neural Network
BOW	Bag-of-Words (
LSI	Latent Semantic Indexing
LSA	Latent Semantic Analysis
SVD	Singular Value Decomposition
WL	Weisfeiler-Lehman
OOV	Out-of-Vocabulary
BM25	Okapi BM25
TF-IDF	Term Frequency-Inverse Document Frequency
FT	Fourier Transform
ARC-I	Convolutional Matching Architecture-I
ARC-II	Convolutional Matching Architecture-II
DSSM	Deep Structured Semantic Models
C-DSSM	Convolutional Deep Structured Semantic Models
MV-LSTM	Multiple Positional Semantic Matching

Chapter 1

Introduction

Matching two pieces of text has long been a core research problem underlying numerous natural language processing tasks. The past few years have seen the great success of deep models [2]–[5] for semantic matching tasks such as question answering (QA) [6], paraphrase identification [7] and automatic conversation [8] etc. However, it is still challenging to estimate the relevance between a pair of *short* and *long* text pieces. For example, in query-document matching, user queries usually contain a few words, while the lengths of documents could vary from hundreds to thousands of words. Given rich semantic and syntactic structures that exist in long documents and the extreme discrepancy between the lengths of queries and documents, accurately estimating the relevance between them is hard.

Existing methods for text matching are typically categorized into three types including unsupervised metrics [9], feature-based models and deep matching models [2]–[5]. For unsupervised metrics, text documents are transferred to vectors with representation methods such as bag-of-words (BOW). Then the distance between vectors are calculated according to metrics like euclidean distance, cosine similarity and so on. However, such approaches are principally based on the term frequency and ignore the semantic structures of natural language. Thus leading to poor performance for complicated tasks. Feature-based models, or feature engineering [10] rely on hundreds or thousands of handcrafted features. In reality, search engines also depend on other auxiliary information like click history, nu-

merous ad-hoc rules and metadata, etc., to boost query-document matching performance. Obviously, handcrafting features is time-consuming, possibly incomplete and application-specific.

Recently, a variety of deep models have also been applied to text matching, e.g., [2]–[5], which can be divided into two categories depending on the model structures: representation-focused and interaction-focused. Representation-focused deep models [3], [4] take the word embedding sequences of a pair of text objects as the input, and learn its intermediate contextual representation through a Siamese convolutional or recurrent neural network, on which final scoring is performed. But for interaction-focused deep models [2], [5], which focus on local interactions between two pieces of text and learn the complex interaction patterns for relevance with deep neural networks. Comparing to unsupervised metrics and feature-based models, deep matching models are generalized while maintaining high accuracy in various NLP tasks.

However, we show that most existing deep models do not yield a satisfactory performance for relevance matching between a pair of *short* and *long* text objects. It is due to the essential differences between semantic matching and relevance matching. Semantic matching tasks, such as paraphrase identification and semantic textual similarity, concentrate on identifying the semantic meaning and inferring the semantic relations between two pieces of text. While relevance matching tasks, such as query document matching in information retrieval, care more about whether the query and document are related or not instead of whether they express the same semantic meaning or not. We figured out that most existing deep matching models [2]–[5], whether they are representation-focused or interaction-focused, mainly concern semantic matching rather than relevance matching. Also, we point out that current deep models [2]–[5] are effectively dealing with text snippets, e.g., a pair of sentences, but have difficulty handling extreme short text and long documents. On one hand, encoding the query consisting of only few words with complicated deep models usually results in excessive deformation. On the other hand, it is more likely to introduce "noise" and redundant information when dealing with long documents

using deep models.

To address the above problems, we propose a deep relevance matching model based on graph and attention mechanisms to improve the matching between a pair of short and long text objects. We show that an appropriate semantic representation, beyond a linear sequence of word vectors [11], of a document plays a central role in relevance matching. Documents are represented as undirected, weighted *Keyword Graph*, in which each vertex is a keyword in the document, and edges indicate the degree of relevance between two corresponding keywords. Such a graphical representation helps to reveal the inner structure of document focuses. Based on such representation, the problem of relevance matching is transformed into a query-graph matching problem.

To match the query and keyword graph of a document, we designed a novel deep matching model called the *Multiresolution Graph Attention Network* (MGAN). It learns a multiresolution representation for each keyword vertex through a multi-layered graph convolutional network (GCN) [1], [12], an emerging variant of convolutional neural networks that specifically encodes graphs. Moreover, we develop deeper insights into the GCN model and improve it to better cope with weighted graphs. By applying attention mechanisms to word vectors of the query with the keyword representations learned by each layer of the GCN, MGAN is able to characterize the relevance between the query and keywords of the document, utilizing multiresolution representations of keywords generated in different layers.To handle the varying number of keywords in different documents, a *rank-and-pooling* strategy is proposed to sort and select keyword vertices. In each layer, we choose a fixed number of query-keyword matching results, and concatenate them together for aggregation. The final relevance score is generated by feeding the concatenated matching vector into a multilayer perceptron network.

We evaluated our model on two datasets for different tasks, including the Ohsumed dataset for topic-document matching and the NFCorpus dataset for query-document matching. Experimental results demonstrate that our model boasts significantly improved performance compared with existing state-of-the-art deep matching models,

including ARC-I [2], ARC-II [2], DSSM [13], C-DSSM [14], MV-LSTM [4], and MatchPyramid [5].

The remainder of this thesis is organized as follows. Chapter 2 formally introduces the problem of relevance matching and analyzes the characteristics as well as challenges of this problem. Besides, we also review the related literature of deep matching models in this chapter. Chapter 3 presents the methods of document representations, especially discuss the graph representations of documents. Then, we describe the process of our own keyword graph construction of long documents. In Chapter 4, we propose a novel Multiresolution Graph Attention Network (MGAN) for relevance matching of short-long text pairs based on the graph. Moreover, we develop deeper insights into the GCN model in [1] and address its limits to weighted graphs. Experimental results are demonstrated in Chapter 5 and finally conclude the thesis in Chapter 6.

Chapter 2

Preliminaries and Background

The goal of relevance matching is to determine whether two pieces of text are related or not, or inferring the relevance degree between them. It is a core problem in many real-word applications such as information retrieval and web searching. In this chapter, we formally define the problem of relevance matching, and compare it with the problem of semantic matching, which is another typical and significant task in natural language processing. Besides, we analyze the underlying characteristics of relevance matching between short text and long document, and point out the challenges when dealing with this kind of problem. Prior work is also included in this chapter, more specifically, we introduce various methodologies which are widely used in text matching and information retrieval. Particularly, we review the related literature of deep matching models, which are designed for the text matching problem and have achieved great success in recent years.

2.1 Relevance Matching Problem

Here, we formally introduce the problem of relevance matching, and show the differences between relevance matching and semantic matching. Most importantly we point out the difficulties of matching the relevance between query and document, in other words, the challenges of matching extreme short text and long document.

Denote a query as q and a text document as d. Given a query-document pair

(q, d), the relevance matching problem can be formalized as:

$$r = \mathcal{F}(\phi_q(q), \phi_d(d)) \tag{2.1}$$

where ϕ_q and ϕ_d are representation functions that map query and document to their feature space. $\mathcal{F}$ is the scoring function based on the interactions between query and document. The relevance score r can be binary or numerical: binary r indicates whether the text pair is related or not, while numerical r reflects the relevance degree between a query and a document. This brand of text matching problems are generally related to a variety of NLP tasks, such as information retrieval that matches user queries with document collections.

A lot of deep matching models have been proposed [2]–[5], and most of them have only been demonstrated to be effective on a set of NLP tasks such as semantic textual similarity, paraphrase identification, question answering [15] and so on. However, when apply these deep models on relevance matching problem such as the task of query document matching in information retrieval, their performance is usually disappointing.

This is due to some fundamental differences between the tasks of semantic matching and relevance matching, as pointed out by [15]. The goal of semantic matching is to understand the semantic meaning of the text or infer the relationship between two pieces of text, which are usually homogeneous sentences. However, relevance matching focuses on deciding whether two pieces of text are describing the same event or relevant events. Two examples in Table 2.1 show the differences between relevance matching and semantic matching. For instance, "A man is playing basketball." is semantically similar with "A man is playing football.", but these two sentences are not relevant. Another example is that "The stock price of Apple is increasing." is relevant to "Apple is an excellent company.", but they are not semantically similar. Compared with semantic matching, relevance matching emphasizes more on the exact matching signals between query keywords and a document. Actually, most existing models are concerned about *semantic matching* problem, such

TABLE 2.1
EXAMPLES TO SHOW THE DIFFERENCES BETWEEN RELEVANCE MATCHING AND SEMANTIC MATCHING.

Text1	Text1	Relationship
"A man is playing basketball."	"A man is playing football."	Semantic Similar
"The stock price of Apple is increasing."	"Apple is an excellent company."	Relevant

as paraphrase identification, question answering [15] and so on, but few of them consider the characteristics of the relevance matching.

Besides, in the task of query document matching, query and document vary considerably in text length and provide unbalanced information for directly matching. The query is usually extremely short and consists of only few words, while the document varies considerably in length, from tens of words to tens of thousands of words. Current deep models [2]–[5] are effectively dealing with text snippets, e.g., a pair of sentences, but have difficulty handling extreme short text and long documents in query document matching tasks. On one hand, encoding the query consisting of only few words with complicated deep models usually results in excessive deformation. On the other hand, it is more likely to introduce "noise" and redundant information when dealing with long documents using deep models.

What is more, most existing approaches consider text pieces as sequences of words or word vectors. However, the semantic structure information of text pieces is not fully utilized, which can be helpful for the task of relevance matching between queries and documents, especially when the document length is long. In the next chapter, we will introduce our proposed procedures to transform a document into a keyword graph. Such a graph representation proves to be effective at uncovering the underlying attention structure of a long text document such as a news article.

2.2 Related Work

Most existing works on text matching problem can be generalized into two categories: unsupervised algorithms and deep matching models. For deep matching models, we can further divide them into representation-focused deep neural models and interaction-focused deep neural models [16]. In this section, we will have a brief introduction about some common algorithms of text matching and information retrieval.

2.2.1 Unsupervised Algorithms

The most straight forward method for text matching, especially for the task of query document matching in information retrieval, is lexical matching [17]. For example, simply match the keywords in document with terms in query. However, in most cases, lexical matching can be unreliable. On one hand, there are usually many ways to express a given concept (synonymy), the literal terms in a user's query may not match those of a relevant document. On the other hand, one word usually have multiple explanations and can represent totally different concepts (polysemy) according to the context. For example, "Apple" is usually referred to a kind of fruit, but sometimes it is used to demonstrate a company in a piece of technology news. Thus synonymy and polysemy are two main challenges for lexical matching methods in the text matching problem.

Instead of exactly lexical matching, bag-of-words (BOW) [18] model converts NLP text into numbers and matching text based on statistics. This process is called vectorization and these transformed text vectors can further be implemented on machine learning models or unsupervised metrics. BOW is widely utilized in natural language processing tasks, which represents the text as an unordered collection of words. Each word is scored by designed metrics or functions to evaluate the occurrence of words. Some additional simple scoring methods include count, frequency and so forth. Among these metrics, term frequency-inverse document frequency

(TF-IDF) is mostly used as a strong baseline, which is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. After representing the text as a word vector, we then calculate the distance or similarity between vectors with euclidean distance, cosine correlation, Jackard coefficient and so on. Another metric Okapi BM25 (BM25) [19] based on the probabilistic model is also widely implemented in information retrieval task. However, these models are based on the assumption that words in the text (such as sentence or document) are independent, disregarding the word order as well as the semantic meaning of each word.

Despite above algorithms, topic models [20] are also attractive in text matching problem. Latent semantic indexing (LSI) [21], or latent semantic analysis (LSA) is one of empirically topic models for text matching problem. LSI assumes that there is some underlying or latent structure in word usage that is partially obscured by variability in word choice. A truncated singular value decomposition (SVD) is used to estimate the structure in word usage across documents. More specifically, given m documents, and we select n words with ID from 1 to n. Matrix $A \in \mathbb{R}^{m \times n}$ is formalized with A_{ij} represents the feature value (such as TF-IDF) of j_{th} word in i_{th} document. With SVD, $A_{m \times n}$ can be decomposed into

$$A_{m \times n} = U_{m \times m} \Sigma_{m \times n} V^T_{n \times n} \tag{2.2}$$

To reduce the dimensions from n to k, the decomposition Eq. 2.2 can be approximated to

$$A_{m \times n} \approx U_{m \times k} \Sigma_{k \times k} V^T_{k \times n} \tag{2.3}$$

where k represents the number of topics, and k is usually smaller than n. U_{il} denotes the relevance degree between i_{th} document and l_{th} topic. To calculate the text similarity, cosine coefficient can be implemented.

LSI shows the ability to avoid the independent assumption mentioned above and try to explore the second-order co-occurrence in the text. However, the weakness of LSI is obvious for huge computation in SVD process. Additionally, although

LSI takes the latent semantic of words into consideration, it is still far away from understanding the semantic meaning and contextual information of the whole text.

Feature-based models, such as IRGAN [10], are generally applied in industry. These models rely on hundreds or thousands of handcrafted features, such as TF-IDF, BM25 [19], string similarity and so on. In reality, search engines also depend on other auxiliary information like click history, numerous ad-hoc rules and meta-data, etc., to boost query-document matching performance. However, handcrafting features are time-consuming, incomplete and over-specified. We need to design features carefully for each different task and dataset.

2.2.2 Deep Matching Models

Deep learning models have seen great success in various natural language processing tasks such as paraphrase identification, question answering and so on. These deep models can be divided into two categories depending on their structures: representation-focused models and interaction-focused models. In this subsection, we will introduce different kinds of state-of-art deep matching models.

Representation-focused deep neural matching models usually transform the word embedding sequences of text pairs into context representation vectors through a neural network encoder, followed by a fully connected network or score function which gives the matching result based on the representation vectors. Such models include Siamese RNN [22], HCTI [23], DSSM [13], C-DSSM [14] and ARC-I [2]. Typically, they have the Siamese structure and intend to map the text into the semantic feature space by neural networks such as CNN and LSTM. Fig. 2.1 shows the Siamese structure of representation-focused deep neural matching models, where the parameters of neural networks are usually shared. The representation-focused model are flexible, since both the representing neural networks and aggregation layer can be customized according to the characteristics of the text. However, the Siamese architecture defers the interaction between two sentences, which will lead to the loss of details.

Interaction-focused models build local interactions between words and aggre-

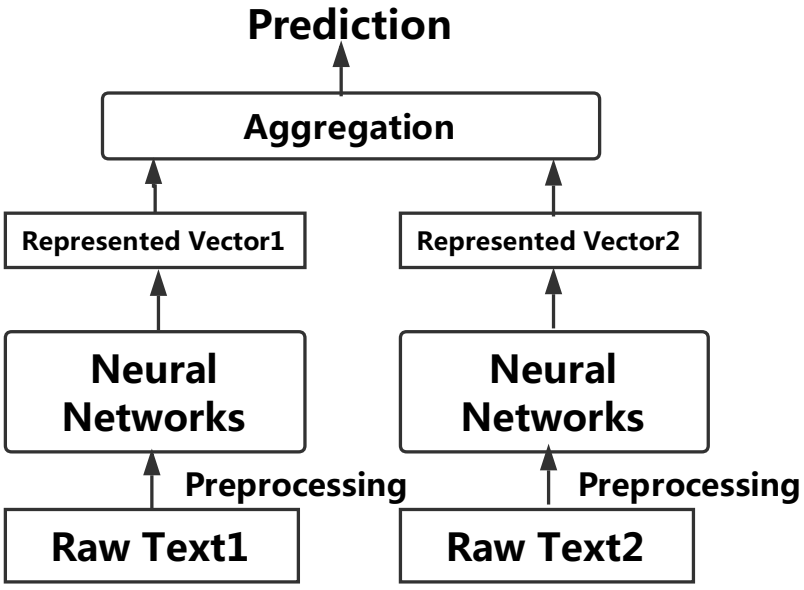

Fig. 2.1. Architecture of representation-focused deep models.

gate the interaction features to give a matching result. Examples include ARC-II [2], DeepMatch [24] and MatchPyramid [5]. Fig. 2.2 shows the model structure of MatchPyramid, which is a classic interaction-focused deep model and demonstrates positive performance on a number of datasets. Different from representation-focused models, interaction-focused models compare two sentences and extract the matching features before their own high-level representations, to retain more information about the text from word level and phrase level.

Deep matching models are generalized while maintaining high accuracy in various NLP tasks, such as semantic textual similarity and question answering. The reason is that deep models fully exploit both semantic meaning of words and the sequence of words, to solve the synonymy and polysemy in text matching problem. A large amount of references have shown the priority of deep matching models comparing to unsupervised metrics and feature-based models. It is easy to find that these deep models are mainly used to sentences, however, when apply them to a

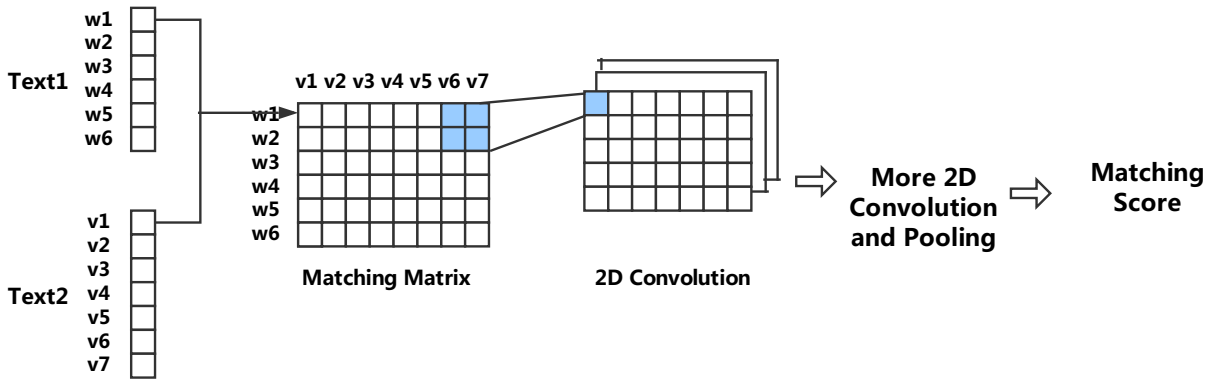

Fig. 2.2. Architecture of interaction-focused deep models.

pair of short and long text objects, such as query document matching in information retrieval, the results are usually disappointing. To address the issues mentioned in Sec. 2.1, we propose a novel model in the following Chapters.

Chapter 3

Document as Graph

Representing text document as graphs can model relationship and structural information effectively. In this Chapter, various methods on modeling of text document using Graph are introduced. Moreover, we demonstrate the construction process of our designed keyword graph.

3.1 Document Graph Representation

Various of graph representations have been proposed for document modeling. According to the different types of graph nodes, a majority of existing works can be generalized into four categories: word graph, text graph, concept graph, and hybrid graph.

For word graphs, the graph nodes represent different non-stop words in a document. [25] extracts subject-predicate-object triples from text based on syntactic analysis, and merge them to form a directed graph. The graph is further normalized by utilizing WordNet [26] to merge triples belonging to the same semantic pattern. [27], [28] represent a document as graph-of-word, where nodes represent unique terms and directed edges represent co-occurrences between the terms within a fixed-size sliding window. [29] connect terms with syntactic dependencies. [30] connects two words by directed edge if one word is immediately precedes another word in document title, body or link. The edges are categorized by the three differ-

ent types of linking.

Text graphs use sentences, paragraphs or documents as vertices, and establish edges by word co-occurrence, location or text similarities. [31]–[33] connect sentences if they near to each other, share at least one common keyword, or sentence similarity is above a threshold. [34] connects web documents by hyperlinks. [35] constructs directed weighted graphs of sentences for evaluating text coherence.

For concept graphs, they link terms in a document to real world entities or concepts based on resources such as DBpedia [36], WordNet [26], VerbNet [37] and so forth. [38] identifies the semantic roles in a sentence using WordNet and VerbNet, and combines these semantic roles with a set of syntactic/semantic rules to construct a concept graph.

Hybrid graphs contains multiple types of vertices and edges. [39] uses sentences as vertices and encodes lexical, syntactic, and semantic relations in edges. [40] extract tokens, syntactic structure nodes, semantic nodes and so on from each sentence, and link them by different types of edges. [41] builds a sentence graph based on Frame Semantics and Construction Grammar.

3.2 Keyword Graph Construction

In our case, as the relevance between a short query and a long document mostly relies on the relations between query words and document keywords, therefore, we construct a graph of keywords to represent a document. To address the challenges of the relevance matching problem mentioned above, we model the document as a weighted, undirected *keyword graph*. The aim of this graph representation is to model the interaction structure of document keywords, as well as uncovering the term importance of keywords induced by the topological structure of keyword interactions. Compared with linear representation of text pieces, a graphical representation can better capture the rich intrinsic semantic structures in long text objects. Furthermore, it is helpful in overcoming the long-distance dependency problem in NLP, as it breaks the linear organization of words.

We first describe the structure of a document keyword graph before presenting the detailed steps to derive it. Given an input document $\mathcal{D}$, our objective is to obtain a graph representation G_D of $\mathcal{D}$. Each vertex in G_D is a keyword in document $\mathcal{D}$. We link two vertices by an edge if the word distance of the two keywords in the document is smaller than a threshold (we set the threshold as 20 in our experiments). The edge weight is proportional to the inverse of the word distance between two keywords.

As a toy example, Fig. 3.1 illustrates how we convert a document into a keyword graph. We can extract keywords or key phrases such as *ZTE*, *Qualcomm*, *US Department of Commerce*, *export* and so on from the document using common keyword extraction algorithms [42]. These keywords represent the topics or concerns in this document. We then connect the keyword vertices by weighted edges, where the edge weight between a pair of keywords denotes how close they are related, and the whole topological structure of the keyword graph shows the semantic structure of the document. For example, in Fig. 3.1, *export* is highly correlated with *ZTE*, *Chinese*, *American* and so on. In this way, we have transformed the original document into a graph of different focal points, as well as the interaction topology among them.

We now introduce our detailed procedure to restructure a document $\mathcal{D}$ into a desired keyword graph G_D as described above. The whole process consists of three steps: 1) document preprocessing, 2) keyword extraction, and 3) edge construction.

3.2.1 Document Preprocessing

Off-the-shelf NLP tools such as Stanford CoreNLP [43] can be utilized to preprocessing the input documents. The first step is tokenization, which splits long strings of text into smaller pieces, or tokens. Text can be tokenized into sentences, and sentences can be tokenized into words. Secondly, we need to clean the data, that is removing stop words and meaningless characters to make text neat. Then, we extract named entities from the document. Named entity is defined as a real-world object, such as persons, locations, organizations, products and so on. For docu-

Document:

The **US Department of Commerce** just announced a **ban** on **American exports** to the **Chinese smartphone** maker **ZTE**. That means **American** companies like **Dolby** and **Qualcomm** won't be able to **export** any parts to **ZTE** for up to seven years. The loss of **Qualcomm** is particularly damaging, as it severely restricts **ZTE**'s options for devices in the US market.

Keyword Graph:

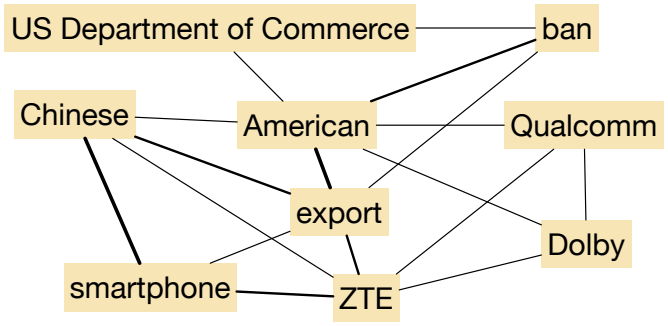

Fig. 3.1. An example to show a piece of document and its corresponding Keyword Graph representation.

ments, especially news articles, the named entities are usually critical keywords.

3.2.2 Keyword Extraction

The next step is to extract the keywords of documents. As the named entities alone are not enough to cover the main focuses of the document, we therefore apply a keyword extraction algorithm to expand the keyword set. There are different algorithms for keyword extraction [42], such as TF-IDF, TextRank, RAKE and so on. Since TF-IDF takes the advantages of wide generality and high efficiency, we implemented it in our experiments. More specifically, we first calculate the term frequency–inverse document frequency (TF-IDF) value for each token, and choose the top 20 percentage tokens to expand the set of document keywords. Even though more sophisticated algorithms may achieve better performance for the keyword extraction, in this thesis, we concentrate on the graph modeling of documents and the algorithm of relevance matching. After we extract the set of keywords from a document, each keyword will be a vertex in the document's graph.

3.2.3 Edge Construction

Our last step is linking correlated keywords in the document by weighted edges. For each pair of keyword vertices v_i and v_j, we calculate the word distance d_{ij} in the document. Suppose keyword v_i shows m times in the document and keyword v_j shows n times in the document, with $m \leq n$. For each v_i, we select the v_j that is most close to it, and calculate the word distance d_{ij}^t for t_{th} keyword v_i. The distance d_{ij} is the mean distance between each v_i and its most nearby v_j. Based on the word distance d_{ij}, the weight w_{ij} of the edge e_{ij} between v_i and v_j is calculated as

$$w_{ij} = g(d_{ij}) = \frac{1}{d_{ij}} = \frac{m}{\sum_{t=1}^{m} d_{ij}^t}. \tag{3.1}$$

Now, we have transformed an input document into a weighted undirected graph of keywords. Compared with the original document's sequential structure, a graph structure organized keywords in terms grants a correlation structure. Therefore, the problem of long distance dependency can be alleviated as related keywords are linked by weighted edges. Furthermore, the weighted edges represent the strengths of interactions among these concepts. Together with the topology structure of the whole graph, we can also model the importance of different keyword in the document. A keyword with a lot of edges linking it to other keywords is usually more important than other keywords that only have a few edges. A keyword that has strong connections with other keywords (i.e., the edge weights are large) are typically more important than keywords that only have edges with small weights.

There are also existing works that model a document as a graph of sentences [31]–[33], or construct vertices and edges via more complicated methods, such as linking terms in a document to real world entities or concepts based on resources. On such example is DBpedia [36], which extracts subject-predicate-object triples from text based on syntactic analysis to construct directed edges [25], and so forth. However, as the problem of relevance matching is more focused on the exact matching signals between query keywords and a document, we therefore choose to model the correlation between keywords of a document, rather than using sentences as

graph nodes. Compared with constructing a keyword graph with complicated mechanisms rooted in a knowledge base or performing syntactic analysis, which are usually time-consuming, we choose to model the structure of keyword correlations by a more efficient procedure described above to make it available for real world industry applications. We will see that our keyword graph is both efficient and able to improve the performance of relevance matching tasks when combined with the *Multiresolution Graph Attention Network* model, which we will describe in detail in the next chapter.

Chapter 4

Multiresolution Graph Attention Network

In this Chapter, we further exploit the keyword graph representation of documents, and propose a deep relevance model based on multi-layer graph convolutional networks and attention-based matching, namely Multiresolution Graph Attention Network (MGAN), for query document matching. Fig. 4.1 illustrates the overall architecture of our proposed model, which mainly has five sequential stages. First, query and vertices in the document graph are embedded with word vectors such as GloVe [11]. Second, the embedded query and document graph are respectively encoded with convolutional layers. Specifically, for the document graph, graph convolutional layers are implemented to extract the local features of vertices and iteratively revise the encoding vectors. Third, a Rank-and-Pooling layer is utilized to sort the vertices in a specific order and unify the graph size. Next, we compute the matching scores between query and selected vertices in each graph convolutional layer based on the attention mechanism. Finally, these matching scores are concatenated as a match vector and fed into the aggregation layer to get the final relevance matching result. We will describe each layer in detail in the following.

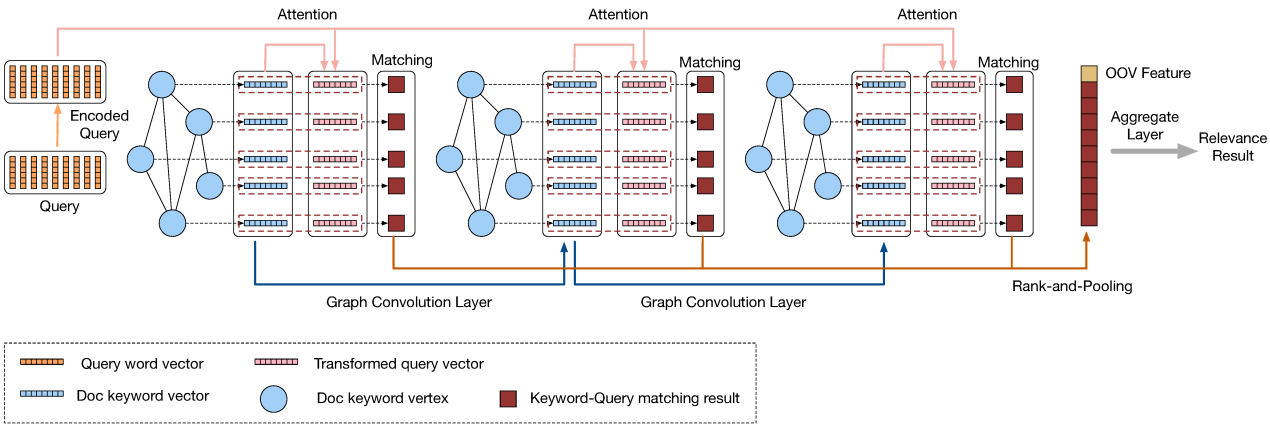

Fig. 4.1. An overview of the proposed *Multiresolution Graph Attention Network* (MGAN) for matching a short query and a long text document.

4.1 Query Embedding and Encoding

The embedding layer turns each token of the query and each keyword of the document into a dense vector. Given a query with d_q words, a document graph with d_g vertices and a d_e dimensional pre-trained embedding vectors, we will get a query embedding matrix $Q_{\text{emb}} \in \mathbb{R}^{d_e \times d_q}$ and a graph vertex feature matrix $G_{\text{emb}} \in \mathbb{R}^{d_e \times d_g}$ after the word embedding layer. In this work, we utilize the pre-trained, 300-dimensional Glove Word Vectors [11] for word embedding in our experiments. Word Embedding allows the word matching in the semantic level, instead of the term level such as bag-of-word.

After we embedded the query, we further use a simple 1D convolutional neural network (CNN) as an encoder to produce a refined encoding representation $Q \in \mathbb{R}^{d_e \times d_q}$ of the query, where the i-th column in Q is the context vector of token i that incorporates the contextual information in the query.

Notice that the out-of-vocabulary (OOV) words, which are not able to be embedded, can still play significant roles in the matching. Especially for a query with only 2 or 3 terms, in this case, each word counts and should not be ignored. To fully exploit these OOV words, we match them on a term level by calculating how many common OOV words x_{oov} are in the query and document graph. x_{oov} is defined as the OOV feature, and will be concatenated to the final match vector, which we will

describe later.

It is worth mentioning that we can potentially further improve the performance of our neural network model by concatenating the character-level embedding and feature embedding of words to form the final word representation. A character-level embedding of a word (or token) can be obtained by encoding the character sequence with a bi-directional long short-term memory network (BiLSTM) and concatenate the two last hidden states to form the embedding of the token [44]. This can help to learn meaningful embedding vectors for out-of-vocabulary (OOV) words.

4.2 Vertex Encoding based on Graph Convolutional Network

Unlike the linearly structured query, the document is restructured into a keyword graph. After we embedded the vertices by word vectors, we utilize the ability of Graph Convolutional Network (GCN) [1] to capture the interactions between vertices and get the contextual representation for each vertex. Now let us briefly describe the GCN propagation layers in our model, which are used to encode graph vertices with contextual information and revise the vertex vector representation iteratively.

4.2.1 Graph Convolutional Network for Weighted Graphs

Graph Convolutional Networks (GCN) generalize traditional CNN from low-dimensional regular grids to high-dimensional irregular graph domains. In our work, we improve the graph convolutional network (GCN) proposed in [1] to better deal with weighted graphs, and learn multiresolution vertex representations through multi-layer graph convolutions. In this way, we can match query and document keywords in different semantic levels and enhance the performance of relevance matching.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected weighted graph consisting of a set of vertices $\mathcal{V}$ with $|\mathcal{V}| = N$ and a set of edges $\mathcal{E}$. To clearly depict the vertex-connection of a

graph, the weighted adjacency matrix $A \in \mathbb{R}^{N\times N}$ is introduced, where A_{ij} indicates the weight between vertex $\mathcal{V}_i$ and $\mathcal{V}_j$. The diagonal degree matrix of A is denoted by $D \in \mathbb{R}^{N\times N}$ with $D_{ii} = \sum_j A_{ij}$.

Graph Laplacian is the fundamental operator in the spectral graph analysis, which is formally defined as

$$L = D - A \in \mathbb{R}^{N\times N} \tag{4.1}$$

In addition, there are two normalized versions of the Graph Laplacian, known as Symmetric Laplacian L_{sys} and Random Walk Laplacian L_{rw}, which are respectively denoted as

$$L_{sys} = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{4.2}$$

$$L_{rw} = I_n - D^{-1} A \tag{4.3}$$

Since the graph $\mathcal{G}$ is undirected and weighted, L is a symmetric positive semidefinite matrix, which can be decomposed to

$$L = U \Lambda U^T \tag{4.4}$$

with a diagonal matrix of eigenvalues $\lambda = diag([\lambda_0, \lambda_1, \cdots, \lambda_{N-1}])$ and a matrix of eigenvectors $U = [u_0, u_1, \cdots, u_{N-1}]$.

Let us consider the graph convolution in the Fourier domain. As mentioned in [1], the spectral convolution can be generalized as the Hadamard production of the graph signal and spectral filter in the Fourier domain. Thus, we have the convolution result y defined as:

$$y = U g_\theta(\Lambda) U^T x \tag{4.5}$$

where $x \in \mathbb{R}^N$ is the graph signal with scalar feature for each vertex. Spectral filter $g_\theta(\Lambda)$ is a function of eigenvalues of L parameterized by $\theta \in \mathbb{R}^N$. Note that

$\widetilde{x} = U^T x$ represents the Fourier transform (FT) of the signal x, while $U\widetilde{x}$ is the inverse FT. However, the convolution in Eq. 4.5 requires explicitly computation of Laplacian eigenvectors, which is not feasible especially for large graphs. To solve this problem, Chebyshev polynomials are implemented to approximate the filter $g_\theta(\Lambda)$ as the K-localized filter $g_\theta^K(\Lambda)$:

$$g_\theta(\Lambda) \approx g_\theta^K(\Lambda) = \sum_{k=0}^{K} \theta_k T_k(\widetilde{\Lambda}) \tag{4.6}$$

where $\widetilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_N$ is a diagonal matrix with scaled eigenvalues in the range $[-1, 1]$. $\theta = [\theta_0, \theta_1, \cdots, \theta_K]$ is a vector of Chebyshev coefficients, and $T_k(\widetilde{\Lambda})$ is the k-th order Chebyshev polynomial evaluated at $\widetilde{\Lambda}$. By the approximation of the filter, Eq. 4.5 can be estimated as the K-th localized convolution:

$$y \approx \sum_{k=0}^{K} \theta_k T_k(\widetilde{L}) x \tag{4.7}$$

where $\widetilde{L} = \frac{2}{\lambda_{max}} L - I_N$. Recall that Chebyshev polynomials $T_k(\widetilde{L})$ can be derived from a recurrence relation $T_k(\widetilde{L}) = 2xT_{k-1}(\widetilde{L}) - T_{k-2}(\widetilde{L})$ with $T_0(\widetilde{L}) = 1$ and $T_1(\widetilde{L}) = \widetilde{L}$. In this way, the computation complexity is reduced to $\mathcal{O}(K\,|\mathcal{E}|)$.

Rather than working on all vertices, the K-th localized convolution only focus on the K-hop neighborhoods from the central vertex. Let $K = 1$ and $\lambda_{max} = 2$, the above model is simplified as:

$$y = \theta_0 x + \theta_1 (L - I_N) x \tag{4.8}$$

Properly reduce the number of parameters not only to accelerate computations, but also avoid overfitting in the training process. Unlike parameter settings in [1] with $\theta_0 = -\theta_1$, we constrain the parameters to $\theta_0 = -\lambda\theta_1$. Denote θ_1 by θ, we have:

$$y = \theta((\lambda + 1) I_N - L) x \tag{4.9}$$

Let $X = [x_1, x_2, \cdots, x_N] \in \mathbb{R}^{N \times d_e}$ denotes the vertex feature matrix with

each $x_i \in \mathbb{R}^{d_e}$ representing a d_e-dimensional feature vector of vertex $\mathcal{V}_i$. When $L = L_{rw} = I_N - D^{-1}A$, the graph convolutional layer can be expressed as:

$$X^{n+1} = \sigma(\widetilde{D}^{-1}(A + \lambda I_N)X^n W^n) \tag{4.10}$$

where $\widetilde{D}_{ii} = \lambda + \sum_j A_{ij}$, and σ is the active function in each layer such as ReLU.

The parameter λ controls the balance between the central vertex and its neighbor vertices. With larger λ, the central vertex will involve more in the convolutional operation. If λ equals to zero, the central vertex will have no contribution to its vertex convolution result.

The convolutional layer of Eq. 4.10 is essentially a generalization of the graph convolutional layer in [1][45] with $\lambda = 1$. When Graph Laplacian $L_{sys} = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, the convolution layer becomes the GCN in [1]. However, when $L_{rw} = I_n - D^{-1}A$, it is exactly the same with graph convolutional layer in DGCNN [45]. Obviously, with the introduced parameter λ, the graph convolutional layer of Eq. 4.10 can better deal with weighted graph for different scaler of weights. For example, if the edge weights are all larger than a hundred, let $\lambda = 1$ just like it is in GCN and DGCNN, the central vertex will almost have no influence on its convolution results.

4.2.2 Vertex Embedding

Graph Convolutional Networks learn representations of vertices considering both the graph structure and feature description of vertices.

Here, we will compare the operation of graph convolutional with the 1-dimensional Weisfeiler-Lehman (WL) [46], which is a classic algorithm providing unique assignment of vertex labels in a graph. As demonstrated in Algorithm 4.1, x_i^t denotes the coloring (label assignment) of vertex v_i at iteration t and $\mathcal{N}_i$ is its set of neighboring vertices. At each iteration t, the color of each vertex is updated to reflect its previous color together with the multiset of colors of its neighbors. This proceeds iteratively until a stable coloring. Replace the $hash$ function in Algorithm 4.1 with

Algorithm 4.1 1-dim Weisfeiler-Lehman Algorithm

Initial $t := 0$, vertex coloring $[x_1^0, x_2^0, \cdots, x_N^0]$

for $t = 0$ to T **do**
  for $v_i \in \mathcal{V}$ **do**
    $x_i^{t+1} = hash(x_i^t + \sum_{j \in \mathcal{N}_i} x_j^t)$;
    if stable vertex coloring is reached **then**
      $break$;
Output $[x_1^t, x_2^t, \cdots, x_N^t]$

the following equation

$$x_i^{t+1} = \sigma((\lambda x_j^t + \sum_{j \in \mathcal{N}_i} \frac{1}{a_{ij}} x_j^t) W^t) \tag{4.11}$$

where a_{ij} is normalization constant for the edge between v_i and v_j. We recover the graph convolution rules in Eq. 4.10.

Since a graph convolutional layer can be viewed as differentiable and parameterized generalization of the 1-dimensional Weisfeiler-Lehman (WL) [46] algorithm on graphs [1], for our keyword graph, the convolution process can be interpreted as iteratively revising the representations of vertices based on their neighboring vertices. In this way, the contextual information of each vertex in the document is incorporated. With the increasing layers of graph convolution, each vertex will incorporate the information of a broader context (neighbors with a larger distance to it will be considered in the vertex encoding). Therefore, the multi-layer graph convolution gives multiresolution representations of the vertices.

Fig 4.2 shows how the vertex is embedded through layers of graph convolution. At the beginning, each vertex is embedded with its own feature values X_i. After going through the first graph convolution layer, each vertex Y_i is embedded with the combination of its own feature X_i and the features of its 1-hop neighboring vertices X_j. When we apply the second graph convolution on Y, the vertex feature Z_i including the features of its 2-hop neighboring vertices.

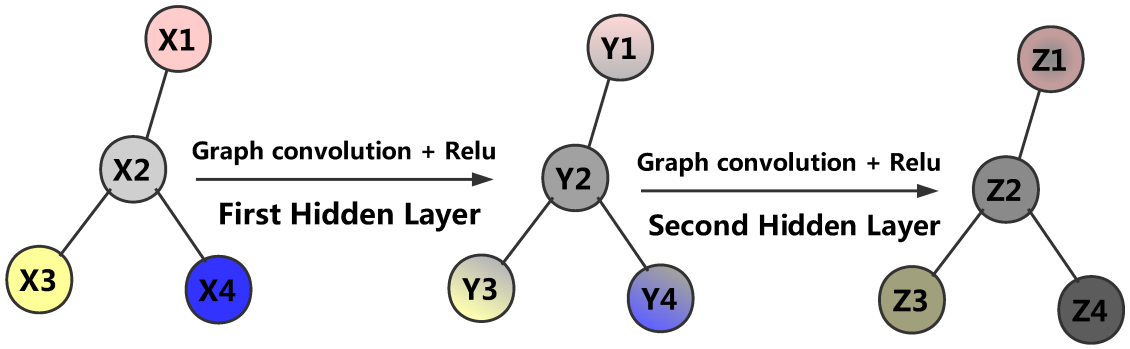

Fig. 4.2. The impact of convolution on each vertex.

4.3 Rank-and-Pooling Layer

After encoding graph vertices through a multi-layer GCN, we propose a Rank-and-Pooling mechanism to sort and select the vertices for later processing. To be specific, let Z denotes the d_e-dimensional vector representations of the d_g vertices in the last graph convolution layer, that is

$$Z = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1d_e} \\ z_{21} & z_{22} & \cdots & z_{2d_e} \\ \vdots & \vdots & \vdots & \vdots \\ z_{d_g1} & z_{d_g2} & \cdots & z_{d_gd_e} \end{bmatrix} \tag{4.12}$$

For each dimension of the vertex features, we normalize it by calculating the softmax over all d_g vertices and then sum up the feature values of all dimensions, so that we can get $\hat{Z} = [\hat{z}_1, \hat{z}_2, \cdots, \hat{z}_{d_g}]^T$, where $\hat{z}_i$ is defined as

$$\hat{z}_i = \sum_{j=1}^{d_e} \frac{e^{z_{ij}}}{\sum_{i=1}^{d_g} e^{z_{ij}}} \tag{4.13}$$

According to the sum, d_g vertices are sorted. We then select the top K vertices with largest $\hat{z}_i$ for further processing.

The Rank-and-Pooling operation is designed for two purposes. First, as there is no order for the vertices in a graph, we use the ranking mechanism to sort the vertices and find out the vertices with significant feature values in the last graph convolution layer. As the feature values of different dimensions may have different scale, we apply softmax to normalize the feature values of each dimension. Second, the number of keywords d_g (or vertices) varies for different documents. We apply the "max-pooling" operation to select the top K vertices from each layer. In this way, we can focus on significant keywords for relevance matching, and also control the dimension of the final matching vector.

4.4 Attention-based Query-Graph Matching

Based on the above sorted K vertices, we apply an attention matching scheme between the query and selected vertices in each layer. Given the encoded query matrix $Q \in \mathbb{R}^{d_e \times d_q}$, where d_e is the encoding dimension and d_q is the number of tokens in the query. Suppose $\mathbf{v}_j$ is the j-th keyword vertex vector in the graph. For each vertex j, we calculate a vertex-aware query representation $\mathbf{q}_j$ as:

$$\mathbf{q}_j = \text{Attention}(Q, \mathbf{v}_j) = Q \cdot \text{softmax}(Q^T \mathbf{v}_j), \;\; 1 \leq j \leq K. \tag{4.14}$$

After we get $\mathbf{q}_j$ for each vertex j, we then calculate the match score between query and vertex as

$$s_{lj} = \text{CosineSim}(\mathbf{v}_j, \mathbf{q}_j), \tag{4.15}$$

where s_{lj} denotes the match score between query and vertex j in layer l.

This layer helps each vertex to focus on the matching signals with a part of the query tokens that are most related to that vertex. If only a small portion of the tokens in the query are correlated to a specific keyword vertex, our attention based query-vertex matching will help to decrease the influence of uncorrelated tokens.

4.5 Aggregation Layer

In this layer, we concatenate the matching scores of each vertex in each graph convolution layer, as well as the OOV feature x_{oov} described above, to form a final matching vector $\mathbf{m}$ as following:

$$m = [s_{11}, s_{12}, \cdots, s_{1K}, \cdots, s_{lk}, \cdots, s_{L1}, s_{L2}, \cdots, s_{LK}, x_{oov}], \tag{4.16}$$

where s_{lk} is the attention matching score between query and k_{th} vertex in l_{th} layer. Apparently, $m \in \mathbb{R}^{(KL+1)\times 1}$ with L denotes the number of graph convolution layers.

We then feed the concatenated matching vector $\mathbf{m}$ into an aggregation layer to get the final relevance matching result. In our experiment, a one-layer feed forward neural network was implemented with the hidden size set to 100.

Chapter 5

Experiments

In this chapter, our proposed Multiresolution Graph Attention Network is evaluated on two datasets and compared with other existing deep matching models, including both representation-focused deep neural matching models and interaction-focused models. Then, we further execute an ablation study by removing different parts of our model and evaluating the performance of the model variants. The ablation study proves that each module in our model plays a significant role in the task of relevance matching.

5.1 Description of Tasks and Datasets

Two datasets are evaluated in our experiments, they are described as following

- **Ohsumed dataset for topic document matching [47].** The Ohsumed dataset consists of 34394 documents from medical abstracts and are classified into

TABLE 5.1
DESCRIPTION OF EVALUATION DATASETS.

Dataset	Pos	Neg	Train	Dev	Test
Ohsumed	56976	56976	68370	22789	22793
NFCorpus	64467	35465	59959	19986	19987

TABLE 5.2
AVERAGE LENGTH OF TEXT IN EVALUATION DATASETS.

Dataset	Query	Document
Ohsumed	2.6	109.4
NFCorpus	3.5	146.7

23 categories of cardiovascular disease groups. The dataset is originally for the document topic classification. In our experiment, we generate topic-document pair samples from the original dataset, where a positive sample means the topic is the true category of the document. A negative topic-document sample is generated by randomly assigning an incorrect topic to a document. The average length of topic text and documents are 2.6 and 109.4, respectively.

- **NFCorpus dataset for medical information retrieval.** The NFCorpus dataset is a full-text English retrieval dataset for the task of Medical Information Retrieval. It contains a total of $3,244$ non-technical English queries that harvested from the NutritionFacts.org site, with $169,756$ automatically extracted relevance judgments for $9,964$ medical documents (written in a complex terminology-heavy language), mostly from PubMed [48]. We selected a subset of the original dataset which contains $64,467$ samples, as the original dataset is extremely unbalanced. The average length of queries and documents are 3.5 and 146.7, respectively.

Table 5.1 shows a detailed breakdown of the datasets used in the evaluation. Table 5.2 demonstrates the average length of query and documents in two datasets. For both of the two datasets, we use 60% of samples as a training set, 20% of samples as a development set, and the remaining 20% as a test set. We use training sets to train the models, development sets to tune the hyper-parameters and each test set is only used once in the final evaluation. We train our model by the Adam optimizer with learning rate set to 0.001. The metrics we used to evaluate the performance of

our proposed models on the text matching tasks are the accuracy and the F1 score of classification results. For each model, we carry out training for 5 epochs. We then choose the model with the best validation performance to be evaluated on the test set.

5.2 Competitor Methods

In the following, We briefly describe the baseline methods:

- **Convolutional Matching Architecture-I (ARC-I)** [2]: ARC-I is a typical representation-focused deep matching model, which encodes each piece of text to a vector by CNN and compares the representing vectors with a multi-layer perceptron (MLP).
- **Convolutional Matching Architecture-II (ARC-II)** [2]: ARC-II is built directly on the interaction space between two texts, and intends to capture the rich matching patterns at different levels with the 2-D convolution.
- **Deep Structured Semantic Models (DSSM)** [13]: DSSM utilizes deep neural networks to map high-dimension sparse features into low-dimensional dense features, and then computes the semantic similarity of the text pair.
- **Convolutional Deep Structured Semantic Models (C-DSSM)** [14]: C-DSSM learns low-dimensional semantic vectors for input text by CNN. Particularly, DSSM and C-DSSM are designed for Web search. However, they were only evaluated on (query, doc title) pairs.
- **Multiple Positional Semantic Matching (MV-LSTM)** [4]: MV-LSTM matches two pieces of text with multiple positional text representations, and aggregates interactions between different positional representations to give a matching score.

TABLE 5.3
ACCURACY AND F1-SCORE RESULTS OF DIFFERENT ALGORITHMS ON THE OHSUMED DATASET.

Algorithm	Dev		Test	
	Accuracy	F1-score	Accuracy	F1-score
ARC-I	0.5067	0.6676	0.5068	0.6681
ARC-II	0.5490	0.6759	0.5511	0.6775
DSSM	0.5243	0.4811	0.5138	0.4721
C-DSSM	0.5155	0.5650	0.5112	0.5613
MatchPyramid	0.5597	0.6597	0.5648	0.6625
MV-LSTM	0.5610	0.4559	0.5555	0.4481
MGAN	**0.8040**	**0.8090**	**0.8075**	**0.8118**

- **MatchPyramid** [5]: MatchPyramid calculates pairwise word matching matrix, and models text matching as image recognition, by taking the matching matrix as an image.

For the above baseline deep matching models, we utilized MatchZoo [16] for evaluation. For our MGAN model, since the edge weights of the graph is in the range of 0 to 1, we set $\lambda = 1$. Besides, considering the average length of documents, K is set to 20 in the Rank-and-Pooling. The number of graph convolution layers L is 2.

5.3 Performance Analysis

Table 5.3 and Table 5.4 compares our model with existing deep matching models on the Ohsumed dataset and the NFCorpus dataset, in terms of classification accuracy and F1 score. Results demonstrate that our Multiresolution Graph Attention Net-

TABLE 5.4
ACCURACY AND F1-SCORE RESULTS OF DIFFERENT ALGORITHMS ON THE NFCORPUS DATASET.

Algorithm	Dev		Test	
	Accuracy	F1-score	Accuracy	F1-score
ARC-I	0.5067	0.6676	0.7969	0.8548
ARC-II	0.5490	0.6759	0.7576	0.8361
DSSM	0.5243	0.4811	0.6336	0.7646
C-DSSM	0.5155	0.5650	0.6259	0.7590
MatchPyramid	0.5597	0.6597	0.6408	0.7811
MV-LSTM	0.5610	0.4559	0.6683	0.7564
MGAN	**0.9425**	**0.9553**	**0.9407**	**0.9535**

work achieves the best classification accuracy and F1 score on both two datasets.

This can be attributed to multiple characteristics of our model. First, the input to our neural network model is the keyword graph representation of documents, rather than the original sequential word representation. Based on it, we characterize the interaction patterns between different keywords of the document. This helps to incorporate the semantic structure information of a long document into our model, and alleviates the problem of long-distance dependency (as correlated words are connected by edges directly). Our model solves the problem of matching query and document in a "divide-and-conquer" manner to cope with the long length of documents: it matches the query with each keyword of the document to get matching signals, and aggregate all the matching signals by utilizing the correlations between keywords to give an overall relevance matching result. Second, our model learns a multiresolution encoding representation for each keyword vertex via a multi-layer Graph Convolutional Network. In each graph convolution layer, the representations

of vertices are revised by taking their neighboring vertices into account. In this way, the context information of the keywords in the document is encoded into the high-level vertex representations. Third, for each vertex in each graph convolution layer, we learn a vertex-specific query representation through attention mechanism to match the query with each vertex. This operation helps the vertices to focus on the query information that is related to it. Finally, our rank-and-pooling operation unifies the number of vertices for different documents, and selects the most important matching signals in each layer to get the final matching result.

On the contrary, Table 5.3 and Table 5.4 also indicate that the baseline deep text matching models lead to bad performance in query document relevance matching tasks. The main reasons are the following. First, existing deep text matching models are more suitable for the task of semantic matching, where the main concerns in such tasks are the compositional meanings of text pieces and the global matching between them. In our case, matching query and document is the problem of relevance matching. This problem emphasizes more on the exact matching signals between query keywords and documents. Both the importance of different query keywords and the topic structure of documents are critical to relevance matching, and we need to take them into account. Second, existing deep text matching models can hardly capture meaningful semantic relations between a short query and a long document. When the document is long, it may covers multiple topics, and the query may match only a part of the document. In this case, it is hard to get an appropriate context vector representation for relevance matching, and the part of document that is not related with the query will overwhelm the match signals of the related part. For interaction-focused models, most of the interactions between words in the query and the document will be meaningless, therefore it is not easy to extract useful interaction features for further matching steps. Our model effectively solves the above challenges by representing documents as keyword graphs, and utilize the semantic structure of long documents through Graph Convolution Network for relevance matching.

Moreover, we also tried to represent a document by TF-IDF vector based on

TABLE 5.5
ACCURACY AND F1-SCORE RESULTS OF MGAN AND ITS VARIANTS ON THE OHSUMED DATASET.

Algorithm	Dev		Test	
	Accuracy	F1-score	Accuracy	F1-score
No GCN	0.6837	0.6819	0.6850	0.6810
No Attention	0.7908	0.7882	0.7893	0.7865
No Query Encoding	0.7859	0.7900	0.7927	0.79576
Pooling Size $K = 5$	0.7602	0.7453	0.7642	0.7484
MGAN	**0.8040**	**0.8090**	**0.8075**	**0.8118**

its words or its keywords. Then we calculate the cosine similarity to estimate the relevance level between a query and a document. We found that the performance given by such shallow models are quite bad: the accuracy is around 0.38 and F1 is smaller than 0.1. This proves the necessity of representing words by word vectors, and incorporating document structural information by graph convolution.

In overall, the experimental results demonstrate the superior applicability and generalizability of our proposed model.

5.4 Impact of Different Modules and Parameters

We also tested several model variants for ablation study. For each model variant, we remove one module from the complete Multiresolution Graph Attention Network model, and compare its performance with our complete model on the two datasets to evaluate the impact of the removed component.

Table 5.5 and Table 5.6 show the performance of all the evaluated models for ablation study on two datasets. Specifically, we evaluated the following models:

- **Multiresolution Graph Attention Network (MGAN)**. This is our original

TABLE 5.6
ACCURACY AND F1-SCORE RESULTS OF MGAN AND ITS VARIANTS ON THE NFCORPUS DATASET.

Algorithm	Dev		Test	
	Accuracy	F1-score	Accuracy	F1-score
No GCN	0.8767	0.9053	0.8757	0.9039
No Attention	**0.9432**	**0.9558**	**0.9433**	**0.9556**
No Query Encoding	0.8616	0.8929	0.8629	0.8930
Pooling Size $K = 5$	0.9381	0.9520	0.9381	0.9517
MGAN	0.9425	0.9553	0.9407	0.9535

proposed model.

- **MGAN (no GCN)**. It is a variant model that removes the graph convolutional layers in our neural network. In other words, we just represent each vertex by the word vector of the keyword it contains, and match each keyword with all the query words.

- **MGAN (no attention)**. This variant model removes the attention mechanism in our neural network. In this model, we add a max-pooling layer over the encoded query words to get the hidden vector representation of the query, and use it to match with each vertex.

- **MGAN (less keywords)**. In this model, we reduce the number of selected keywords by setting $K = 5$ instead of 20.

- **MGAN (no query encoding)**. In this model, we remove the 1D CNN encoder for query, and directly use the word vectors to represent each query token.

Impact of graph convolution layers. Compare our model with the version that

do not contain any graph convolution layers. We can see that the performance is worse on both datasets when we remove graph convolution from our model. The reason is that the representation of each vertex will be local and does not contain any context information of its neighboring vertices. Therefore, the topological structure of keyword interactions in the document are totally ignored. In our model with graph convolution layers, in each layer, we lean an adaptive context vector for each vertex. It incorporates the semantic meaning of its neighboring keywords based on their vector representations and edge weights. The multi-layer graph convolution leads to a multiresolution semantic representation of keywords in the document, as in a higher layer, the representation of a vertex covers the information of vertices in a broader range.

Impact of query encoding. Compare our model with the version that do not perform query encoding. When the query tokens are only represented by the original word vectors and not refined by any encoders to incorporate the contextual information, the performance becomes worse. For example, if the main focus of the query is a key phrase that contains multiple tokens, the CNN encoder can help to combine the semantic information in tokens to represent the key phrase, while the original sequence of word vectors can hardly capture the compositional meaning.

Impact of query-vertex attention. Compare our model with the version that do not implement query-vertex attention. In this case, our model gets better performance on the Ohsumed dataset and comparable performance on the NFCorpus dataset. Our model use the attention mechanism to learn a vertex-aware query encoding for each vertex. Thus, each vertex will focus on the matching signals with a subset of the query tokens. In comparison, when we remove the attention mechanism from our model, each vertex will match with the same encoding vector of the query. Given a specific vertex, the unrelated tokens in the query make the matching signal between a query and a keyword less important. However, when tokens in the query have similar meaning, the attention mechanism won't have significant impact on the performance of our model.

Impact of the number of selected keywords in the Rank-and-Pooling. In the

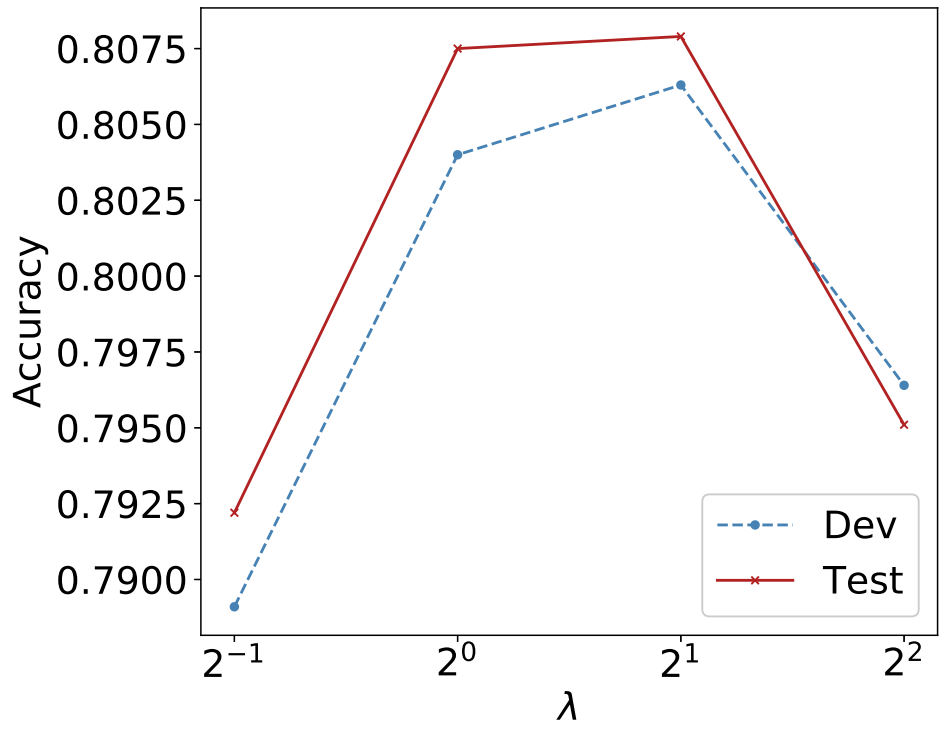

Fig. 5.1. Compare the accuracies given by different λ on the Ohsumed dataset.

Rank-and-Pooling operation, we need to set a parameter K and choose the matching results between the query and the top K vertices for each graph convolution layer. We tested $K = 5$ and $K = 20$ respectively, and the performance is better when $K = 20$. That is reasonable, as $K = 20$, our keyword graphs of documents retain more information of the original documents. If the value of K is small, keywords related to the query are more likely to be removed. However, if the value of K is too large, the unimportant words in the document will become noise to the matching model thus leading to bad performance. Furthermore, we should also take the time complexity of the model into account. More vertices selected in each layer, the more time we need for computation.

Impact of parameter λ. We tested the performance of our MGAN model on the Ohsumed dataset with different values of λ. Fig. 5.1 and Fig. 5.2 show the comparison result in terms of accuracy and F1 score. It shows that the performance of our model achieve the best when λ is set to be around 1. If λ is too small or

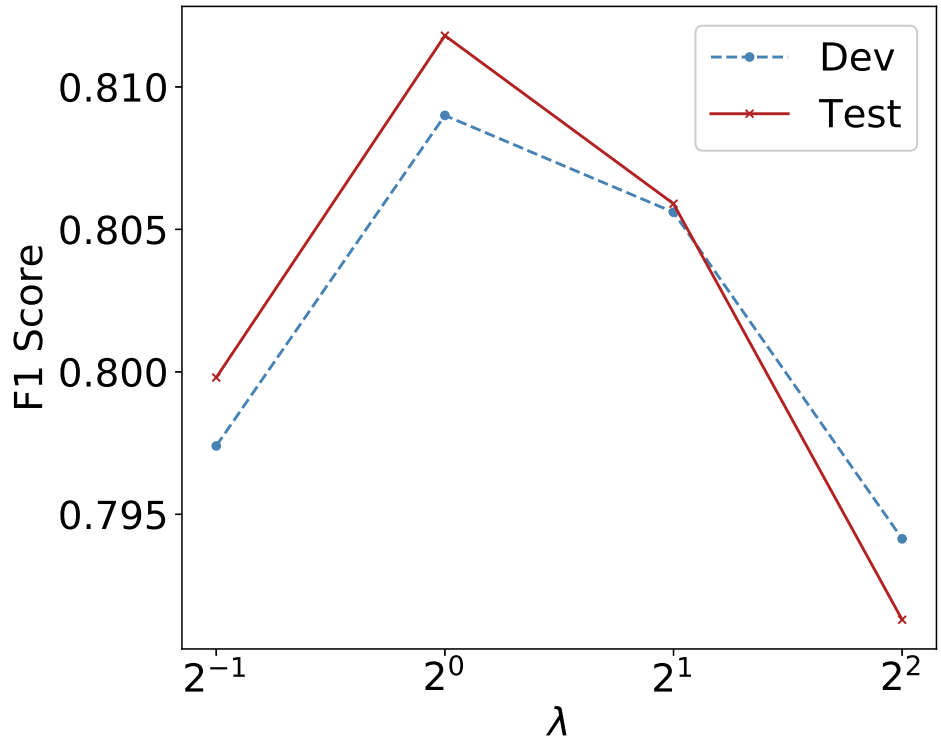

Fig. 5.2. Compare the F1 scores given by different λ on the Ohsumed dataset.

too large, the accuracy and F1 score will decrease. The reason is that the value of λ shall be around the same scale with the edge weights in the keyword graph. In our experiments, the edges weights are within the range of 0 to 1. Large λ means that we focus more on each vertex's own information and incorporate little contextual information into it by graph convolution. In contrast, a small value of λ makes the graph convolution emphasize on incorporating the contextual information of vertex's neighboring vertices, but the vertex's own information plays a less important role. Therefore, λ is significant to the weighted graphs and should set to an appropriate scale.

Chapter 6

Conclusion

In this thesis, we point out the key role of semantic structures of documents in the task of relevance matching between *short-long* text pairs, and show that most existing approaches cannot achieve satisfactory performance for this task. We propose to model a long document as a weighted undirected graph of keywords, with each vertex representing a keyword in the document, and edges indicating their interaction levels. Based on the graph representation of documents, we further propose the *Multiresolution Graph Attention Network* (MGAN), a novel deep neural network architecture, which learns multi-layer representations for keyword vertices through a Graph Convolutional Network. It models the local interactions between query words and each document keyword based on attention mechanism, and combines the multiresolution matching between query and keywords on different graph convolution layers with a *rank-and-pooling* procedure to give the final relevance estimation result.

We apply our techniques to the task of relevance matching based on the Ohsumed dataset and the NFCorpus dataset. The simulation results show that the proposed approach can achieve significant improvement for relevance matching in terms of accuracy and F1 score, compared with multiple existing approaches.

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[2] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *Advances in neural information processing systems*, 2014, pp. 2042–2050.

[3] X. Qiu and X. Huang, "Convolutional neural tensor network architecture for community-based question answering." in *IJCAI*, 2015, pp. 1305–1311.

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

[5] L. Pang, Y. Lan, J. Guo, J. Xu, S. Wan, and X. Cheng, "Text matching as image recognition." in *AAAI*, 2016, pp. 2793–2799.

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

[7] W. Yin and H. Schütze, "Convolutional neural network for paraphrase identification," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 901–911.

[8] Z. Ji, Z. Lu, and H. Li, "An information retrieval approach to short text conversation," *arXiv preprint arXiv:1408.6988*, 2014.

[9] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From word embeddings to document distances," in *International Conference on Machine Learning*, 2015, pp. 957–966.

[10] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang, "Irgan: A minimax game for unifying generative and discriminative information retrieval models," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2017, pp. 515–524.

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

[12] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[13] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. 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*. ACM, 2013, pp. 2333–2338.

[14] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "Learning semantic representations using convolutional neural networks for web search," in *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 2014, pp. 373–374.

[15] J. Guo, Y. Fan, Q. Ai, and W. B. Croft, "A deep relevance matching model for ad-hoc retrieval," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016, pp. 55–64.

[16] Y. Fan, L. Pang, J. Hou, J. Guo, Y. Lan, and X. Cheng, "Matchzoo: A toolkit for deep text matching," *arXiv preprint arXiv:1707.07270*, 2017.

[17] R. Krovetz and W. B. Croft, "Lexical ambiguity and information retrieval," *ACM Transactions on Information Systems (TOIS)*, vol. 10, no. 2, pp. 115–141, 1992.

[18] D. D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *European conference on machine learning*. Springer, 1998, pp. 4–15.

[19] S. Robertson, H. Zaragoza *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[20] H. Li and J. Xu, "Beyond bag-of-words: machine learning for query-document matching in web search," in *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2012, pp. 1177–1177.

[21] B. Rosario, "Latent semantic indexing: An overview," *Techn. rep. INFOSYS*, vol. 240, pp. 1–16, 2000.

[22] P. Neculoiu, M. Versteegh, M. Rotaru, and T. B. Amsterdam, "Learning text similarity with siamese recurrent networks," *ACL 2016*, p. 148, 2016.

[23] Y. Shao, "Hcti at semeval-2017 task 1: Use convolutional neural network to evaluate semantic textual similarity," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017, pp. 130–133.

[24] Z. Lu and H. Li, "A deep architecture for matching short texts," in *Advances in Neural Information Processing Systems*, 2013, pp. 1367–1375.

[25] J. Leskovec, M. Grobelnik, and N. Milic-Frayling, "Learning sub-structures of document semantic graphs for document summarization," 2004.

[26] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[27] F. Rousseau and M. Vazirgiannis, "Graph-of-word and tw-idf: new approach to ad hoc ir," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 2013, pp. 59–68.

[28] F. Rousseau, E. Kiagias, and M. Vazirgiannis, "Text categorization as a graph classification problem." in *ACL (1)*, 2015, pp. 1702–1712.

[29] Y. Wang, X. Ni, J.-T. Sun, Y. Tong, and Z. Chen, "Representing document as dependency graph for document clustering," in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 2177–2180.

[30] A. Schenker, M. Last, H. Bunke, and A. Kandel, "Clustering of web documents using a graph model," *SERIES IN MACHINE PERCEPTION AND ARTIFICIAL INTELLIGENCE*, vol. 55, pp. 3–18, 2003.

[31] H. Balinsky, A. Balinsky, and S. Simske, "Document sentences as a small world," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2583–2588.

[32] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

[33] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–479, 2004.

[34] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[35] J. W. G. Putra and T. Tokunaga, "Evaluating text coherence based on semantic similarity graph," in *Proceedings of TextGraphs-11: the Workshop on Graph-based Methods for Natural Language Processing*, 2017, pp. 76–85.

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

[37] K. K. Schuler, "Verbnet: A broad-coverage, comprehensive verb lexicon," 2005.

[38] S. Hensman, "Construction of conceptual graph representation of texts," in *Proceedings of the Student Research Workshop at HLT-NAACL 2004*. Association for Computational Linguistics, 2004, pp. 49–54.

[39] B. Rink, C. A. Bejan, and S. M. Harabagiu, "Learning textual graph patterns to detect causal event relations." in *FLAIRS Conference*, 2010.

[40] C. Jiang, F. Coenen, R. Sanderson, and M. Zito, "Text classification using graph mining-based feature extraction," *Knowledge-Based Systems*, vol. 23, no. 4, pp. 302–308, 2010.

[41] C. Baker and M. Ellsworth, "Graph methods for multilingual framenets," in *Proceedings of TextGraphs-11: the Workshop on Graph-based Methods for Natural Language Processing*, 2017, pp. 45–50.

[42] S. Siddiqi and A. Sharan, "Keyword and keyphrase extraction techniques: a literature review," *International Journal of Computer Applications*, vol. 109, no. 2, 2015.

[43] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[44] M. Hu, Y. Peng, and X. Qiu, "Mnemonic reader for machine comprehension," *arXiv preprint arXiv:1705.02798*, 2017.

[45] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," 2018.

[46] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[47] W. Hersh, C. Buckley, T. Leone, and D. Hickam, "Ohsumed: an interactive retrieval evaluation and new large test collection for research," in *SIGIR'94*. Springer, 1994, pp. 192–201.

[48] V. Boteva, D. Gholipour, A. Sokolov, and S. Riezler, "A full-text learning to rank dataset for medical information retrieval," in *European Conference on Information Retrieval*. Springer, 2016, pp. 716–722.