End-to-end Learning for Mining Text and Network Data

by

Cheng Li

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Information)
in the University of Michigan
2017

Doctoral Committee:

Associate Professor Qiaozhu Mei, Chair
Assistant Professor Jia Deng
Professor Paul J. Resnick
Assistant Professor Daniel M. Romero

Cheng Li

lichengz@umich.edu

ORCID iD: 0000-0003-0678-1357

To my parents and my husband Xiaoxiao

ACKNOWLEDGEMENTS

My deepest gratitude and appreciation goes to all those who have helped me reach this important milestone: my advisor, Dr. Qiaozhu Mei, who provides excellent guidance and sustained support during my PhD life; other members of my committee, Dr. Jia Deng, Dr. Paul Resnick, and Dr. Daniel Romero, who offer valuable suggestions for my doctoral work; my fellow doctoral students and friends at Michigan and elsewhere; the supportive staff of the University of Michigan School of Information; my mentor Dr. Yue Lu at Twitter, who introduces to me the world of industry; and my mentor Dr. Michael Bendersky, who shows me how research is conducted at Google.

Finally, I would like to thank my husband Dr. Xiaoxiao Guo for always being there for me, providing strong support both for my life and my research; and my parents for giving birth to me, providing steadfast support and unconditional love ever since then.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

ABSTRACT

A wealth of literature studies user behaviors in online communities, e.g., how users respond to information that are spreading over social networks. One way to study user responses is to analyze user-generated text, by identifying attitude towards target topics. Another way is to analyze the information diffusion networks over involved users. Conventional methods require manual encoding of world knowledge, which is ineffective in many cases. Therefore, to push research forward, we design end-to-end deep learning algorithms that learn high-level representations directly from data and optimize for particular tasks, relieving humans from hard coding features or rules, while achieving better performance. Specifically, I study attitude identification in the text mining domain, and important prediction tasks in the network domain. The key roles of text and networks in understanding user behaviors in online communities are not the only reason that we study them together. Compared with other types of data (e.g., image and speech), text and networks are both discrete and thus may share similar challenges and solutions.

Attitude identification is conventionally decomposed into two separate subtasks: target detection that identifies whether a given target is mentioned in the text, and polarity classification that classifies the exact sentiment polarity. However, this decomposition fails to capture interactions between subtasks. To remedy the issue, we developed an end-to-end deep learning architecture, with the two subtasks interleaved by a memory network. Moreover, as the learned representations may share the same semantics for some targets, but vary for others, our model also incorporates the

interactions among entities.

For information networks, we aim to learn the representation of network structures in order to solve many valuable prediction tasks in the network community. An example of prediction tasks is network growth prediction, which assists decision makers in optimizing strategies. Instead of handcrafting features that could lead to severe loss of structural information, we propose to learn graph representations through a deep end-to-end prediction model. By finding "signatures" for graphs, we convert graphs into matrices, where convolutional neural networks could be applied.

In additional to topology, information networks are often associated with different sources of information. We specifically consider the task of cascade prediction, where global context, text content on both nodes, and diffusion graphs play important roles for prediction. Conventional methods require manual specification of the interactions among different information sources, which is easy to miss key information. We present a novel, end-to-end deep learning architecture named *DeepCas*, which first represents a cascade graph as a set of cascade paths that are sampled through random walks. Such a representation not only allows incorporation of the global context, but also bounds the loss of structural information.

After modeling the information of global context, we equip DeepCas with the ability to jointly model text and network in a unified framework. We present a gating mechanism to dynamically fuse the structural and textual representations of nodes based on their respective properties. To incorporate the text information associated with both diffusion items and nodes, attention mechanisms are employed over node text based on their interactions with item text.

CHAPTER I

Introduction

In today's online communities and social media, text and networks have become the most important data sources for researchers to study users and understand a variety of phenomena. Facebook users post daily life events and make new friends by connecting to other users. Every day, a large number of tweets are retweeted on Twitter, discussing popular products like the latest iPhone and political issues like elections. Researchers make innovations based on findings from other people, citing their work when publishing papers. Modeling text and networks for various tasks have become increasingly important for people from different fields, including social scientists, online marketers, government officers, scientific researchers, and daily users. Studying user behavior is an important research direction in online communities and social media. Researchers have studied for years, both from text and networks, as to how users respond to different kinds of topics that are spreading over social networks, such as rumors, political issues, and commercial products. From the text perspective, text documents, such as user posts and tweets, are analyzed by performing techniques like sentiment and attitude analysis towards topics of interest. In this way, people's stances could be understood from both the individual level and population level if aggregated. Another perspective is to analyze the network composed of users who are involved in this topic, which exhibits the development and signals the future

popularity of the topics to be studied. Therefore, to push research forward in this direction, I study attitude identification in the text mining domain, and valuable prediction tasks in the network domain.

On the other hand, compared with data types that are continuous (e.g., vision, speech, and time series), text and networks are both discrete. This discrete property leads to potentially shared challenges between the two data types. Thus, technologies developed for text might inspire innovations in the field of network mining, which also holds the other way round. This commonality from another perspective motivates the study of text and network mining tasks at the same time.

Despite the importance of tasks for mining text and network data, conventional methods sometimes fall short of resolving the challenges rooted in these tasks. For example, in network mining tasks, various handcrafted features are developed to characterize the network structure, which are then fed to a classic machine-learning algorithm, like logistic regression. The performance of logistic regression heavily depends on features designed by experts, who find it hard to encode every piece of necessary knowledge for a particular task. As another example, in attitude identification, we usually want to track the attitude towards a set of entities. Traditional methods train a separate model for each individual entity, failing to consider the interactions among entities and between the different subtasks. Consequently, information in different components is not shared, leading to inefficiency in learning. An example to show the necessity of considering subtask interaction is as follows. The positive sentiment in "*the new Keynote is user friendly*" provides good evidence that "*Keynote*" is a software (the target) instead of a speech (not the target).

To resolve these issues, I focus my dissertation research on designing end-to-end deep learning algorithms for text and information network mining. Deep learning has emerged as a technique that allows computer programs to learn from data and experience by using deeply layered, hierarchical concepts, with complicated concepts built

upon simpler ones. By directly gathering knowledge from raw data with its ability to accommodate data at large scale, and automatically learning the nonlinear mapping from input to output in an end-to-end manner, this technique relieves humans from the burden of hard coding world knowledge by, e.g., designing features or rules. With a carefully designed multi-layer neuron network, learning errors backpropagate from upper layers to lower layers and from subtasks to subtasks, which enables deep interactions between the learning of multi-grained representations of the data and multiple subtasks, solving tasks that are hard for humans to manually design features, and alleviate the burdens to provide manual annotation for each subtask. The end-to-end nature of deep learning brings in new possibilities of resolving the issues emerged in current approaches to attitude identification and network prediction tasks, with examples mentioned above. Therefore, I focus on developing end-to-end deep learning algorithms for the two tasks, tackling issues that are unable to be fixed by conventional methods.

Attitude identification aims to identify people's attitudes towards a given set of entities. Examples include companies who want to know customers' opinions about their products, governments who are concerned with public reactions about policy changes, and financial analysts who identify daily news that could potentially influence the prices of securities. In a more general case, attitudes towards all entities in a knowledge base may be tracked over time for various in-depth analyses.

In attitude identification, there are conventionally three key components that fail to receive end-to-end treatment. First, the task is decomposed into two separate subtasks. A first model is trained for target detection, which identifies whether an entity is mentioned in the text, either explicitly or implicitly. A second model that is completely independent of the first one is then trained for polarity classification, which classifies the exact sentiment towards an identified entity (the target), usually into three categories: positive, negative, and neutral. This decomposition neglects

intrinsic interactions between the two subtasks. Indeed, features identified in the first subtask – both the words that refer to the target and the positions of these words, could provide useful signals for the polarity of sentiments. On the other hand, sentimental expressions identified in the second subtask and their positions could, in turn, provide feedback to the first task and signal the existence of the target. Secondly, existing methods tend to ignore interactions between targets by training a separate model for each target [73, 42], ignoring that certain targets and their sentiments may share some important semantic dimensions with each other while differing on other dimensions. For example, two targets, *food* and *service*, may share many sentimental expressions, but the sentence "*we have been waiting for food for one hour*" is clearly about the service instead of the food. Lastly, various handcrafted features such as sentiment, syntactic information, and topics are extracted from text [73, 42, 104] to approach the task. Powerful and predictive as these features are, it is hard for humans to enumerate and capture all important pieces of knowledge and their interactions.

To resolve these issues, we propose an end-to-end machine learning architecture, where the two subtasks are interleaved by a deep memory network that directly learns from the raw text input [60]. The proposed model also considers target interactions, by allowing targets to share a common semantic space and simultaneously keep their own space, making it possible for all targets to be learned in a unified model. The proposed deep memory network outperforms models that do not consider the subtask or target interactions, including conventional supervised learning methods and state-of-the-art deep learning models.

A less explored area in deep learning is how to learn a good representation for information networks. In the modern society, it is hard to find an isolated object, and almost everything is connected, forming networks: when we are interacting with our friends on Facebook, all our friends and their connections form a friendship network; when a paper accumulates citations, all papers citing this paper form a citation

network; when an interesting tweet is posted on Twitter and is passed from person to person, all these paths form an information diffusion network. Extracting knowledge from these networks is very valuable for many data mining tasks, with one of them being making many important predictions so that actions could be taken in advance to encourage good outcomes while avoiding bad ones. For example, predicting the popularity of research communities helps scientists to identify promising research directions; predicting the growth of social groups helps social network vendors optimize their marketing strategies; predicting the influence of the diffusion of a rumor helps analysts to estimate its potential damage and apply interventions early when necessary.

Several challenges have to be resolved in order to learn representations of information networks. The first challenge is to learn the representation of the graph structure, or topology that makes up the skeleton of information networks. Figure 1.1 shows some diffusion networks at different time stages. It might be easy to manually encode the diffusion patterns in the beginning. However, as the diffusion expands and affects more users, it becomes more difficult for humans to fully describe the network shape and single out important patterns. In the social network literature, researchers strive to design structural features based on theoretical and empirical findings. For example, open triads with two strong ties are likely to form a closed triangle in the near future [29]; dense communities are resistant to novel information from outside and thus grow slower [40]; nodes spanning structural holes are likely to gain social capital and prestige [14]. Features such as network density, clustering coefficients, triadic profiles, and structural holes are, therefore, designed to implement these intuitions and represent the graph structure.

Despite the informativeness of these handcrafted features, there are some issues. Some features, such as network density, only describe a global property of the network; some of them, such as triads, provide a fine-grained description of local structures but

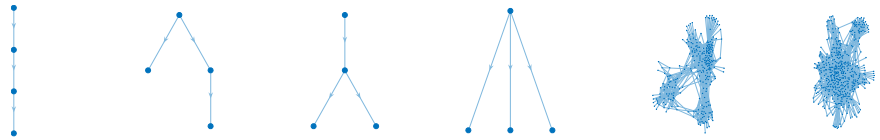


Figure 1.1: Examples of diffusion networks. The first four are the diffusions of Facebook posts at their early stages [19], while the last two are Weibo retweets at later stages.

fail to capture global information. None of these features are able to fully represent both the local and the global structure of a graph and the complex interaction between them. On the other hand, these features usually have a limited characterization power for networks, as many different networks may share the same feature representation.

To remedy these issues, we introduce a graph descriptor that is based on the Heat Kernel Signature (HKS) [100], which serves as a universal low-level representation of the topological structures of networks [59]. HKS has been successfully employed in representing the surface of 3D objects [31, 121]. By modeling the amount of heat flow over the nodes of a network over time, HKS successfully stores both the global and the local structural information of the entire network, and networks with the same topological structure can be mapped to a unique representation of the little loss of structural information. However, unlike 3D objects that are composed of polygon meshes, the structures of networks vary in shape, size, and complex local structures. To address this issue, some computations of HKS need to be approximated carefully. Inspired by the semantics of the HKS-based graph descriptors, we propose a multicolumn, multiresolution neural network that learns latent hierarchical representations of graphs on top of the HKS-based graph descriptor. The proposed deep neural network, named DeepGraph, predicts network growth in an end-to-end process.

We conduct extensive experiments to evaluate the effectiveness of DeepGraph. Different growing properties are predicted for four genres of real-world networks. Empirical results show that our method outperforms baseline approaches that use alternative graph representations, handcrafted features, or existing deep learning ar-

chitectures.

After addressing the challenge of learning graph structures, we then turn to the second challenge of jointly learning different information sources that reside in information networks, including the global network context and text content. Taking a cascade of retweets on Twitter as an example, each cascade only occurs within a subset of Twitter users, forming its cascade network. The global Twitter network, each user's tweets and retweets in the past, and the current tweet being retweeted are all important signals to predict the future popularity of the retweet cascade. To address this challenge, we specifically consider the task of learning a network representation for cascade prediction, whose objective is to predict the future size of a cascade network. Existing studies mostly take a feature-based approach [118, 19, 23, 47]. Many of these features are specific to the particular platform or the specific type of network, and the performance of the resulting algorithm hinges on the researcher's knowledge and familiarity with particular sources of information. For example, whether a photo was posted with a caption is shown to be predictive of how widely it spreads on Facebook [19]; mentioning Twitter users in Tweets is shown to help them gain more retweets [103]. These features are indicative, but cannot be generalized to other platforms or other types of networks. In addition, it is hard for humans to specify the interaction between different sources of information.

Our first step towards learning network representation with consideration of additional information is to take into account the context of the global network structure. We present a novel, end-to-end deep learning architecture named the *DeepCas*, which first represents a cascade graph as a set of cascade paths that are sampled through multiple random walk processes [61]. Such a representation not only takes into account the global context, but also bounds the loss of structural information. Analogically, cascade graphs are represented as documents, with nodes as words and paths as sentences. The challenge is how to sample the paths from a graph to as-

semble the "document," which is also automatically learned through the end-to-end model to optimize the prediction of cascade growth. We evaluate the performance of the proposed method using real world information cascades in two different domains, Tweets, and scientific papers. DeepCas is compared with multiple strong baselines, including feature-based methods, node-embedding methods, and graph kernel methods. DeepCas significantly improves the prediction accuracy over these baselines, which provides interesting implications for the understanding of information cascades.

After modeling global context, the next problem is how to incorporate the rich text information into our cascade prediction model. A diffusion item can be described by text message – tweets, posts, and scientific papers are themselves written in text. On the contrary, users who are propagating these items also have text associated with them. For example, Twitter users have a history of tweets and retweets, while researchers have a list of publications.

Text greatly complements structural information, especially when node members of cascades rarely participate in previous diffusions, causing lack of structural information. In the extreme case, new nodes that are absent in the training stage could appear in the test stage. If the graph representation is only learned from the structural relationships between nodes, which is exactly what DeepCas does, embedding vectors learned from the structure will not be available for these new nodes. Text could help in these cases, based on the intuition that nodes with similar text content might be close to each other in the embedding space. This motivates us to jointly model text and network so that we can effectively embed all nodes into the hidden space, which forms the basis of learning a good representation of graphs.

To better utilize the structural and textual information of nodes, a gating mechanism is designed to dynamically fuse the node representations from two sources, based on how well each representation is learned. To incorporate the text information from both diffusion items and nodes, an attention mechanism is employed over

node text, which is conditioned on their interactions with item text. Empirical evaluations demonstrate that incorporating text information benefits the cascade prediction task, and that the proposed gating mechanism is superior to alternatives, including a simple combination of text and structure information, and standard multimodal learning.

CHAPTER II

Related Work

To facilitate the reading of the following chapters, this chapter gives a brief introduction of existing algorithms and concepts related to representation learning and deep learning, including word embeddings, convolutional neural networks [57], recurrent neural networks [89], memory networks [99], attention mechanisms, and node embedding methods like deepwalk [81]. Readers who are already familiar with these methods or concepts can safely skip reading this chapter.

2.1 Word embeddings

Conventionally, natural language processing systems treat words discretely, by encoding each word using a unique id. These encodings are arbitrary, providing no information about the relationships between words. Furthermore, such representation leads to data sparsity, demanding more data to successfully train a model. To overcome these issues, researchers proposed to use vector representations, or embeddings, to encode words. The intuition is that words could be embedded in a hidden space, where distance between words indicates some form of semantic closeness. For example, in this space the vector of *cat* could be very close to that of *dog*.

Among the vector based methods, word2vec [69] is an efficient and well-performed predictive model for learning word embeddings. These embeddings are learned in an

unsupervised manner, such that each word in a sentence is trained to predict its surrounding words, or vice versa. To overcome the problem of large-sized vocabulary, word2vec employs techniques like negative sampling. Readers who are interested could refer to the original paper [69].

2.2 Convolutional neural networks

Convolutional neural networks (CNNs) [57], are neural networks with layers that use a mathematical operation called convolution. Convolutional layers make CNNs good at processing any form of locally connected data, e.g., images. Suppose a convolutional layer takes as input a 2-dimensional image, or matrix, with size 3×4. To explain the convolution operation, we can imagine that there is a flashlight that is sliding across the image from left to right, top to bottom, each time shining over a 2×2 area. In the terminology of deep learning, this flashlight is called a filter. An example from [37] well explains how exactly the convolution computation is performed, which is illustrated in Figure 2.1.

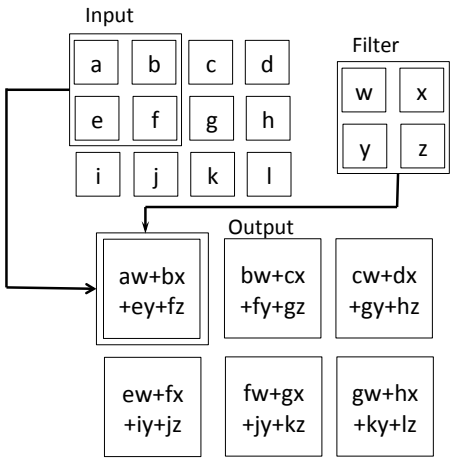


Figure 2.1: An example to compute convolution. The boxes with arrows indicate how the upper-left element of the output matrix is formed by applying the filter to the corresponding upper-left region of the input matrix.

The output of a convolutional layer can again be treated as an image, serving as

the input to another convolutional layer. In this way, multiple convolutional layers could be stacked, learning more and more abstract features.

2.3 Recurrent neural networks

Recurrent neural networks or RNNs [89] are a family of neural networks specialized for sequential data. They are called *recurrent* due to the presence of loops in their structures. Below we first introduce RNNs in their simplest form, followed by their variants.

2.3.1 Standard RNNs

As Figure 2.2 (a) shows, a recurrent neural network module, N, takes x_t as input at time step t, and maintains a hidden state vector h_t, which is fed as an input to N in the next step $t+1$. The input x_t could be the vector representation of the t-th word in a sentence. In the simplest case, the vector could be a one-hot vector, such that the i-th word in the vocabulary will have a single 1 in the i-th entry of its vector and all the others 0. The learned representation h_t summarizes, with certain loss, the past sequence of inputs $(x_0, x_1, ..., x_t)$ up to time step t. Mathematically, h_t is computed as

$$h_t = f(Ux_t + Wh_{t-1}), \tag{2.1}$$

where f is a nonlinear function such as tanh or sigmoid. U and W are parameters to be learned.

As Figure 2.2 (b) shows, we could unroll the loop in the structure. This will lead to a network that looks very similar to a normal neural network with multiple layers. A major difference is that in RNN, all unrolled modules share the same set of parameters as module N, while in a normal neural network, parameters in different layers are typically independent.

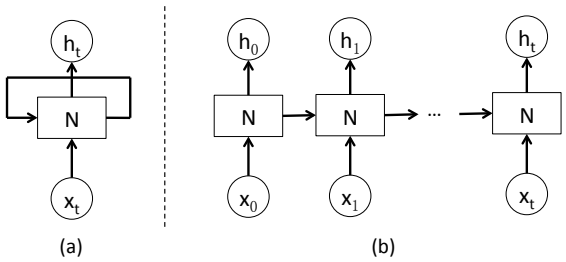


Figure 2.2: (a) A recurrent neural network. (b) An unrolled recurrent neural network.

The problem with RNNs is related to the modeling of long-term dependencies. Consider predicting the underscored word in the text "They offer cakes or bread for breakfast. Though bread is great, I prefer cakes." In order to find the relevant information to predict the word *cakes*, we need to go back until we reach the third word from the beginning of the sentence. Theoretically, RNNs are able to handle such long-term dependencies. Unfortunately in practice, RNNs seem to be unable to do so. Explanations for this phenomenon are explored in [8].

2.3.2 Variants of RNNs

Two variants of RNNs are widely used in the literature – Long Short Term Memory networks (LSTMs) [44] and Gated Recurrent Units (GRUs) [20]. They are capable of learning long-term dependencies by including structures called gates, which regulate the amount of information to go through from previous steps. Here we introduce GRUs in details, as they are simpler in structure than LSTMs, thus being more computationally efficient, while keeping similar performance.

Similar to standard RNNs, GRUs also compute the updated hidden state h_t based on current input x_t and previous state h_{t-1}. The difference lies in how the computation is performed. Specifically, GRUs first compute an update gate:

$$u_t = \sigma(W^{(u)}x_t + U^{(u)}h_{t-1}), \tag{2.2}$$

followed by a reset gate computed similarly but with different weights:

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}), \tag{2.3}$$

where $\sigma(.)$ is the sigmoid function. $W^{(u)}$, $W^{(r)}$, $U^{(u)}$, and $U^{(r)}$ are parameters. The new hidden state could then be computed:

$$\hat{h}_t = \tanh(Wx_t + r_t \cdot Uh_{t-1}), \tag{2.4}$$

where $\cdot$ represents an element-wise product. We can see that if the reset gate unit r_t is close to zero, it tends to ignore previous information summarized by h_{t-1}, and only stores the new information from input x_t. The final hidden state is a combination of current and previous time steps:

$$h_t = u_t \cdot \hat{h}_{t-1} + (1 - u_t) \cdot h_{t-1}. \tag{2.5}$$

If the update gate u_t is close to one, we can copy information from previous states through many time steps, allowing the modeling of long-term dependencies.

2.4 Memory networks

There are many tasks that require the access to a long-term memory component so that reasoning could be made based on the accessed information. Consider a task where a story is told, after which a list of relevant questions have to be answered. A simple example is shown as follows:

Example II.1. Consider the following story:

1. Jim moved to garden.

2. Jim went to kitchen.

3. Jim drops apple there.

Question: where is Jim?

Answer: kitchen.

In this example, the second sentence *Jim went to kitchen* provides the most important supporting factor to produce the answer. In theory, RNNs are able to accomplish such tasks by compressing the learned representation of the story in hidden states, which can be regarded as the memory of RNNs. However, their memory is typically too small, and the compression of knowledge might lead to forgetting of facts from the past.

Memory networks (MemNNs) [119, 99] are introduced to rectify this issue. They are a class of models that combine large memory with learning component to access it, and reasoning are incorporated by applying attention mechanisms over memory.

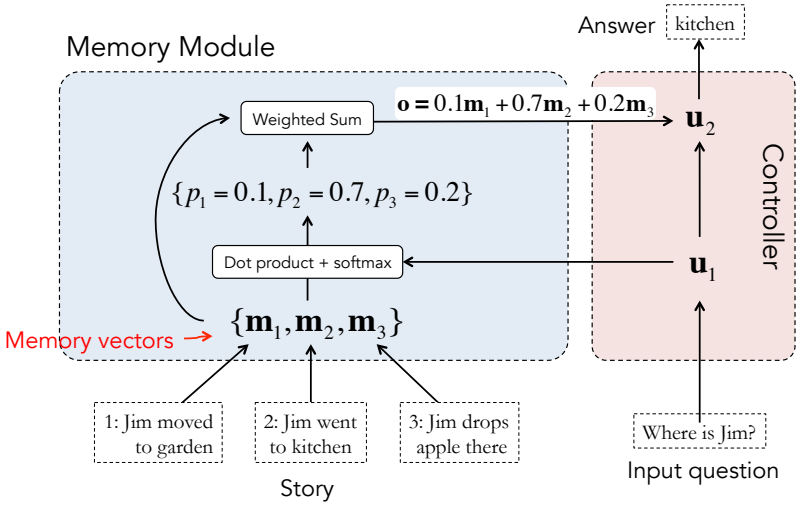


Figure 2.3: An end-to-end memory network model.

Figure 2.3 shows a one-layer structure of end-to-end memory networks [99], together with how the example story is processed to produce the final answer. In this example, words are converted to their embedding vectors, based on which sentence representations are obtained. The representation of i-th sentence in the story is stored

in memory cell as vector $\mathbf{m_i}$, while the sentence of the input question is represented as $\mathbf{u}_1$.

In order to retrieve the most important memory cells with respect to the question, an **attention mechanism** is employed. Attention mechanisms in neural networks are inspired by the visual attention mechanism of humans. This visual mechanism enables humans to focus on particular areas of their visual inputs while perceiving the rest in "low resolution". The attended areas could then be adjusted over time to process more information. Here for memory networks, attention is implemented as the match between $\mathbf{u}_1$ and each memory $\mathbf{m_i}$ by taking the dot product followed by a softmax:

$$p_i = Softmax(\mathbf{u}_1^T \mathbf{m}_i), \tag{2.6}$$

where $Softmax(zi) = e^{z_i} / \sum_j e^{z_j}$. In this way, p_i can be viewed as the probability that each memory cell is attended. In the example story, we want to train our memory network in a way such that the second sentence *Jim went to kitchen*, where the answer could be found, scores the highest attention. That is, the network will learn to assign p_2 the highest score given the story and the question. The output memory representation $\mathbf{o}$ can then be computed as the sum over the memory vectors $\mathbf{m_i}$, weighted by the probability vector $\mathbf{p}$.

$$\mathbf{o} = \sum_i p_i \mathbf{m_i}. \tag{2.7}$$

Based on some transformation of $\mathbf{u}_1$ and $\mathbf{o}$, the final output representation $\mathbf{u}_2$ could be computed. The transformation could be as simple as $\mathbf{u}_2 = W(\mathbf{u}_1 + \mathbf{o})$, where W is a parameter to be learned. The representation $\mathbf{u}_2$ is then used to generate the prediction word *kitchen*.

There are many extensions to make the memory networks more powerful. For example, the layer of memory network could be stacked multiple times, so that a

more complex and non-linear representation could be learned; temporal encoding could be incorporated so that we could account for the order of the memory cells.

2.4.1 Node embeddings

Networks are traditionally represented as affiliation matrices or discrete sets of nodes and edges. For example, we can have a matrix with $A_{ij} = 1$ if there is an edge from node i to node j. Modern representation learning methods attempt to represent nodes as high-dimensional vectors in a continuous space (a.k.a., node embeddings) so that nodes with similar embedding vectors share similar structural properties (e.g., [81, 107, 38]).

Much of this work is inspired by the huge success of representation learning applied to various domains such as text [7] and image [54]. One of the earliest approaches, DeepWalk [81], makes an analogy between the nodes in networks and the words in natural language. By doing random walks on graphs, sequences of nodes are sampled from graphs, which are analogous to textual sentences. In this way, node embeddings could be learned in the same way as we learn word embeddings from sentences, simply by feeding the sampled node sequences to the word2vec algorithm [69].

CHAPTER III

Deep Memory Networks for Attitude Identification

We consider the task of identifying attitudes towards a given set of entities from text. Conventionally, this task is decomposed into two separate subtasks: target detection that identifies whether each entity is mentioned in the text, either explicitly or implicitly, and polarity classification that classifies the exact sentiment towards an identified entity (the target) into positive, negative, or neutral.

Instead, we show that attitude identification can be solved with an end-to-end machine learning architecture, in which the two subtasks are interleaved by a deep memory network. In this way, signals produced in target detection provide clues for polarity classification, and reversely, the predicted polarity provides feedback to the identification of targets. Moreover, the treatments for the set of targets also influence each other – the learned representations may share the same semantics for some targets but vary for others. The proposed deep memory network outperforms methods that do not consider the interactions between the subtasks or those among the targets, including conventional machine learning methods and the state-of-the-art deep learning models.

3.1 Introduction

In many scenarios, it is critical to identify people's attitudes [1] towards a set of entities. Examples include companies who want to know customers' opinions about their products, governments who are concerned with public reactions about policy changes, and financial analysts who identify daily news that could potentially influence the prices of securities. In a more general case, attitudes towards all entities in a knowledge base may be tracked over time for various in-depth analyses.

Different from a *sentiment* which might not have a target (e.g., "I feel happy") or an *opinion* which might not have a polarity (e.g., "we should do more exercise"), an *attitude* can be roughly considered as a sentiment polarity towards a particular target (e.g., "WSDM is a great conference"). Therefore, the task of attitude identification has been conventionally decomposed into two separate subtasks: target detection that identifies whether an entity is mentioned in the text, either explicitly or implicitly, and polarity classification that classifies the exact sentiment towards an identified target, usually into three categories: positive, negative, and neutral.

Solving the two subtasks back-to-back is by no means unreasonable, but it may not be optimal. Specifically, intrinsic interactions between the two subtasks may be neglected in such a modularized pipeline. Indeed, signals identified in the first subtask – both the words that refer to the target and the positions of these words, could provide useful information for the polarity of sentiments. For example, the identified target in the sentence "*this Tiramisu cake is* _" indicates a high probability that a sentimental word would appear in the blank and is highly likely to be related to flavor or price. On the other hand, sentimental expressions identified in the second subtask and their positions could in turn provide feedback to the first task and signal the existence of the target. For example, the phrase "*user friendly*" signaling pos-

[1] "*The way you think and feel about someone or something*," as defined by Merriam-Webster. `http://www.merriam-webster.com/dictionary/attitude`

itive sentiment in "*the new Keynote is user friendly* " provides good evidence that "*Keynote*" is a software (the target) instead of a speech (not the target). In addition, models learned for certain targets and their sentiments may share some important dimensions with each other while differing on other dimensions. For example, two targets *food* and *service* may share many sentimental expressions, but the sentence "*we have been waiting for food for one hour*" is clearly about the service instead of the food. Failure to utilize these interactions (both between tasks and among targets) may compromise the performance of both subtasks.

Recent developments of deep learning has provided the opportunity of an alternative to modularized pipelines, in which machine learning and natural language processing tasks can be solved in an *end-to-end* manner. With a carefully designed multi-layer neural network, learning errors backpropagate from upper layers to lower layers, which enables deep interactions between the learning of multi-grained representations of the data or multiple subtasks. Indeed, deep learning has recently been applied to target-specific sentiment analysis (mostly the second subtask of attitude identification) and achieved promising performance, where a given target is assumed to have appeared exactly once in a piece of text and the task is to determine the polarity of this text [112, 130, 104]. A deep network structure learns the dependency between the words in the context and the target word. In another related topic known as multi-aspect sentiment analysis, where the goal is to learn the fine-grained sentiments on different aspects of a target, some methods have attempted to model aspects and sentiments jointly. Aspects are often assumed to be mentioned explicitly in text, so that the related entities can be extracted through supervised sequence labeling methods [66, 62, 132]; aspects mentioned implicitly can be extracted as fuzzy representations through unsupervised methods such as topic models [68, 115, 91]. While unsupervised methods suffer from low accuracy, it is usually difficult for supervised methods, like support vector machines (SVMs) [52], to interleave aspect extraction

and sentiment classification.

In this paper, we show that the accuracy of attitude identification can be significantly improved through effectively modeling the interactions between subtasks and among targets. The problem can be solved with an end-to-end machine learning architecture, where the two subtasks are interleaved by a deep memory network. The proposed model also allows different targets to interact with each other, by sharing a common semantic space and simultaneously keeping their own space, making it possible for all targets to be learned in a unified model. The proposed deep memory network outperforms models that do not consider the subtask or target interactions, including conventional supervised learning methods and state-of-the-art deep learning models.

The rest of the paper is organized as follows. Section 3.2 summarizes the related literature. In Section 3.3, we describe how the deep neural network is designed to incorporate the interaction both between subtasks and among targets. We present the design and the results of empirical experiments in Section 3.4 and Section 3.5, and then conclude the paper in Section 3.6.

3.2 Related work

Sentiment analysis has been a very active area of research [80, 84]. While sentiment in general does not need to have a specific target, the notion of *attitude* is usually concerned with a sentiment towards a target entity (someone or something). As one category of sentiment analysis, there is much existing work related to attitude identification, which generally takes place in three domains: multi-aspect sentiment analysis in product reviews, stance classification in online debates, and target-dependent sentiment classification in social media posts. Below we categorize existing work by the problem settings, e.g., whether the target is required to be explicitly mentioned.

Explicitly tagged targets. There has been a body of work that classifies the

sentiment towards a particular target that is explicitly mentioned and tagged in text, mostly applied to social media text such as Tweets. Due to the short length of Tweets, many models assume that targets appear exactly once in every post. Jiang et al. [48] developed seven rule-based target-dependent features, which are fed to an SVM classifier. Dong et al. [27] proposed an adaptive recursive neural network that propagates sentiment signals from sentiment-baring words to specific targets on a dependence tree. Vo et al. [112] split a Tweet into a left context and a right context according to a given target, and used pre-trained word embeddings and neural pooling functions to extract features. Zhang et al. [130] extended this idea by using gated recursive neural networks. The paper most relevant to ours is Tang et al. [104], who applied Memory Networks [99] to the task of multi-aspect sentiment analysis. Aspects are given as inputs, assuming that the aspect has already been annotated in the text. Their memory network beat all LSTM-based networks but did not outperform SVM with hand-crafted features.

Model structures for target-dependent sentiment classification heavily rely on the assumption that the target appears in the text *explicitly*, and exactly *once*. These models could degenerate when a target is implicitly mentioned or mentioned multiple times. Additionally, they do not consider the interactions between the subtasks (target detection and sentiment classification) or targets.

Given target, one per instance. In the problem of stance classification, the target, mentioned explicitly or implicitly, is given but not tagged in a piece of text. The task is only to classify the sentiment polarity towards that target. Most methods train a specific classifier for each target and report performance separately per target. Many researchers focus on the domain of online debates. They utilized various features based on n-grams, part of speech, syntactic rules, and dialogic relations between posts [114, 42, 32, 85]. The workshop SemEval-2016 presented a task on detecting stance from tweets [72], where an additional category is added for the given target,

indicating the absence of sentiment towards the target. Mohammad et al. [73] beat all teams by building an SVM classifier for each target.

As stance classification deals with only one given target per instance, it fails to consider the interaction between target detection and sentiment classification. Furthermore, the interplay between targets is ignored by training a separate model per target.

Explicit targets, not tagged. In the domain of product reviews, a specific aspect of a product could be considered as a target of attitudes. When the targets appear in a review but are not explicitly tagged, they need to be extracted first. Most work focuses on extracting explicitly mentioned aspects. Hu et al. [45] extracted product aspects via association mining, and expanded seed opinion terms by using synonyms and antonyms in WordNet. When supervised learning approaches are taken, both tasks of aspect extraction and polarity classification can be cast as a binary classification problem [52], or as a sequence labeling task and solved using sequence learning models such as conditional random fields (CRFs) [66, 62] or hidden Markov models (HMMs) [132].

Implicit targets. There are studies that attempt to address the situation when aspects could be implicitly mentioned. Unsupervised learning approaches like topic modeling treat aspects as topics, so that topics and sentiment polarity can be jointly modeled [68, 115, 91]. The workshop of SemEval-2015 announced a task of aspect based sentiment analysis [82], which separates aspect identification and polarity classification into two subtasks. For aspect identification, top teams cast aspect category extraction as a multi-class classification problem with features based on n-grams, parse trees, and word clusters.

Although aspect identification and polarity classification are modeled jointly here, it is hard to train unsupervised methods in an end-to-end way and directly optimize the task performance.

Deep learning for sentiment analysis. In the general domain of sentiment analysis, there has been an increasing amount of attention on deep learning approaches. In particular, Bespalov et al. [9] used Latent Semantic Analysis to initialize the word embedding, representing each document as the linear combination of n-gram vectors. Glorot et al. [36] applied Denoising Autoencoders for domain adaptation in sentiment classification. A set of models have been proposed to learn the compositionality of phrases based on the representation of children in the syntactic tree [95, 96, 43]. These methods require parse trees as input for each document. However, parsing does not work well on user generated contents, e.g., tweets [35]. Liu et al. [63] used recurrent neural networks to extract explicit aspects in reviews.

Compared to the existing approaches, our work develops a novel deep learning architecture that emphasizes the interplay between target detection and polarity classification, and the interaction among multiple targets. These targets can be explicitly or implicitly mentioned in a piece text and do not need to be tagged a priori.

3.3 AttNet for Attitude Identification

We propose an end-to-end neural network model to interleave the target detection task and the polarity classification task. The **target detection** task is to determine whether a specific target occurs in a given context either explicitly or implicitly. The **polarity classification** task is to decide the attitude of the given context towards the specific target if the target occurs in the context. Formally, a **target detection classifier** is a function mapping pairs of targets and contexts into binary labels, (context, target) $\rightarrow$ {present, absent}. A **polarity classifier** is a function mapping pairs of targets and contexts into three attitude labels, (context, target) $\rightarrow$ {positive, negative, neutral}. For example, given a **context**, *green is the way forward!*, and a **target**, *climate change is a real concern.*, the correct label is *present* for the target detection and *positive* for the polarity classification.

Our model builds on the insight that the target detection task and the polarity classification task are deeply coupled in several ways.

- The polarity classification depends on the target detection because the polarity is meaningful only if the target occurs in the context. Conversely, the polarity classification task provides indirect supervision signals for the target detection task. For example, if the attitude label *negative* is provided for a context-target pair, the target must have occurred in the context following the definition. Such indirect supervision signals are useful especially when the target only occurs in the context implicitly, as in the example *we have been waiting for food for one hour*, where *service* is the target.

- The signal words in the target detection and the polarity classification task are usually position-related: the signal words to determine the polarity are usually the surrounding words of the signal words to detect the target. Moreover, when a context has multiple targets, the signal words usually cluster for different targets [45, 84].

- Different targets interact in both the target detection task and the polarity classification task. Intuitively, some context words could have the same meaning for many targets, while the meaning of other context words could vary for different targets. This point has been illustrated by our *service* and *food* example.

Specifically, our model introduces several techniques building on the interaction between the target detection task and the polarity classification task accordingly.

- The output of the target detection is concatenated as part of the input of the polarity classification task to allow the polarity classification condition on target detection. Polarity classification labels are also used to train the target detection classifier by back-propagating the errors of the polarity classification to the target detection end-to-end.

- The attention of polarity classification over context words are preconditioned by the attention of target detection. The polarity classification task benefits from such precondition especially when there are multiple targets in the context.

- Target-specific projection matrices are introduced to allow some context words to have similar representations among targets and other context words to have distinct representations. These matrices are all learned in an end-to-end fashion.

We propose a deep memory network model implementing the above motivation and ideas. In the rest of this section, we describe a single layer version of our model following a brief introduction to the idea of the memory network model. Then we extend the expressiveness and capability of the model by stacking multiple layers.

3.3.1 Background: Memory Networks

As recently proposed models, end-to-end memory networks [99] have been successfully applied to language modeling, question answering, and aspect-level sentiment analysis [104], which generates superior performance over alternative deep learning methods, e.g., LSTM.

Given a context (or document, e.g., "*we have been waiting for food for one hour*") and a target (e.g., *service*), a memory network layer converts the context into a vector representation by computing a weighted sum of context word representations. The weight is a score that measures the relevance between the context word and the target (e.g., a higher score between the words *waiting* and *service*), based on their vector representations, or embeddings. The vector representation of the context is then passed to a classifier for target detection or polarity classification. An attractive property is that all parameters, including the target embeddings, context word embeddings and scores, are end-to-end learnable without additional supervision signals.

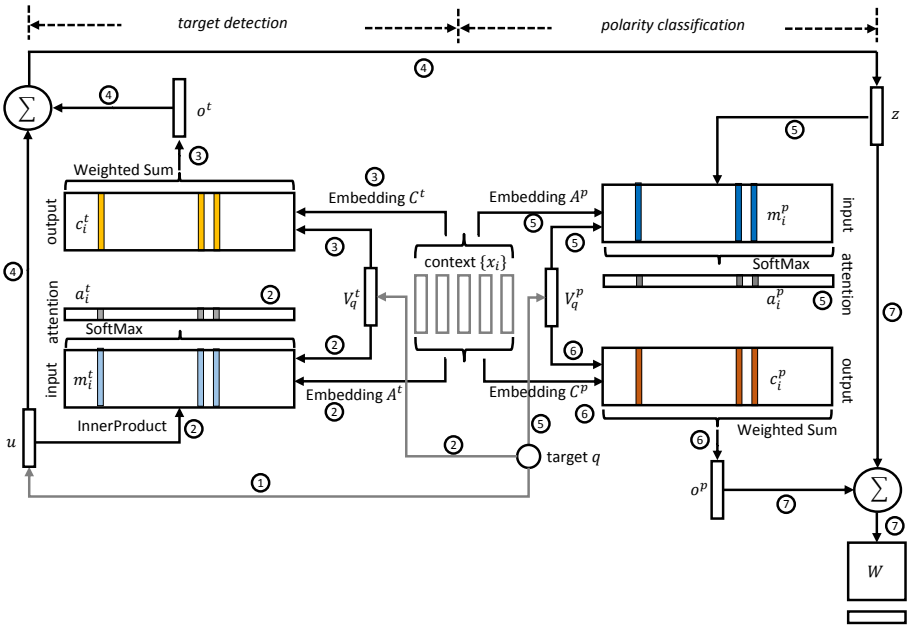


Figure 3.1: A single layer version of our model. Key submodules are numbered and correspondingly detailed in the text.

Our model improves the original memory network models for attitude identification by (1) interleaving the target detection and polarity classification subtasks and (2) introducing target-specific projection matrices in representation learning, without violating the end-to-end trainablity.

3.3.2 Single Layer Model

We begin by describing our model in the single layer case, shown in Figure 3.1. Hereafter for simplicity, we refer to the task of target detection as **TD**, and polarity classification as **PC**.

(1) Target Embedding Each query target is represented as a one-hot vector, $\mathbf{q} \in \mathbb{R}^{N_{\text{target}}}$, where N_{target} is the number of targets. All targets share a target embedding matrix $\mathbf{B} \in \mathbb{R}^{d \times N_{\text{target}}}$, where d is the embedding dimensionality. The matrix $\mathbf{B}$ converts a target into its embedding vector $\mathbf{u} = \mathbf{Bq}$, which is used as the input for the **TD** task.

(2) Input Representation and Attention for TD We compute match scores between the context (or document) and the target for content-based addressing. The context is first converted into a sequence of one-hot vectors, $\{\mathbf{x}_i \in \mathbb{R}^{N_{\text{voc}}}\}$, where $\mathbf{x}_i$ is the one-hot vector for the i-th word in the context and N_{voc} is the number of words in the dictionary. The entire set of $\{\mathbf{x}_i\}$ are then embedded into a set of input representation vectors $\{\mathbf{m}_i^t\}$ by:

$$\mathbf{m}_i^t = \mathbf{V}_q^t \mathbf{A}^t \mathbf{x}_i$$

where $\mathbf{A}^t \in \mathbb{R}^{d \times N_{\text{voc}}}$ is the word embedding matrix shared across targets, superscript t stands for the **TD** task, and $\mathbf{V}_q^t \in \mathbb{R}^{d \times d}$ is a target-specific projection matrix for target q, which allows context words $\mathbf{x}_i$ to share some semantic dimensions for some targets while vary for others.

In the embedding space, we compute the match scores between the target input representation $\mathbf{u}$ and each context word representation $\mathbf{m}_i^t$ by taking the inner product followed by a softmax, $a_i^t = \text{SoftMax}(\mathbf{u}^\intercal \mathbf{m}_i^t)$, where $\text{SoftMax}(w_i) = \exp(w_i)/\sum_j \exp(w_j)$. In this way, $\mathbf{a}^t$ is a soft attention (or probability) vector defined over the context words.

(3) Output Representation for TD A different embedding matrix, $\mathbf{C}^t \in \mathbb{R}^{d \times N_{\text{voc}}}$, is introduced for flexibility in computing the output representation of context words by:

$$\mathbf{c}_i^t = \mathbf{V}_q^t \mathbf{C}^t \mathbf{x}_i$$

The response output vector $\mathbf{o}^t$ is then a sum over the outputs $\mathbf{c}_i^t$, weighted by the attention vector from the input: $\mathbf{o}^t = \sum_i a_i^t \mathbf{c}_i^t$.

(4) Interleaving TD and PC In the single layer case, the sum of the output vector $\mathbf{o}^t$ and the target query embedding $\mathbf{u}$ is then passed to the **PC** task, $\mathbf{z} = \mathbf{o}^t + \mathbf{u}$.

(5) Input Representation and Attention for PC Similar to the **TD** task, we convert the entire set of $\{\mathbf{x}_i\}$ into input representation vectors $\{\mathbf{m}_i^p\}$ by:

$$\mathbf{m}_i^p = \mathbf{V}_q^p \mathbf{A}^p \mathbf{x}_i$$

where $\mathbf{A}^p \in \mathbb{R}^{d \times N_{\text{voc}}}$ is the input embedding matrix for **PC**. We use separate embedding matrices $\mathbf{A}^t, \mathbf{A}^p$ for **TD** and **PC**, as the words could have different semantics in the two tasks. For similar reasons, we use different projection matrices $\mathbf{V}_q^t, \mathbf{V}_q^p$ for the two tasks.

Given the polarity input representation $\{\mathbf{m}_i^p\}$, we also compute the soft attention over the context words for polarity identification, $a_i^p = \text{SoftMax}(\mathbf{z}^\intercal \mathbf{m}_i^p)$.

(6) Output Representation for PC There is also one corresponding output vector $\mathbf{c}_i^p$ in **PC** for each $\mathbf{x}_i$:

$$\mathbf{c}_i^p = \mathbf{V}_q^p \mathbf{C}^p \mathbf{x}_i$$

where $\mathbf{C}^p \in \mathbb{R}^{d \times N_{\text{voc}}}$ is the polarity output embedding matrix. It has been observed that sentiment-bearing words are often close to the target [45, 84]. Based on this observation, attentions, or positions of important words that identify the target in the first module, could provide prior knowledge to learn the attention of the second module. Therefore we compute the final attention vector as a function of original attentions of both tasks:

$$\mathbf{b}^p = \mathbf{a}^p + \lambda f(\mathbf{a}^t) \tag{3.1}$$

where $\lambda > 0$ controls the importance of the second term, and f is a moving average function which shifts attentions from words of high values to their surrounding neighbors. The output vector is $\mathbf{o}^p = \sum_i b_i^p \mathbf{c}_i^p$.

(7) Prediction for TD and PC To predict whether a target presents, the sum of the output vector of target classification $\mathbf{o}^t$ and the target query vector $\mathbf{u}$ is passed through a weight matrix $\mathbf{W}^t \in \mathbb{R}^{2\times d}$ (2 is the number of classes: *present*, *absent*) and a softmax operator to produce the predicted label, a vector of class probabilities: $\mathbf{y}^t = \text{SoftMax}(\mathbf{W}^t(\mathbf{o}^t + \mathbf{u}))$.

Similarly, the sum of the output vectors $\mathbf{o}^p$ of **PC** and its input vector $\mathbf{z}$ is then passed through a weight matrix $\mathbf{W}^p \in \mathbb{R}^{3\times d}$ and a softmax operator to produce the predicted attitude label vector, $\mathbf{y}^p = \text{SoftMax}(\mathbf{W}^p(\mathbf{o}^p + \mathbf{z}))$.

3.3.3 Multiple Layer Model

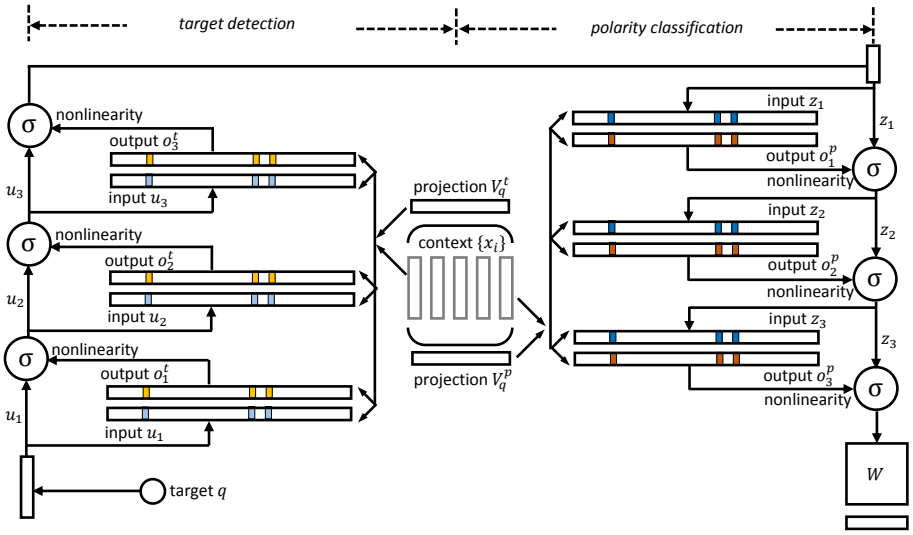


Figure 3.2: A three layer version of our model. Both the **TD** and **PC** modules have three stacked layers.

We now extend our model to stacked multiple layer case. Figure 3.2 shows a three layer version of our model. The layers are stacked in the following way:

Functionality of Each Layer For **TD**, the input to the (k+1)-th layer is the sum of the output $\mathbf{o}_k^t$ and the input $\mathbf{u}_k$ from the k-th layer, followed by a sigmoid nonlinearity: $\mathbf{u}_{k+1} = \sigma(\mathbf{H}^t\mathbf{u}_k + \mathbf{o}_k^t)$, where $\sigma(x) = 1/(1+\exp(x))$ is the sigmoid

function and $\mathbf{H}^t$ is a learnable linear mapping matrix shared across layers. For the **PC** task, the input to the first layer is the transformed sum from the last layer of the **TD** module, $\mathbf{z}_1 = \sigma(\mathbf{H}^t\mathbf{u}_{K_t} + \mathbf{o}^t_{K_t})$, where K_t is the number of stacked layers in the **TD** task. Thus the prediction of polarity would depend on the output of the **TD** task and conversely the **TD** task would benefit from indirect supervision from the **PC** task by backward propagation of errors. Similarly for **PC**, the input to the (k+1)-th layer is the sum of the output $\mathbf{o}^p_k$ and the input $\mathbf{z}_k$ from the k-th layer, followed by a sigmoid nonlinearity: $\mathbf{z}_{k+1} = \sigma(\mathbf{H}^p\mathbf{z}_k + \mathbf{o}^p_k)$.

Attention for PC In the single layer case, the attention for **PC** is based on that of the **TD** module. When layers are stacked, all layers of the first module collectively identify important attention words to detect the target. Therefore we compute the averaged attention vector across all layers in the **TD** module $\bar{\mathbf{a}}^t = \frac{1}{K_t}\sum_{k=1}^{K_t}\mathbf{a}^t_k$. Accordingly for k-th layer of the **PC** module, the final attention vector is $\mathbf{b}^p_k = \mathbf{a}^p_k + \lambda f(\bar{\mathbf{a}}^t)$, and the output vector is $\mathbf{o}^p_k = \sum_i b^p_{ki}\mathbf{c}^p_{ki}$.

Embedding and Projection Matrix Tying. The embedding matrices and projection matrices are constrained to ease training and reduce the number of parameters following [99]. The embedding matrices and the projection matrices are shared for different layers. Specifically, using subscription (k) denote the parameters in the k-th layer, for any layer k, we have $\mathbf{A}^{t(1)} \equiv \mathbf{A}^{t(k)}$, $\mathbf{C}^{t(1)} \equiv \mathbf{C}^{t(k)}$, $\mathbf{A}^{p(1)} \equiv \mathbf{A}^{p(2)}$, $\mathbf{C}^{p(1)} \equiv \mathbf{C}^{p(k)}$, $\mathbf{V}^{t(1)}_q \equiv \mathbf{V}^{t(k)}_q$ and $\mathbf{V}^{p(1)}_q \equiv \mathbf{V}^{p(k)}_q$.

Predictions for TD and PC. The prediction stage is similar to the single-layer case, with the prediction based on the output of the last layer K_t (For **TD**) and K_p (For **PC**). For the **TD** task, $\mathbf{y}^t = \text{SoftMax}(\mathbf{W}^t\sigma(\mathbf{H}^t\mathbf{u}_{K_t} + \mathbf{o}^t_{K_t}))$, while for **PC**, $\mathbf{y}^p = \text{SoftMax}(\mathbf{W}^p\ \sigma(\mathbf{H}^p\mathbf{z}_{K_p} + \mathbf{o}^p_{K_p}))$.

3.3.4 End-to-End Multi-Task Training

We use cross entropy loss to train our model end-to-end given a set of training data $\{ct_i, q_j, g^t_{ij}, g^p_{ij}\}$, where ct_i is the i-th context (or document), q_j is the j-th target, g^t_{ij}, g^p_{ij} are the ground-truth label for the **TD** and **PC** tasks respectively. The training is to minimize the objective function:

$$\mathcal{L} = -\sum_i \sum_j \Big(\log(\mathbf{y}^t_{ij}(g^t_{ij})) + \mathbb{1}_{g^t_{ij}} \log(\mathbf{y}^p_{ij}(g^p_{ij})) \Big)$$

where $\mathbf{y}^t_{ij}$ is a vector of predicted probability for each class of **TD**, $\mathbf{y}^t_{ij}(s)$ selects the s-th element of $\mathbf{y}^t_{ij}$, $\mathbb{1}_{g^t_{ij}}$ equals to 1 if g^t_{ij} equals to class *present* and 0 otherwise. Note that when a target does not exist, the polarity term plays no role in the objective because the value of $\mathbb{1}_{g^t_{ij}}$ is zero.

3.4 Experiment Setup

In the experiments, we compare our model to conventional approaches and alternative deep learning approaches on three real world data sets, and we show the superior performance of our model. We also experiment with variants of our models as credit assignments for the key components in our model.

3.4.1 Data Sets

We examine our models on three domains that are related to attitude classification: online debates (Debates), multi-aspect sentiment analysis on product review (Review), and stance in tweets (Tweets).

Debates. This data set is from the Internet Argument Corpus version 2[2]. The data set consists of political debates on three Internet forums[3] . On these forums, a

[2] `https://nlds.soe.ucsc.edu/iac2`.

[3] 4forums(`http://www.4forums.com/political/`), ConvinceMe(`http://www.convinceme.net/`) and

person can initiate a debate by posting a topic and providing sides such as *favor* vs. *against*. Examples of topics are *gun control*, *death penalty* and *abortion*. Other users participate in the debate by posting their arguments for one of the sides.

Tweets. This data set comes from a task of the workshop SemEval-2016 on detecting stance from tweets [72]. Targets are mostly related to ideology, e.g., *atheism* and *feminist movement*[4].

Review. This data set includes reviews of restaurants and laptops from SemEval 2014 [83] and 2015 [82], where subtasks of identifying aspects and classifying sentiment are provided. We merge two years' data to enlarge the data set, and only include aspects that are annotated in both years.

To guarantee enough training and test instances, for all the data sets we filter out targets mentioned in less than 100 documents. The original train-test split is used if provided, otherwise we randomly sample 10% data into test set. We further randomly sample 10% training data for validation. Text pre-processing includes stopword removal and tokenization by the CMU Twitter NLP tool [35]. The details of the data sets are shown in Table 3.1.

3.4.2 Metrics

For our problem, each data set has multiple targets, and each target can be classified into one of the outcomes: *absent* (do not exist), *neutral*, *positive*, and *negative*. If we treat each outcome of each target as one *category*, we can adopt common metrics for multi-class classification. Since most targets do not appear in most instances, we have a highly skewed class distribution, where measures like accuracy are not good choices [17].

Apart from *precision*, *recall* and AUC, we also use the *macro-average F-measure* [127].

CreateDebate(`http://www.createdebate.com/`)

[4]Since there is less than 10 tweets with neutral stance, we only consider *positive* and *negative* attitude by discarding these neutral tweets.

Table 3.1: Statistics of each data set.

Data set	Set	#docs	#pos	#neg	#neutral	#absent	#targets
Debates	train	24352	13891	10711	0	0	10
	val	2706	1530	1203	0	0	10
	test	3064	1740	1371	0	0	10
Tweets	train	2614	682	1253	0	679	5
	val	291	71	142	0	78	5
	test	1249	304	715	0	230	5
Review	train	5485	2184	1222	210	2336	9
	val	610	260	121	17	277	9
	test	1446	496	455	60	634	9

#pos means the number of documents with positive sentiment for each target. If one document contains positive sentiment towards two targets, it will be counted twice. *#absent* counts the number of documents without any attitude towards any target. *#targets* is the total number of targets appeared in one data set.

Let ρ_i and π_i be recall and precision respectively for a particular category i, $\rho_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i+\mathrm{FN}_i}$, $\pi_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i+\mathrm{FP}_i}$, where $\mathrm{TP}_i, \mathrm{FP}_i, \mathrm{FN}_i$ are the number of true positive, false positive, and false negative for category i. Given ρ_i and π_i, F-score of category i is computed as $F_i = \frac{2\pi_i\rho_i}{\pi_i+\rho_i}$. The macro-average F-score is obtained by taking the average over all categories. The final precision and recall are also averaged over individual categories. There is another micro-averaged F-measure, which is equivalent to accuracy. Therefore, we do not include it.

3.4.3 Baselines

We compare baseline methods from two large categories: conventional methods and alternative deep learning methods.

Each baseline method has various configurations, based on whether: (1) it trains *a single model* or *two separate models* for the **TD** and **PC** subtasks, and (2) it trains *one universal model* for *all* targets or *separated models* for *different* targets. To distinguish different configurations, we append *-sgl* when using a single model for the two subtasks, and *-sep* when using separate models for each subtask. Taking SVM

as an example, SVM-sgl directly classify targets into four classes: *absent*, *neutral*, *positive*, and *negative*. In contrast, SVM-sep first classifies each target into two classes: *absent*, *present*, and use a second model to classify polarity: *neutral*, *positive*, and *negative*. Moreover, we append *-ind* when individual targets are trained on separate models, or *-all* when one model for all targets.

3.4.3.1 Conventional baselines

SVM+features. SVM using a set of hand-crafted features has achieved the state-of-the-art performance in stance classification of SemEval 2016 task [73], online debates [42], and aspect-based sentiment analysis [104]. SVM has also demonstrated superior performance in document-level sentiment analysis compared with Conditional Random Field methods [116]. Therefore we include all features from these methods that are general across domains, and use a linear kernel SVM implemented by LIBSVM [15] for classification. We list the set of features:

Document info: basic counting features of a document, including the number of characters, the number of words, the average words per document and the average word length.

N-grams: word unigrams, bigrams, and trigrams. We insert symbols that represent the start and end of a document to capture cue words.

Sentiment: the number of positive and negative words counted from NRC Emotion Lexicon [74], Hu and Liu Lexicon [45], and MPQA Subjectivity Lexicon [120].

Target: presence of the target phrase in the text. Furthermore, if the target is present, we generate a set of target dependent features according to [48]. To get a sense of these features, for the target *iPhone* in text *I love iPhone*, a feature *love_arg* could be generated.

POS: the number of occurrences of each part-of-speech tag (POS).

Syntactic dependency: a set of triples obtained by Stanford dependency parser [24].

More specifically, the triple is of the form (rel, w_i, w_j), where rel represents the grammatical relation between word w_i and w_j, e.g., subject of.

Generalized dependency: the first word of the dependency triple is "backed off" to its part-of-speech tag [113]. Additionally, words that appear in sentiment lexicons are replaced by positive or negative polarity equivalents [113].

Embedding: the element-wise averages of the word vectors for all the words in a document. We use three types of word embeddings. Two of them are from studies on target-dependent sentiment classification [112, 130], which are the skip-gram embeddings of Mikilov et al. [70] and the sentiment-driven embeddings of Tang et al. [105]. The first type of embedding is trained on 5 million unlabeled tweets that contain emoticons, which guarantees that more sentiment related tweets are included. The second type of embedding is of 50 dimensions, which is publicly available[5]. The third type of embedding is also 50-dimensional, released by Collobert et al. [22] and trained on English Wikipedia[6].

Word cluster: the number of occurrences of word clusters for all words in text. We perform K-means clustering on the word vectors.

Apart from two standard SVM model configurations, *SVM-sep-ind* and *SVM-sgl-ind*, we also compare with a hybrid model *SVM-cmb-ind*, whose prediction is *absent* if *SVM-sep-ind* says so, and otherwise it follows the decisions of *SVM-sgl-ind*.[7]

3.4.3.2 Deep Learning Baselines

BiLSTM, **MultiBiLSTM** and **Memnet**. We also compare to the bidirectional LSTM (**BiLSTM**) model, the state-of-the-art on target-dependent sentiment classification [130]. Their variant of BiLSTM model assumes that the given target always appears exactly once, and can be tagged in text by starting and ending offsets. When

[5] `http://ir.hit.edu.cn/~dytang/`

[6] `http://ronan.collobert.com/senna/`

[7] *SVM-sgl-all* and *SVM-sep-all* have performance degeneration due to the interference of different targets. We do not include their results for simplicity.

such assumption fails, their model is equivalent to standard BiLSTM. We include the standard multi-layered bidirectional LSTM (**MultiBiLSTM**) [46] as an extension. Recently, Tang et al. [104] applied memory networks (**Memnet**) to multi-aspect sentiment analysis. Their results show memory network performs comparably with feature based SVM and outperforms all LSTM-related methods in their tasks.

CNN and **ParaVec**. We include related deep learning techniques beyond the sentiment analysis domain, such as the convolutional neural networks (**CNN**) [50] and **ParaVec** [56]. **ParaVec** require a huge amount of training data to reach decent performance. We enhance the performance of the ParaVec model by training over the merged training set of all data sets, plus the 5 million unlabeled tweets mentioned above.

Parser-dependent deep learning methods have also been applied to sentiment analysis [95, 96, 43]. These models are limited in our attitude identification problem for two reasons. First, they often work well with phrase-level sentiment labels, but only document-level sentiment labels are provided in our problems. Second, their parsers do not extend to user generated content, such as Tweets and Debates [35]. Our preliminary results show these methods work poorly on our problems and we do no include their results for simplicity.

For all deep learning methods, we report their *-sep-all* and *-sgl-all* version. Unlike SVM, deep methods perform quite well when using a single model for all targets, by casting the problem as a multi-task multi-class classification. Though not scalable, for the strongest baselines (BiLSTM and MultiBiLSTM), we in addition train a separate model for each target. Since *-sep-ind* works better than *-sgl-ind*, we only report the former one. The variants of memory networks are detailed below.

3.4.4 Variants of Proposed Model

To assign the credit of key components in our model, we construct a competing model *AttNet*. Unlike our proposed model, for *AttNet* the target-specific projection matrices $\mathbf{V}_q^p$ and $\mathbf{V}_q^t$ are replaced by the identity matrix and fixed during training. Thus the *AttNet* model interleave the target detection and polarity classification subtasks, but do not consider the interactions among targets. We refer our proposed model as *AttNet+*, which allows the projection matrices to be learned during training, and thus word semantics could vary for targets.

For *AttNet*, we report two settings in our experiments: *AttNet-ind* and *AttNet-all*. The former makes all targets share the same embedding, while the latter separates the embedding space completely for each target, i.e., targets are trained on separate models.

Table 3.2: Hyper-parameters for our method AttNet+.

Hyper-parameters	Tweets	Review	Debates
L1 coeff	1e-6	1e-4	1e-6
L2 coeff	1e-4	1e-8	1e-8
init learning rate	0.05	0.01	0.005
#layers(target)	4	4	3
#layers(sentiment)	4	8	6
prior attention λ	0.5	0.1	0.5

The embedding size is set to 100 for all data sets. The sliding window size of the moving average function in Equation 3.1 is set to 3. *#layers(target)* is the number of memory layers for target detection, and *#layers(sentiment)* is the number for sentiment classification. *prior attention* λ is the weight for prior attention in Equation 3.1.

3.4.5 Training Details

All hyper-parameters are tuned to obtain the best performance of F-score on validation set. The candidate embedding size set is {50, 100, 200, 300} for LSTM-related methods, SVM and CNN. The candidate number of clusters for K-means is

$\{50, 100, 150\}$. The candidate relaxing parameter C for SVM model is $\{2^7, 2^6, ..., 2^{-3}\}$. The CNN model has three convolutional filter sizes and their filter size candidates are $\{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}, \{2, 4, 6\}\}$, and the candidate number of filters is {50, 100, 200, 300}. For ParaVec, we experiment with both skip-gram model or bag-of-words model, and select the hidden layer size from $\{2^6, 2^7, ..., 2^{10}\}$.

We explored three weight initialization methods of word embeddings for LSTM-related and CNN baselines: (1) sampling weights from a zero-mean Gaussian with 0.1 standard deviation; (2) initializing from the pre-trained embedding matrix, and (3) using a fixed pre-trained embedding matrix.

Memory network models, including our model, are initialized by sampling weights from a zero-mean Gaussian with unit standard deviation. The candidate number of memory layers is $\{2, 3, ..., 9\}$. The prior attention parameter λ of our model is selected from $\{0, 0.1, 0.5, 0.9, 1\}$. The capacity of memory, which has limited impact on the performance, is restricted to 100 words without further tuning. A null symbol was used to pad all documents to this fixed size. To reduce the model complexity, the projection matrices are initialized in the way that each column is a one-hot vector.

Deep learning models are optimized by Adam [51]. The initial learning rate is selected from $\{0.1, 0.05, 0.01, 0.005, 0.001\}$, and L1-coefficient and L2-coefficient of regularizers are selected from $\{10^{-2}, 10^{-4}, ..., 10^{-10}\}$. The hyper-parameters of our model *AttNet+* for different data sets are listed in Table 3.2.

3.5 Experiment results

3.5.1 Overall Performance

The overall performance of all competing methods over data sets are shown in Table 3.3[8]. Evaluating with F-score and AUC, we make the following observations.

[8]The performance of all methods on the Review data set is lower than the other two because Review data set handles three polarities while the others only need to handle two polarities as shown

Our method *AttNet+* outperforms all competing methods significantly. This empirically confirms that interleaving target detection and polarity classification subtasks combined with target-specific representations could benefit attitude identification.

The variants of our model, *AttNet-all* and *AttNet-ind*, have already gained significant improvement over the strongest baselines on all data sets. More importantly, the two methods significantly outperforms the *Memnet-sep-all* and *Memnet-sep-all* baselines, which do not interleave the subtasks. Such empirical finding cast light on that interleaving the subtasks indeed improves the attitude identification performance. In contrast, separating the two subtasks of attitude identification could lead to performance degeneration.

Our model *AttNet+* also outperforms its variants, *AttNet-all* and *AttNet-ind*, on all data sets. The performance advantage of our model comes from the adoption of target-specific projection matrices in representation learning since the projection matrices are the only differences between these two methods. Even though the improvement from adopting target-specific projection matrices is not as tremendous as the techniques in interleaving the subtasks, the improvement is still significant. This results confirm that attitude identification could benefit from the learned representations that share the same semantics for many targets but vary for some targets.

By examining the *precision* and *recall* results, we find that the superior performance of our model is mainly from the significant improvement in *recall*, though both *precision* and *recall* are improved significantly on the Debates data set.

3.5.2 Performance on Subtasks

We have established that our model outperforms competing methods on all data sets. In order to further assign the credits of the improvement of our methods, we evaluate our models on the two subtasks: target detection and polarity classification,

in Table 3.1.

with results given in Table 3.4 and 3.5 respectively. Since different configurations of the same method work similarly, we only present the results where separate models are trained for each task. It can be seen from Table 3.4 that the target detection task is relatively easy, as all methods can achieve quite high scores. This also means that it is hard to improve any further on this task. In terms of precision and recall, SVM perform quite well on the precision metric, especially for the Review data set. While most deep learning methods focus more on enhancing recall. When considering both precision and recall, most deep learning methods are still better, as the F-score shows.

The second task is only evaluated on documents with ground-truth sentiments towards particular targets, with F-scores averaged over all targets and three sentiment classes: *positive*, *negative*, and *neutral*. We make several notes for this evaluation. (1) In order to achieve a high score in the second task, it is still important to classify correctly the presence of a target. (2) In general the scores for all methods in the second task are low, due to that the classifier might predict a target as *absent*, even though the ground-truth class can only be drawn from three sentiment classes. (3) It is possible for a method to outperform SVM on both tasks, while still obtain close results when two tasks are evaluated together. This results from our method of evaluation on the second task, where a document is included only when it expresses sentiment towards a particular target.

Based on the results from Table 3.5, we can see that the percentage of improvement over SVM is much higher than that of the first task. Intuitively, the sentiment task requires better modeling of the non-linear interaction between the target and the context, while for the target detection task, presence of certain signal words might be enough.

3.5.3 Training Time Analysis

In order to measure the training speed of each model, we train all deep learning methods on a server with a single TITAN X GPU. For SVM, it is trained on the same server with a 2.40 GHz CPU and 120 G RAM. All methods are trained sequentially without parallelization.

SVM can finish training in less than one hour, but its required training time increase linearly as the number of targets increases.

For all deep learning methods, the number of epochs required for training is in general very close, which is around 20 epochs averaged over all data sets.

Comparing the training time per epoch, ParaVec and CNN are much faster than other methods (less than 5 seconds / epoch). Despite the training efficiency, their effectiveness is a problem. When all targets share a single model, LSTM has a speed of 200 seconds/epoch, while standard memory networks have a speed of 150 seconds/epoch. Memory networks in many tasks, e.g., language modeling, are much faster than LSTM, due to the expensive recursive operation of LSTM. However in our problem setting, each target has to be forwarded one by one for every document, lowering the efficiency of memory networks. When individual targets are trained on separate LSTMs, LSTMs require far more training time (1000 seconds/epoch).

AttNet+ consumes 200 seconds per epoch. Comparing to standard memory networks, *AttNet+* produces some additional overhead by introducing the interaction between subtasks, and by adding a projection matrix. But this overhead is acceptable.

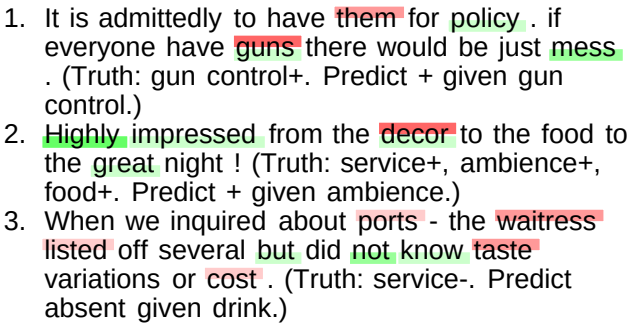
The efficiency of deep learning methods could be improved by parallelization. Since there are already many studies on this topic, which could increase the speed without sacrificing effectiveness, we do not go further into this direction.

Summary: empirical experiments demonstrated that the proposed deep memory

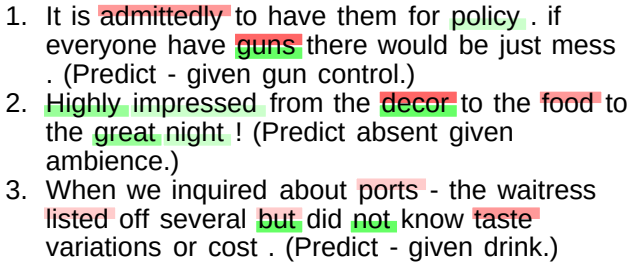
networks, AttNet+ and its variants outperforms conventional supervised learning methods. This is promising but perhaps not surprising given the success of deep learning in general. It is encouraging to notice that AttNets also improves the state-of-the-art deep learning architectures. This improvement is statistically significant, and can be observed for both subtasks and for attitude identification as a whole. The improvement of effectiveness does not compromise learning efficiency.

3.5.4 Visualization of attention

In order to better understand the behavior of our models, we compare the attention weights given by our model AttNets+ and the competing method Memnet.



(a) Attention given by AttNets+.



(b) Attention given by Memnet.

Figure 3.3: Visualization of learned attention. Red patches highlighting the top half of the text indicate model's attention weight in the target detection task, while green ones highlighting the bottom half show attention in the polarity classification task. Darker colors indicate higher attentions. *Truth: service+* means that the ground-truth sentiment towards *service* is positive, while *Predict + given ambience* gives the predicted positive sentiment given the query target *ambience*.

Figure 3.3 (a) and (b) list some examples of word attentions generated by different

models for the same set of sentences in the test set. In the first sentence, both *them* and *guns* are found as targets by AttNets+, while words like *mess*, and *policy* are found as sentiment words. Though Memnet correctly identifies the existence of the attitude towards gun control, it fails to find important words to classify the polarity of sentiment. This suggests the importance of interleaving the two tasks – successfully identifying mentioned targets could offer clues about the finding of sentiment words for the second task.

The second sentence is from a review of a restaurant, when *ambience* is used as the query target. We can see that the target detection module of AttNets+ captures the word *decor*, which signals the presence of the target *ambience*. The polarity classification module then focuses on extracting sentiment words associated with the target. However for the baseline Memnet, it captures both *decor* and *food* in the first task, mistakenly considering all sentiments are only describing food other than the ambience. Consequently, it judges that there is no sentiment towards ambience. This example shows us the benefit of using the projection matrices to consider the interaction and distinction between targets. Otherwise the model might easily confuse to which entity the sentiments are expressed.

From the third sentence, we can see how our model AttNets+ determines that the query target *drink* does not exist. The first module highlights words like ports (wine name), waitress, and the second module extracts negative sentiments *but not know*, which is usually used to describe people, rather than drink. Memnet almost has the same attention distribution as AttNets+, but still fails to produce the correct prediction. Similar to the second case, projection matrices are important for models to figure out the common phrases used to describe different set of entities.

3.6 Conclusion

Attitude identification, a key problem of modern natural language processing, is concerned with detecting one or more target entities from text and then classifying the sentiment polarity towards them. This problem is conventionally approached by separately solving the two subtasks and usually separately treating each target, which fails to leverage the interplay between the two subtasks and the interaction among the target entities. Our study demonstrates that modeling these interactions in a carefully designed, end-to-end deep memory network significantly improves the accuracy of the two subtasks, target detection and polarity classification, and attitude identification as a whole. Empirical experiments prove that this novel model outperforms models that do not consider the interactions between the two subtasks or among the targets, including conventional methods and the state-of-the-art deep learning models.

This work opens the exploration of interactions among subtasks and among contexts (in our case, targets) for sentiment analysis using an end-to-end deep learning architecture. Such an approach can be easily extended to handle other related problems in this domain, such as opinion summarization, multi-aspect sentiment analysis, and emotion classification. Designing specific network architecture to model deeper dependencies among targets is another intriguing future direction.

Table 3.3: Performance of competing methods: AttNets significantly improves existing methods; AttNet+ achieves top performance.

(a) Tweets

Method	F-score	AUC	Precision	Recall
SVM-sep-ind	59.93	69.20	68.70	55.69
SVM-sgl-ind	57.44^{***}	66.64^{***}	**69.87**	52.45^{***}
SVM-cmb-ind	57.09^{***}	66.46^{***}	69.84	52.25^{***}
ParaVec-sep-all	53.17^{***}	62.88^{***}	56.75^{***}	48.29^{***}
ParaVec-sgl-all	54.15^{***}	63.41^{***}	57.52^{***}	48.76^{***}
CNN-sep-all	58.05^{*}	70.10	62.43^{***}	56.19
CNN-sgl-all	58.69	70.71^{*}	61.83^{***}	56.64
BiLSTM-sep-all	61.16	71.26^{**}	63.45^{*}	59.87^{***}
BiLSTM-sgl-all	60.86	71.02^{**}	62.58^{*}	59.61^{***}
BiLSTM-sep-ind	59.49	71.92^{**}	61.44^{***}	57.86
MultiBiLSTM-sep-all	60.53	71.51^{**}	64.81^{*}	57.76
MultiBiLSTM-sgl-all	60.59	71.32^{**}	64.27^{*}	57.97
MultiBiLSTM-sep-ind	59.71	71.16^{**}	63.72^{*}	57.94
Memnet-sep-all	59.44	71.68^{**}	63.22^{*}	59.80^{***}
Memnet-sgl-all	60.69	71.80^{**}	63.48^{*}	59.97^{***}
Proposed methods				
AttNet-all	$63.42^{**}_{\diamond\diamond\diamond}$	$72.94^{***}_{\diamond}$	$68.78_{\diamond\diamond\diamond}$	60.57^{***}
AttNet-ind	$63.09^{**}_{\diamond\diamond\diamond}$	72.73^{***}	$68.33_{\diamond\diamond\diamond}$	59.68^{**}
AttNet+	$\mathbf{64.62}^{***}_{\triangledown}$	$\mathbf{74.76}^{***}_{\triangledown\triangledown}$	68.40	$\mathbf{62.09}^{***}_{\triangledown\triangledown}$

(b) Review

Method	F-score	AUC	Precision	Recall
SVM-sep-ind	38.43	57.99	**51.22**	36.83
SVM-sgl-ind	36.06^{**}	56.84^{**}	50.79	34.07^{**}
SVM-cmb-ind	35.71^{***}	56.61^{***}	50.73	33.78^{***}
ParaVec-sep-all	34.02^{***}	55.26^{***}	38.04^{***}	30.47^{***}
ParaVec-sgl-all	34.26^{***}	55.31^{***}	38.26^{***}	30.89^{***}
CNN-sep-all	57.55	43.73^{***}	33.24^{**}	57.38
CNN-sgl-all	35.45^{***}	56.29^{*}	44.65^{***}	32.83^{**}
BiLSTM-sep-all	40.78^{*}	61.01^{***}	42.54^{***}	39.01^{**}
BiLSTM-sgl-all	39.68	60.84^{**}	41.88^{***}	38.81^{*}
BiLSTM-sep-ind	40.42	62.25^{***}	42.68^{***}	39.78^{**}
MultiBiLSTM-sep-all	40.47	60.71^{**}	44.89^{***}	37.67^{*}
MultiBiLSTM-sgl-all	39.38	59.68^{*}	43.22^{***}	37.92^{*}
MultiBiLSTM-sep-ind	40.81	61.27^{**}	44.76^{***}	38.02^{*}
Memnet-sep-all	41.75^{**}	61.82^{***}	45.61^{***}	39.25^{**}
Memnet-sgl-all	41.65^{**}	61.53^{***}	45.23^{***}	39.13^{**}
Proposed methods				
AttNet-all	$63.18^{***}_{\diamond\diamond}$	$47.89^{**}_{\diamond\diamond}$	$42.77^{***}_{\diamond\diamond}$	$64.23^{***}_{\diamond\diamond\diamond}$
AttNet-ind	$44.15^{***}_{\diamond\diamond}$	$63.53^{***}_{\diamond\diamond}$	$50.02_{\diamond\diamond\diamond}$	40.58^{**}
AttNet+	$\mathbf{45.93}^{***}_{\triangledown}$	$\mathbf{65.58}^{***}_{\triangledown\triangledown}$	50.34	$\mathbf{44.95}^{***}_{\triangledown\triangledown}$

(c) Debates

Method	F-score	AUC	Precision	Recall
SVM-sep-ind	58.30	72.10	64.48	57.81
SVM-sgl-ind	59.75	72.25	66.39	57.67
SVM-cmb-ind	59.86	71.68	66.28	56.48
ParaVec-sep-all	56.32^{**}	68.12^{***}	59.09^{***}	49.41^{***}
ParaVec-sgl-all	55.35^{***}	67.48^{***}	59.46^{***}	49.82^{***}
CNN-sep-all	57.38	70.70^{**}	61.81^{**}	52.94^{***}
CNN-sgl-all	56.23^{**}	69.92^{**}	60.75^{**}	52.29^{***}
BiLSTM-sep-all	59.83^{*}	71.91	65.94^{*}	57.65
BiLSTM-sgl-all	58.66	72.01	64.87	57.89
BiLSTM-sep-ind	58.75	72.83	64.73	57.95
MultiBiLSTM-sep-all	59.24^{*}	72.24	64.75	58.43
MultiBiLSTM-sgl-all	58.98	71.18	63.46	57.26
MultiBiLSTM-sep-ind	58.36	72.93	64.15	57.14
Memnet-sep-all	60.42^{**}	73.84^{*}	65.37	58.92^{*}
Memnet-sgl-all	59.67^{*}	73.31^{**}	64.27	58.83
Proposed methods				
AttNet-all	$64.23^{***}_{\diamond\diamond\diamond}$	$76.13^{***}_{\diamond\diamond}$	$67.19^{**}_{\diamond\diamond}$	$62.17^{**}_{\diamond\diamond\diamond}$
AttNet-ind	$65.01^{***}_{\diamond\diamond\diamond}$	$76.35^{***}_{\diamond\diamond}$	$70.08^{***}_{\diamond\diamond\diamond}$	$60.70^{*}_{\diamond}$
AttNet+	$\mathbf{67.68}^{***}_{\triangledown\triangledown}$	$\mathbf{78.48}^{***}_{\triangledown\triangledown\triangledown}$	$\mathbf{74.55}^{***}_{\triangledown\triangledown\triangledown}$	$\mathbf{66.31}^{***}_{\triangledown\triangledown}$

*(**,***) indicate that one method is statistically significantly better or worse than *SVM-sep-ind* (which is in general the best configuration among all SVM models) according to t-test [127] at the significance level of 0.05(0.01,0.001). $\diamond\diamond(\diamond\diamond\diamond)$ indicate *AttNet* outperforms the better one between *Memnet-sep-all* and *Memnet-sgl-all* at the significance level of 0.01(0.001). $\triangledown\triangledown(\triangledown\triangledown\triangledown)$ indicate *AttNet+* outperforms the better one between *AttNet-all* and *AttNet-ind* at the significance level of 0.01(0.001).

Table 3.4: Performance on target detection for *-sep* models.

	Tweets	Review	Debates
SVM	79.74 **89.12**,75.00	67.84 **84.13**,63.52	84.59 91.47,81.00
ParaVec	75.63^{**} 82.67^{***},71.76^{***}	63.74^{***} 67.41^{***},57.03^{***}	76.16^{***} 80.32^{***},71.80^{***}
CNN	80.62 87.57^{**},77.41^{*}	65.34^{**} 73.23^{***},59.16^{**}	80.21^{**} $\mathbf{93.26^{*}}$,73.48^{**}
BiLSTM	81.50^{**} 86.48^{*},81.68^{**}	70.41^{**} 76.39^{***},69.11^{***}	85.05 92.03,82.05
MultiBiLSTM	81.53^{**} 87.69,78.82^{**}	69.26^{**} 75.38^{***},68.40^{***}	85.33 92.82,83.77^{*}
Memnet	81.29^{**} 87.82,79.36^{**}	70.71^{**} 75.52^{***},68.59^{***}	86.29^{*} 92.31,83.26
AttNet-all	82.58^{**} 88.78,79.05^{**}	71.84^{***} 75.85^{***},69.99^{***}	$89.02^{***}_{\diamond\diamond}$ 92.24,$86.47^{***}_{\diamond\diamond}$
AttNet-ind	$82.74^{**}_{\diamond}$ 88.29, 79.82^{**}	71.95^{***} $82.67_{\diamond\diamond\diamond}$, 66.19^{**}	$88.89^{**}_{\diamond\diamond}$ 92.35, 82.22
AttNet+	$\mathbf{84.89^{***}_{\triangledown\triangledown}}$ 88.76,$\mathbf{82.24^{***}_{\triangledown\triangledown}}$	$\mathbf{72.59^{***}}$ 76.38^{***},$\mathbf{71.09^{***}_{\triangledown}}$	$\mathbf{89.35^{***}}$ 92.05,$\mathbf{87.12^{***}}$

The first row of each method shows F-score, followed by precision and recall on the second row.

Table 3.5: Performance on polarity classification for *-sep* models.

	Tweets	Review	Debates
SVM	44.45 64.50,34.28	21.37 53.25,14.66	42.52 57.37,38.29
ParaVec	39.20^{***} 56.91^{**},26.46^{***}	17.69^{***} 31.05^{***},9.25^{***}	30.59^{***} 56.15,20.88^{***}
CNN	42.95 58.20^{**},35.71	19.42^{*} 39.91^{**},11.34^{**}	35.15^{***} 49.82^{**},25.40^{***}
BiLSTM	46.18^{**} 61.35^{*},41.79^{***}	25.25^{**} 41.06^{***},19.40^{**}	42.69 54.58,35.59
MultiBiLSTM	46.26^{**} 60.56^{**},39.36^{**}	24.06^{**} 47.15^{**},17.52^{**}	41.87 49.69^{*},37.26
Memnet	47.81^{**} 61.20^{**},40.52^{**}	25.47^{**} 46.34^{**},19.81^{**}	44.61 54.40^{*},38.14
AttNet-all	$50.91^{***}_{\diamond\diamond}$ $\mathbf{66.39}_{\diamond\diamond\diamond}$,$42.00^{***}_{\diamond}$	$32.43^{***}_{\diamond\diamond\diamond}$ $55.92_{\diamond\diamond\diamond}$,$24.43^{***}_{\diamond\diamond}$	$50.46^{***}_{\diamond\diamond\diamond}$ $60.72^{*}_{\diamond\diamond\diamond}$,$44.02^{**}_{\diamond\diamond}$
AttNet-ind	$49.16^{**}_{\diamond}$ $64.01_{\diamond\diamond}$, 41.74^{***}	$32.79^{***}_{\diamond\diamond\diamond}$ $56.15^{**}_{\diamond\diamond}$, $23.17^{***}_{\diamond\diamond}$	$51.88^{***}_{\diamond\diamond\diamond}$ $62.70^{**}_{\diamond\diamond\diamond}$, 40.52
AttNet+	$\mathbf{52.23^{***}_{\triangledown\triangledown}}$ 65.54,$\mathbf{44.57^{***}_{\triangledown\triangledown}}$	$\mathbf{35.34^{***}_{\triangledown\triangledown}}$ $\mathbf{59.99^{***}_{\triangledown\triangledown}}$,$\mathbf{27.32^{***}_{\triangledown\triangledown}}$	$\mathbf{55.93^{***}_{\triangledown\triangledown\triangledown}}$ $\mathbf{71.53^{**}_{\triangledown\triangledown}}$,$\mathbf{50.25^{***}_{\triangledown\triangledown}}$

The first row of each method shows F-score, followed by precision and recall on the second row.

CHAPTER IV

DeepGraph: Graph Structure Predicts Egonet Growth

The topological (or graph) structure of one's ego network is known to be predictive of multiple dynamic properties of the ego center. For instance, a researcher's collaboration network is predictive of her future h-index. Conventionally, a graph structure is represented using an adjacency matrix or a set of hand-crafted structural features. These representations either fail to highlight local and global properties of the graph or suffer from a severe loss of structural information. There lacks an effective graph representation, on which hinges the realization of the predictive power of network structures.

In this study, we propose to learn the representation of the topological structure of a egonet through a deep learning model. This end-to-end prediction model, named DeepGraph, takes as input the raw adjacency matrix of an egonet and outputs a prediction of the growth of the network, e.g., the size of the network. The adjacency matrix is first represented using a graph descriptor based on the heat kernel signature, which is then passed through a multi-column, multi-resolution convolutional neural network. Extensive experiments on four large collections of real-world networks demonstrate that the proposed prediction model significantly improves the effectiveness of existing methods, including linear or nonlinear regressors that use

hand-crafted features, graph kernels, and competing deep learning methods.

4.1 Introduction

Today we are surrounded by real-world networks of people, information, and technology. These heterogeneous, large scale, and fast evolving networks have provided a new perspective of scientific research, which has resulted in a rapid development of new theories, algorithms, and applications.

How to model and predict the dynamic properties of social or information networks has received considerable attention recently [102, 125, 1, 88, 55, 108, 19]. In the present work, we focus specifically on k-hop egonets, which are composed of a "ego" node, and its k-hop neighbors. Many interesting properties could be studied from these ego networks, including the size of the network, metrics of individual nodes or structures (e.g., degree or diameter), or even external properties that are not directly observed from the network structure (e.g., prestige, productivity or revenue of the ego center). All these properties change over time, and their dynamics can be generally referred to as the *growth* of a ego network[1]. Indeed, the prestige of an individual node grows with the size of its egonet. Accurate prediction of network growth has many valuable applications. For example, predicting the growth of paper's citation networks helps scientists to identify promising research directions; predicting the growth of Facebook user's friendship networks helps social network vendors optimize their marketing strategies.

Taking a typical data mining perspective, most existing methods extract features from both the network itself and any external information sources available. A function is learned that takes these features as input and outputs a predicted value of the network property in the future [1]. From many explorations on different genres

[1]The *growth* refers to both the increment and decrement of the dynamic properties of the ego networks, i.e., positive or negative *growth*.

of networks, there has been a consensus in literature that features extracted from the topological structure of the network (a.k.a., the *graph*) are generally very informative in these prediction tasks [1, 19]. As a comparison, other types of information, e.g., content or demographics, are only useful in certain scenarios. For example, the content of a hashtag is predictive to its diffusion [126] and homophily (e.g., similar demongraphics) is predictive to the growth of friendship networks [18], but these effects are not generalizable to other networks and other dynamic properties. In this study, we focus on investigating the predictive power of the *graph structure* of a ego network on its growth.

Existing structural features are typically hand-crafted based on theoretical and empirical findings in the social network literature. For example, open triads with two strong ties are likely to be closed in the near future [29]; dense communities are resistant to novel information and they grow slower than others [40]; nodes spanning structural holes are likely to gain social capital and experience a rapid growth of its prestige and other properties [14]. Features such as network density, clustering coefficients, triadic profiles, and structural holes are therefore designed to implement these intuitions and represent the graph structure.

Despite the success in predicting network growth, there are observable issues of representing the topological structure of a network using these hand-crafted features. Some of them only describe a global property of the network, such as network density or degree distribution; some of them provide a fine-grained description of local structures but fail to capture global information, such as triads and other substructures; others lie between the two extremes, such as structural holes. None of these features is able to fully represent both the local and the global structure of a graph and the complex interaction between local and global properties. On the other hand, these heuristic features usually have a limited characterization power for networks, as many networks may share the same feature representation. For example, most real-world

networks at scale may have a similar (power-law) degree distribution, and two very different networks may happen to have the same ratio of closed triangles. Taking a machine-learning point of view, we are intrigued by the following questions: *what is a suitable representation of network structure and how effective is such a representation when used to predict network growth?*

Our answers to the two questions are inspired by the recent developments in deep learning and graph representation. We introduce a graph descriptor that is based on the Heat Kernel Signature (HKS) [100], which serves as a universal low-level representation of the topological structures of networks. HKS has been successfully employed in representing the surface of 3D objects [31, 121]. By modeling the amount of heat flow over nodes of a network over time, HKS successfully stores both the global and the local structural information of the entire network. Using a histogram to describe the probability distribution of heat values at a series of time points [31, 121], isomorphic networks (networks with the same topological structure) can be mapped to a unique representation at little loss of structural information. However, unlike 3D objects which are composed of polygon meshes, the structures of networks vary in shape, size, and complex local structures. To address this issue, some computations of HKS need to be approximated carefully. Inspired by the semantics of the HKS-based graph descriptors, we propose a multicolumn, multiresolution neural network that learns latent hierarchical representations of graphs on top of the HKS-based graph descriptor. The proposed deep neural network, named DeepGraph, predicts network growth in an end-to-end process.

We conduct extensive experiments to evaluate the effectiveness of DeepGraph. Different growing properties are predicted for four genres of real-world ego networks. Empirical results show that our method outperforms baseline approaches that use alternative graph representations, hand-crafted features, or existing deep learning architectures. High-level representations learned by DeepGraph well connect to existing

findings in the social network literature.

4.2 Related work

Predicting the growth of networks or the evolution of certain properties of networks has been widely studied. People attempt to predict the dynamics of various network metrics or aggregated activities in a network, e.g., the number of up-votes on Digg stories [102], the number of newly infected nodes in diffusion [125], the growth of a community [1, 88], or the dynamics of a cascade [55, 108, 19]. In these studies, a set of problem-specific features are usually manually designed based on the network structure, textual content, user demographics, historical statistics, and other sources of information. Among them, the features extracted from the network structure are both effective in individual tasks and robust across different tasks. In this work, we limit our focus on information purely from the network structure.

Finding a suitable representation of the topological structure of a network has always been a critical preliminary step of network analysis. Conventionally, a network is represented as an adjacency matrix or a sparse list of edges. However, these lossless representations do not effectively present the structural characteristics of the network. Moreover, they are sensitive to the manipulation of node orders, making networks with the same topological structure mapped to different representations. Other approaches represent the network structure with a series of network metrics and/or a set of structural patterns (e.g., triads [53], quads [110], or meta-paths [101]). Arbitrary higher-order substructures can be included, such as communities and structural holes. These bag-of-substructures better capture local patterns of the network structure.The major problem of this approach is that it is computationally infeasible to enumerate high-order substructures, and low-level substructures have limited representation power of the global structure of the network. As a result, many different networks may share the same or similar bag-of-substructures.

In graph classification, a myriad of graph kernel methods are proposed which compute pairwise similarities between graphs [49, 2, 94, 93]. For example, *graphlets* [93, 109] computes the graph similarity based on the distribution of induced, non-isomorphic sub-graphs. Some other graph kernels integrate frequent graph mining into the model training process [90, 86]. Graph kernels provide an indirect representation of networks so that similar structured networks yield a high value through the graph kernel function. The burden of graph kernels is the design of effective kernels. In the paper, we compare existing graph kernels to highlight the flexibility of our model.

Recently, researchers have started to apply deep learning to network structure representation learning. Several proposals have been made to learn a low-dimensional vector representation of individual nodes by considering their neighborhood [106, 81, 38]. Deep learning techniques have also improved graph kernels for graph structure learning[122, 123, 75]. Recently, Niepert et al. [78] applied convolution over receptive fields constructed by sequence of neighboring nodes. These methods focus only on the local structure of a graph and graph kernels require expensive pairwise comparisons. In the paper, we compare our model to these alternative deep learning approaches and show the performance advantage of our model.

Heat kernels have been studied for the task of graph clustering [3], graph partitioning [30], and modeling social network marketing processes [65]. These applications rely on the raw output of heat kernels for a variety of tasks, rather than developing a *signature*, nor do they abstract graph representations base on heat kernels. In the community of computer vision, Heat kernel signature has been successfully used to model 3D objects [100, 31, 121], whose surfaces are defined by polygon meshes, a network composed of simple convex polygons. In contrast, real-world networks are consist of various shapes, sizes, and local structures. How to represent arbitary networks with heat kernel signatures and how to predict network growth using such a signature remain a challenging question to be studied.

4.3 DeepGraph for Network Growth Prediction

We propose a unified predictive neural network model to learn graph structure representation for network growth prediction problem. The proposed predictive model, named DeepGraph, combines heat kernel signature and deep neural networks. Below we describe the two key components of our model, (1) a heat kernel signature based graph descriptor and (2) a deep multi-column, multi-resolution convolutional neural network, in turn, following a brief definition of the network growth prediction problem.

4.3.1 Problem Formulation and Notations

Given a real-world network snapshot at time t, denote its graph structure as $\mathcal{G}^{(t)} = (V, E)$, with a set of nodes V and a set of edges E. A node $i \in V$ represents an entity (e.g., an actor in a social network or a paper in a citation network), an edge $(i, j) \in E$ represents a relationship (e.g., friendship, citation, or influence) between node i and node j. An adjacency matrix $\mathbf{W} \in \mathbb{R}^{|V|\times|V|}$ encodes the topological structure of the graph $\mathcal{G}$. In this work, we consider the binary adjacency matrix. Its element w_{ij} is 1 if and only if $(i, j) \in E$ and 0 otherwise.

A network property is a function that maps a graph structure $\mathcal{G}^{(t)}$ to a property value $y^{(t)} \in \mathbb{R}$. For example, a network property could be the number of friends given a user's Facebook ego-network. A network growth predictor is a function that maps a graph structure $\mathcal{G}^{(t)}$ to a property value $y^{(t')}$ at time t', satisfying $t' > t$. For example, a network growth predictor could map a user's Facebook ego-network of this year to the number of friends next year.

The network growth prediction can be naturally formulated as a supervised learning problem. Specifically, the problem is to derive a network growth predictor f given a training set of tuples $\{(\mathcal{G}_i^{(t_i)}, y_i^{(t'_i)})\}_{i=1}^{M}$ to minimize the prediction error over a test set of tuples $\{(\mathcal{G}_j^{(t_j)}, y_j^{(t'_j)})\}_{j=1}^{N}$ satisfying $\forall i\ t'_i > t_i$, $\forall j\ t'_j > t_j$, $\min_j(t_j) > \max_i(t_i)$,

and $\min_j(t'_j) > \max_i(t'_i)$. The time ordering constraints highlights the practical motivation that we are interested in using historical data to predict future properties of current networks.[2] To apply a machine learning algorithm, it is critical to first represent the graph $\mathcal{G}^{(t)}$ computationally, such as using a vector of features.

4.3.2 Heat Kernel Signature based Graph Descriptors

The motivation in adopting Heat Kernel Signature (HKS) is its theoretical proven properties in representing graphs: HKS is an **intrinsic** and **informative** representation for graphs [100]. **Intrinsicness** means that isomorphic graphs map to the same HKS representation, and **informativeness** means if two graphs have the same HKS representation, then they must be isomorphic graphs. Our HKS-based graph descriptor builds on the theoretical properties of HKS and further provides universal representations for graph with different sizes in network growth prediction.

Heat kernel function. Formally, the heat kernel $h_z(i, j)$, a function of two nodes i, j at any given diffusion step z, denotes the amount of aggregated heat flow through all edges among two nodes after diffusion step z[3]. In computer vision, graphs are stored as meshed networks and heat kernels are computed by finding eigenfunctions of the Laplace-Beltrami operator [100]. However, most real-world networks are not meshed networks. Instead, we use eigenfunction expansion of a graph Laplacian [100, 3] to compute the heat kernel for information networks. Given a graph $\mathcal{G} = (V, E, W)$, the graph Laplacian is defined as: $\mathbf{L} = \mathbf{D} - \mathbf{W}$, where D is a diagonal degree matrix with diagonal entries being the summation of rows of W: $D_{ii} = \sum_j w_{ij}$. The normalized Laplacian of the graph is given by $\mathbf{L_N} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$.

[2] In practice, researchers focus on a specially case of the network growth prediction problem with the equal interval increment constraint, $t'_j - t_j = t'_i - t_i = C > 0$ [55, 108].

[3] The diffusion is simulated for a given graph snapshot. The heat kernel computation does not require graph snapshot at other timestamps. The diffusion step z should not be confused with the network timestamp t.

The heat kernel is then defined as

$$h_z(i,j) = \sum_{k=1}^{|V|} e^{-\lambda_k z} \phi_k(i) \phi_k(j) \tag{4.1}$$

where λ_k is the k-th eigenvalue of the normalized Laplacian $\mathbf{L_N}$ and ϕ_k is the k-th eigenfunction s.t. $\sum_i |\phi_k(i)|^2 = 1$. Note that the eigenvalues might be unreal in the case of directed graphs. There has been studies on how to tackle this problem [21]. In this work, for simplicity, we convert directed graphs to undirected ones by applying $\mathbf{W} = (\mathbf{W} + \mathbf{W}^\intercal)/2$.

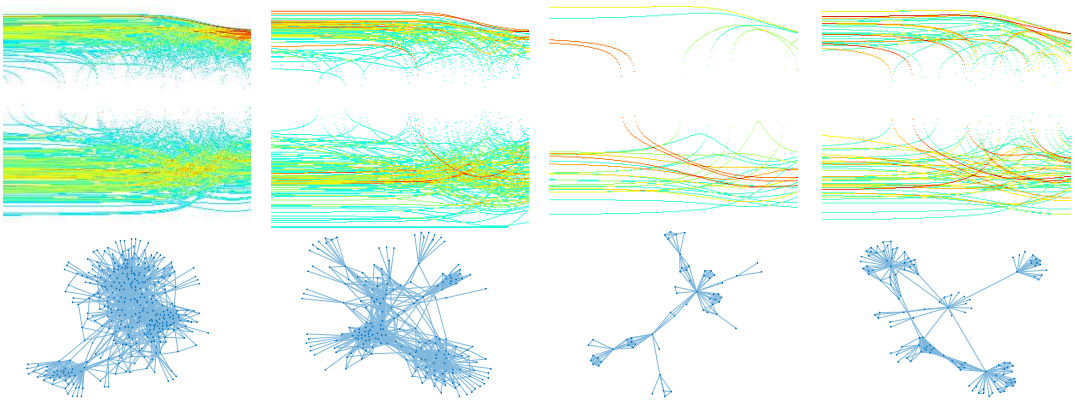
Heat kernel signature. Heat kernel signature was introduced to mitigate the computation bottleneck of using heat kernel functions in representing graphs. Both heat kernel and heat kernel signature are proven to be intrinsic and stable against noises. However, the computation complexity of using heat kernel as a point signature, $\{k_t(v,.)\}_{t>0}$, is defined on the product of temporal and spatial domain. Heat kernel signature simplifies the computation by considering only a subset of product of temporal and spatial domain while keeping as much information as possible. Specifically, heat kernel signature reduces the computation complexity by only requiring $h_z(v,v)$ over a finite set of N diffusion steps $z \in \{z_1, z_2, ..., z_N\}$ for $\forall v \in V$ without losing the intrinsic and informative properties.

Formally, a heat kernel signature (HKS) is a matrix $\mathbf{H} \in \mathbb{R}^{|V| \times N}$ satisfying

$$H_{ij} = h_{z_j}(i,i) \tag{4.2}$$

These time points are sampled with equal difference after logarithm [100], such that $\log z_n - \log z_{n-1} = \log z_{n+1} - \log z_n$.

Graph descriptor. Some graphs might have thousands of nodes, causing difficulties for deep neural networks when they are fed with input that is dependent on $|V|$. Therefore, the practical issues in combining HKS and deep neural networks are that



(a) (b) (c) (d)

Figure 4.1: Examples of HKS-based graph descriptors. The first row shows our graph descriptors for graphs in the second row. Figure (a) and (b) are subnetworks from Facebook [111]. Figure (c) and (d) are some authors' collaboration networks built from ACL Anthology [28].

we need a global vertex indexing to guarantee the uniqueness and that the representation is independent of $|V|$. To this end, we further process heat kernel signature $\mathbf{H}$ into a universal representation independent of $|V|$ using a histogram conversion. Specifically, we use histograms to estimate the distribution of HKS values in each column[4]. By denoting N_B the number of bins used in the histogram, we obtain a universal descriptor $\mathbf{S} \in \mathbb{R}^{N_B \times N}$. Specifically, S_{bj} counts the number of nodes falling into b-th bin at j-th diffusion step. Unlike HKS, the new descriptor is independent of vertex ordering and vertex number. We call this final matrix *graph descriptor*, $\mathbf{S}(\mathcal{G})$, as it is adapted to describe information networks. Figure 4.1 shows four examples of our graph descriptors for real world graph structures.

Graph descriptor vs. adjacency matrix. We have described the process in converting an adjacency matrix into our graph descriptor, which is then passed through a deep neural network for further feature extraction. All computation in this process is to obtain a more effective low-level representation of the topological structure information than the original adjacency matrix.

First, isometric graphs could be represented by many different adjacency matrices,

[4]The bin ranges are aligned column-wise on the training data.

while our graph descriptor would provide a unique representation for those isometric graphs. The unique representation simplifies the neural network structures for network growth prediction.

Second, our graph descriptor provides similar representations for graphs with similar structures. The similarity of graphs is less preserved in adjacency matrix representation. Such information loss could cause great burden for deep neural networks in growth prediction tasks.

Third, our graph descriptor is a universal graph structure representation which does not depend on vertex ordering or the number of vertexes, while the adjacency matrix is not.

Time complexity. The major overhead of computing graph descriptors lies in the calculation of eigenvectors. The time complexity of computing eigenvectors is $\mathcal{O}(K|V|^2)$ where K is the number of eigenvectors. Our graph descriptors finish in acceptable time frame for real world network data. The data description and time complexity analysis are in Section 4.4.

Semantics of graph descriptor. The rows and columns in our graph descriptor reflect the network topology from different perspectives. The rows express the heat density dynamics over diffusion steps, and the columns capture the static heat density patterns for a given diffusion step. Successive rows or columns express higher-order properties of the topology structure information. Such representational properties motivate the adoption of row-wise and column-wise convolution networks for feature learning.

4.3.3 Deep Graph Descriptor

As information abounds in the raw representation extracted by the HKS-based graph descriptor, applying a simple regressor, e.g., linear regression, could fail to fully extract useful information from it. In contrast, deep neural networks (DNN) have

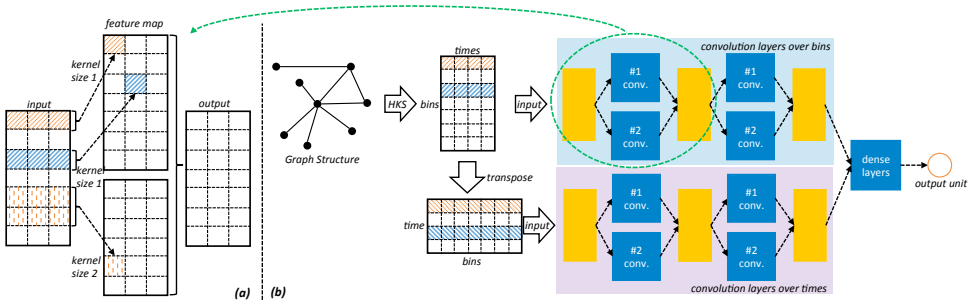


Figure 4.2: (a) An example of the multiresolution convolution unit with two kernel sizes. (b) An example of the multicolumn, multiresolution deep neural network model for network growth prediction with two convolution layers.

achieved tremendous success in learning latent representations from raw inputs in a compositional hierarchy. Combining DNN and HKS-based graph descriptor together thus offers an opportunity to address the graph structure representation challenges in predicting network growth. Inspired by the semantics of the graph descriptors, we propose a deep multicolumn, multiresolution convolutional neural networks for the network growth prediction task.

Multiresolution convolutions. Our model builds on the multiresolution 1-D convolution (MrConv) which maps an input matrix into a feature map matrix. Specifically, let $\mathbf{x}_i \in \mathbb{R}^k$ denote the i-th row of the input matrix. The input is then represented as $\mathbf{x}_{1:n} = \oplus_{i=1}^{n} \mathbf{x}_i$ where $\oplus$ is the concatenation operator and n is the number of rows. The 1-D convolution with a filter size m apply a filter $\mathbf{w} \in \mathbb{R}^{mk}$ to each possible window of m rows to produce a new feature vector $\mathbf{c} = [c_1, c_2, ..., c_{n-m+1}]$. The feature c_i is generated from a window of m rows $\mathbf{x}_{i:i+m-1}$ by $c_i = g(\mathbf{w}^\intercal \mathbf{x}_{i:i+m-1} + b)$, where $b \in \mathbb{R}$ is a bias term and g is a non-linear function such as a hyperbolic tangent function or a rectified non-linearity function.

We have described the process by which *one* feature vector $\mathbf{c}$ is extracted from *one* filter. Our multiresolution convolution (MrConv) layer uses multiple filters with varying filter sizes to obtain multiple resolution features. Specifically, one MrConv layer has l different convolution filter sizes $\{m_1, m_2, ..., m_l\}$. The filter of size m

generates a corresponding feature vector $\mathbf{c}^{(m)}$. Feature vectors generated by different filter sizes are then concatenated into one vector $\mathbf{c}^* = \oplus_{i=1}^{l} \mathbf{c}^{(m_i)}$. Moreover, we extend each filter size to have d different filters, where d is a hyperparameter. The final output feature map is a matrix $\mathbf{O}$ where each column is a feature vector $\mathbf{c}^*$ and there are d columns: $\mathbf{O} = \left(\mathbf{c}^{*1}, \mathbf{c}^{*2}, ..., \mathbf{c}^{*d}\right)$.

An example of our MrConv is shown in Figure 4.2(a). The example MrConv layer has two different filter sizes $\{1, 2\}$. Each filter size has three different filters, whose feature vectors form different columns in the final feature map. Multiple multiresolution convolution layers are stacked to form our model.

Multicolumn model. Inspired by the different semantics of rows and columns in the HKS-based graph descriptor, our model deploys a two network-column structure, as shown in Figure 4.2(b). One column uses multiresolution 1-D convolution (MrConv) operations over the graph descriptor bins and the other one uses MrConv over diffusion times. The two columns extract different features from the graph descriptors at multiple resolution scales. Intuitively, the first column extracts statistical features of the density dynamics in diffusion. The second column extracts features on static density pattern for different diffusion steps. Both kinds of features reflect the topology of the underlying graph structure, but explain the structure topology from different perspectives. A single column convolutional neural network can hardly extract such two kinds of features successfully.

The feature maps from the two columns are then concatenated and passed through multiple dense (i.e. fully-connected) layers with non-linear activation functions. The output from the multiple dense layers are then passed through a final linear fully-connected layer with only one output unit. The output unit $\hat{y}$ is thus the network growth prediction of our model.

4.3.4 End-to-End Training

Let McMrConv$(., \theta)$ denote the multicolumn multiresolution convolutional neural network with parameters θ. The final output of our neural network given a graph $\mathcal{G}_k$ is represented as: $\hat{y}_k = \text{McMrConv}(\mathbf{S}(\mathcal{G}_k), \theta)$. Given a training data set $\{(\mathcal{G}_k, y_k)\}_{k=1}^{K}$, the deep neural network is trained to minimize the average squared error: $\mathcal{L}(\theta) = \frac{1}{K} \sum_{k=1}^{K} \Big(\text{McMrConv}(\mathbf{S}(\mathcal{G}_k), \theta) - y_k\Big)^2$. The HKS-based graph descriptor and the deep neural network assembles DeepGraph, an end-to-end deep architecture to predict network growth based on graph structure.

4.4 Experiment setup

We compare our model with existing approaches on the network growth prediction problem. We then evaluate variants of our model for credit assignment.

4.4.1 Data sets

When selecting real-world data sets for evaluation, we consider both popularity and diversity of the application scenarios. The four data sets we choose include ego networks extracted from social networks, scientific collaboration networks, and entertainment networks. The statistics of these data sets are presented in Table 4.1. Please note that due to the diverse nature of the data sets and the various precision of timestamps available, it is hard to apply an unified time frame for all data sets. Viewed from another perspective, this helps us evaluate the flexibility and generality of our methods, verifying whether it can be applied to any length and granularity of time frames.

We follow the procedure described in [122] to construct ego-nets. The Facebook data set is collected from the New Orleans networks [111], where nodes are Facebook users and edges are friendships. We derive the snapshot of ego-networks for each user

Table 4.1: Statistics of the data sets.

	Dataset	Facebook	YouTube	AAN	IMDB
# graphs	train	12990	15258	8426	12500
	val	890	1283	713	1017
	test	2092	3273	1722	2407
Avg. nodes	train	399.9	147.6	271.4	197.3
	val	397.5	167.3	302.4	208
	test	436	165.8	402.4	216
Avg. edges	train	6800.8	1439.2	2079.8	7801.5
	val	6764.4	1626.1	2327.2	7847.7
	test	7499.2	1620.9	3321.8	7964.5
Avg. growth	train	3.6	9	1.2	1.3
	val	3.8	10.8	1.2	1.3
	test	3.4	9.3	1.2	1.3
Avg. scaled growth[1]	train	1.7	2.4	0.9	1
	val	1.8	2.2	0.9	1
	test	1.6	2.1	0.9	0.9
Graph time[2]	train	2007.6	2007.2	2009	2000
	val	2007.7	2007.3	2010	2001
	test	2007.8	2007.4	2011	2002
Growth time[3]	train	2007.10	2007.4	2010	2001
	val	2007.11	2007.5	2011	2002
	test	2007.12	2007.6	2012	2003
k-hop ego-net[4]	all	2	2	3	2

1. Avg. scaled growth scales label y to $\log_2(y+1)$ [55, 108].
2. Graph time of 2007.6 means the graph is built by taking the snapshot of Jun. 1, 2007.
3. Growth time of 2007.10 means the growth is computed between its corresponding *graph time* to Oct. 1, 2007. Graphs in train/val/test set do not overlap.
4. k-hop ego-net for AAN is set to 3, due to its small size when $k = 2$.

according to the timestamps listed in Table 4.1, which is used to predict the number of new friends this user made in the next four months.

As the YouTube [71] data set also describes user friendships, it follows the same setting as Facebook.

The AAN data set [28] is built upon scientific publications from the ACL Anthology[5], where nodes are authors and edges are collaboration. Each author's ego-nets are extracted to predict her h-index in the next year.

[5] `http://aclweb.org/anthology/`

IMDB is a movie co-star data set[6], where nodes are actors or actresses, and an edge is formed if they appear in the same movie. The ego-nets of each actor/actress is used to predict the number of new movies the actor/actress produced in the next year.

To examine whether we can truly predict future growth, we make sure of two important points: (1) the period to compute growth for test set is always later than that for training set; (2) one graph can only appear in one of the training, validation and test set. To this end, for each node in the global network, they are randomly assigned to the training/validation/test set with probability of 0.8/0.05/0.15. Based on which set they are in, their ego-nets and growth are computed according to the time listed in Table 4.1. If a node has not yet been created for the given time, it is simply removed.

We notice that the growth of all the ego networks in general follows a power-law distribution, where a large number of networks did not grow at all. Therefore we downsampled 50% graphs of each train/val/test set with zero growth (to the numbers shown in Table 4.1) and applied a logarithm transformation of the outcome variable (network growth), following [55, 108]. The network growth are scaled logarithmically for two reasons. First, baseline methods with linear regression are sensitive to extremely large outcomes. Second, when a network grows to a considerably large scale, we care more about its scale rather than the exact number.

4.4.2 Evaluation Metric

We use mean squared error (MSE) as our evaluation metric, which is a common choice for regression tasks. Specifically, denote $\hat{y}$ a prediction value, and y the ground truth value, then MSE $= \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$. As noted before, y in above equation is a scaled version of the original value y^o, that is $y = \log_2(y^o + 1)$.

[6] `http://www.imdb.com/`

4.4.3 Baseline methods

We compare DeepGraph with methods from two categories: feature-based methods used for network prediction tasks, and alternative graph representation methods.

Feature based. Many structural features have been designed for various network prediction tasks [1, 88, 108, 19]. We select from them those that could be generalized across data sets, including:

Frequencies of k-node substructures ($k \leq 4$)[110]. This counts the number of nodes ($k = 1$), edges ($k = 2$), triads (e.g., the number of closed and open triangles) and quads.

Other network properties: average degree, the length of the shortest path, edge density, the number of leaf nodes (nodes with degree 1), the number of leaf edges, the average closeness of all nodes, clustering coefficient, diameter, and the number of communities obtained by a community detection algorithm [11].

Graph kernels. Following [78], we compare with four state-of-the-art graph kernels: the shortest-path kernel (**SP**) [12], the random walk kernel (**RW**) [34], the graphlet count kernel (**GK**) [93], and the Weisfeiler-Lehman subtree kernel (**WL**) [94]. In our experiment, the RW kernel does not finish after 10 days for a single data set, so we exclude it for comparison. This exclusion is also observed for the same reason in [78, 123].

-linear and **-deep**. Feature based methods and graph kernels are usually trained on SVMs. We report linear regression instead, as SVM empirically generates poor results for our regression tasks. We append **-linear** to each method to indicate usage of linear regression. To obtain even stronger baselines, we apply deep learning to both feature vectors and graph kernels, indicated by **-deep**.

Smoothed graph kernels. Yanardag et al. [123] apply smoothing to graph kernels, which extends their method of deep graph kernels [122] by considering structural similarity between sub-structures. We report smoothed results only on deep neural

networks as it outperforms alternatives empirically.

PSCN, which applies convolutional neural networks (CNN) to locally connected regions from graphs [78], achieving better results over graph kernels on some of the classification data sets.

Hyper-parameters. All hyper-parameters are tuned to obtain the best results on validation set. For linear regression, we chose the L2-coefficient from $\{10^0, 10^{-1}, ..., 10^{-7}\}$. For neural network regression, the initial learning rate is selected from $\{0.1, 0.05, 0.01, ..., 10^{-4}\}$, the number of hidden layers from $\{1, 2, ..., 4\}$, and the hidden layer size from $\{32, 64, ..., 1024\}$. The size of the graphlets for GK is chosen from $\{3, 4\}$ (higher than 4 is extremely slow), the height parameter of WL from $\{2, 3, 4\}$, the discount parameter for smoothed graph kernels from $\{1, 0.8, ..., 0\}$. Following [78] for PSCN, the width is set to the average number of nodes, and the receptive field size is chosen between 5 and 10.

Notes. Please notice that in our experiments we are not identifying the nodes in the networks or using the information of the nodes outside the network itself. Of course, knowing the president of United States is in the network provides more confidence on its growth. We choose not to identify nodes because (1) this study focuses on investigating the predictive power of the topological structure of networks, and (2) in practice information about individual nodes may not be available for privacy reasons. For the same reasons, we do not include any information other than the network structure (e.g., content of tweets, or historical metrics of the network) in the prediction task, even though including more information may improve the prediction accuracy.

4.4.4 DeepGraph Model Parameters

Parameters included in HKS are set to default values across all data sets without further tuning. In Equation 4.2, we set $t_1 = 0.1$, $t_N = 25$, and $N = 64$. Number of

bins N_B is set to 64. To compute histograms, HKS values above $+1.2$ and below -1.2 standard deviation are respectively put to the first and last bins. Values in between are assigned to the remaining equally divided 62 bins.

We perform standard normalization for the histograms of graphs. Each histogram is preprocessed by pixel-wise normalization. We compute the mean and standard deviation for each pixel over the training data set. Then each pixel is normalized by subtracting the corresponding mean value and being divided by sd[7] .

We initialize the parameters of the neural networks using a Gaussian distribution with zero mean and unit standard deviation. An adaptive optimizer, Adam, is used to optimize the parameters of the neural networks. Default hyper-parameters of Adam are used [51].

Structure related hyper-parameters of DeepGraph is set to be the same across datasets. There are two multiresolution convolution layers for each network column, with number of filters 32 and 16. For each convolution layer, we apply three sizes of filters, which are 2, 4, and 6. TanH is used as the activation function. There are two fully connected layers both of size 256. Dropout is applied to the last two dense layers with probability of 0.5. Other learning parameters are listed in Table 4.2.

Table 4.2: Setup of hyper-parameters for DeepGraph.

	Facebook	YouTube	AAN	IMDB
L2-coefficient	1e-5	1e-5	0.005	1e-5
Init learning rate	0.005	0.01	5e-4	0.005

4.4.5 Variants of DeepGraph

To assign the credit of each key component in our DeepGraph model, we also experiment with some of its variants, by feeding our graph descriptor (**GD**) to a

[7] $\epsilon = 10^{-8}$ is added to the denominator to avoid numeric issues.

linear regressor (**GD-linear**), a standard convolutional neural network (**GD-CNN**), and a multilayer perceptron (**GD-MLP**). Hyper-parameters for these models are tuned similarly as baselines.

4.5 Experiment results

4.5.1 Overall performance

Table 4.3: Performance measured by MSE (the lower the better), where original label y is scaled to $\log_2(y+1)$.

Dataset	Facebook	YouTube	AAN	IMDB
Feature-deep	1.107	2.623	0.421	0.527
Feature-linear	1.116	2.633	0.439	0.525
GK-smooth	1.313^{***}	2.675^{***}	0.480^{***}	0.561^{**}
GK-deep	1.315^{***}	2.671^{***}	0.492^{***}	0.565^{**}
GK-linear	1.335^{***}	2.736^{***}	0.519^{**}	0.576^{***}
WL-smooth	1.158^{***}	2.659	0.434	0.536
WL-deep	1.165^{**}	2.654	0.437	0.532
WL-linear	1.331^{***}	2.702^{***}	0.445	0.596^{***}
SP-smooth	1.138	2.615	0.422	0.530
SP-deep	1.155^{**}	2.607	0.428	0.531
SP-linear	1.179^{***}	2.613	0.432	0.535
PSCN	1.117	2.534^{***}	0.425	0.528
Proposed methods				
GD-linear	1.174^{***}	2.750^{***}	0.587^{***}	0.583^{***}
GD-MLP	1.082^{*}	2.427^{***}	0.394^{***}	0.513^{*}
GD-CNN	1.087	2.429^{***}	0.391^{***}	0.512^{*}
DeepGraph	$\mathbf{1.068^{**}_{\triangledown}}$	$\mathbf{2.409^{***}_{\triangledown\triangledown}}$	$\mathbf{0.379^{***}}$	$\mathbf{0.508^{***}_{\triangledown}}$

"***(**)" means the result is significantly better or worse over *Features-dp* according to paired t-test test at level 0.01(0.1). "$\triangledown$" means DeepGraph-multi is better than the better one between GD-MLP and GD-CNN.

The overall performance of all competing methods across data sets are displayed in 4.3. We make the following observations. First, integrating graph descriptor with deep learning, our method DeepGraph outperforms all competing methods significantly. This empirically confirms that graph descriptor could preserve more informa-

tion of the network structure than bag-of-substructures, both globally and locally. In contrast, utilizing manually designed features could lead to loss of information.

GD-MLP and GD-CNN have already gain improvement over the strongest baseline on most of the data sets, while DeepGraph can further improve the performance by utilizing the semantics of HKS-based graph descriptor. This shows that we can indeed extract more useful features by applying column-wise and row-wise convolution over graph descriptors.

Comparing with GD-linear, which applies linear regression on top of the HKS-based graph descriptor, DeepGraph, GD-MLP, and GD-CNN performs significantly better. This indicates that the effectiveness of the HKS-based graph descriptor has to be utlized by a "deeper" model which explores the convolutions and non-linear transformations of the low-level representation.

Comparing feature based methods with other baselines, the former exhibit strong prediction power. Incorporating both local and global information, the hand-crafted features are very indicative of network growth, which is hard for automatic methods to compete.

When trained on deep networks, the performance of graph kernels could be improved over their linear version. Smoothing kernels can further bring in some improvement. By applying convolution over locally connected regions of the graphs, PSCN can beat many graph kernels on most data sets. These results are consistent with previous studies [78, 123].

4.5.2 Computational Cost of DeepGraph

Training of DeepGraph is very fast. The models are converged in less than 10 minutes on a Titan X GPU. The major overhead of DeepGraph is the computation of the HKS-based graph descriptors. We empirically measure the computation time for all data sets on a server with 2.40 GHz CPU and 120G RAM. The graphs in our

data sets have size as large as 5,000 nodes and 200,000 edges, which is enough for most network prediction problems [55, 88, 125]. The generation of graph descriptors takes an average of 0.86 hour per data set. In contrast, the strongest baseline, feature based method, takes 7.9 hours on average to generate all features. While the strongest graph kernel, SP, takes nearly 5 days.

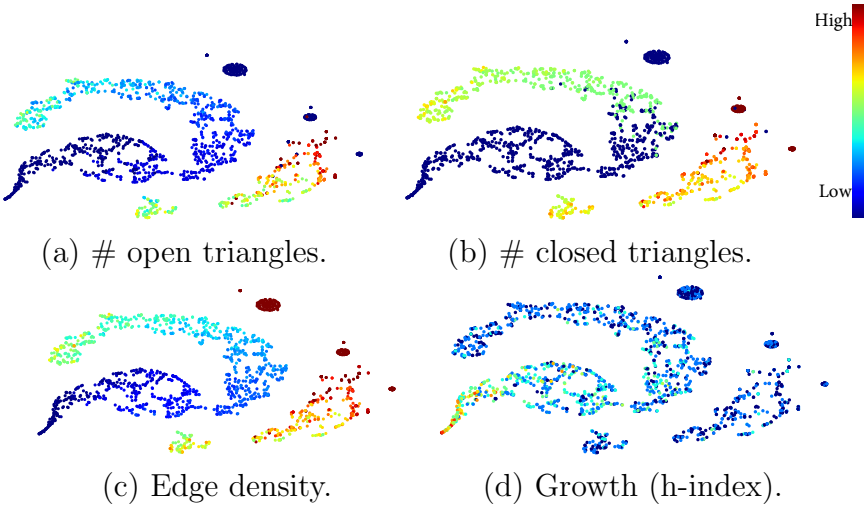
4.5.3 Feature Analysis

It has been shown empirically that DeepGraph could well abstract high-level features to represent graphs. It is intriguing to know whether these learned features correspond to well-known structural patterns in network literature. To this end, we select some of the network properties manually computed for the feature based method. Note that we work only on test set, as we care more about the prediction performance. These properties characterize either global or local aspects of networks, and are listed in Figure 4.3.

In order to examine whether the high-level representations learned by DeepGraph have captured these properties, we need a way to visualize the high-level representations and the above network properties. To do so, the feature vectors output by the last hidden layer of DeepGraph are fed to t-SNE [11], a dimensionality reduction algorithm for visualizing high-dimensional data sets. The t-SNE algorithm projects feature vectors into a 2-dimensional space, where similar vectors are projected closely. The visualizations of data set AAN are displayed in Figure 4.3. We obtain similar results on other data sets, which are omitted to conserve space.

To connect the hand-crafted structural properties with the learned high-level features, we color individual graphs by the values of these properties (e.g., network density). Patterns on the distribution of colors could suggest a connection between learned features and the network property.

Some observations can be made from Figure 4.3. First, as the number of open and



(a) # open triangles. (b) # closed triangles.

(c) Edge density. (d) Growth (h-index).

Figure 4.3: Feature visualization from Data set AAN. One point is a graph in *test* set. The layout is produced from high-level representations of DeepGraph, colored using structural, *hand-crafted* network properties, which are presented under each subfigures. Red (blue) color indicates high (low) property values.

closed triangles are actually features of graphlets [93, 109], we can see that DeepGraph has automatically learned these useful features without human input. Second, since edge density is a function of the number of edges and nodes, DeepGraph not only learns the number of edges and nodes (we do not show the edge and node property in Figure 4.3, but this is true), but also their none-linear relationship that involves division.

4.5.4 Error Analysis

Graphs in our data sets typically have hundreds of nodes, which is hard for humans to directly generalize useful information from a set of graphs. As a compromise, we characterize graphs by a set of simple network properties, e.g., the number of nodes, edges, and edge density.

We first want to investigate graphs for which DeepGraph makes more mistakes than baseline, and also the other way around. Here we use the strongest baseline, feature-based method as our reference. The procedure is as follows: among graphs where DeepGraph has smaller MSE than the baseline, we select the top 100 with the

largest MSE differences between the two methods. For these top graphs, we compute the average of the properties mentioned above. Similar procedure is also applied to the baseline.

The statistics of graphs where either DeepGraph or the baseline significantly outperforms the other are higher than the average statistics of each data set. This could result form the skewed distribution of the data set – a large number of graphs are of smaller size, leading to more training instances of small graphs. We also observe that both methods perform reasonably well on denser networks.

On the other hand, graphs on which DeepGraph performed better have relatively larger sizes than those where the baseline performed better. This indicates that the HKS representation has an advantage on larger graphs, the structures of which are more difficult to be represented by a bag of local substructures.

4.6 Conclusion

We present a novel neural network model that predicts the growth of egonet properties based on its graph structure. This model, DeepGraph, computes a new representation of the graph structure based on heat kernel signatures. A multi-column, multi-resolution convolution neural network is designed to further learn the high-level representations and predict the network growth in an end-to-end fashion. Experiments on large collections of real-world networks prove that DeepGraph significantly outperforms methods based on hand-crafted features, graph kernels, and competing deep learning methods. The higher-level representations learned by DeepGraph well correlate with findings and theories in social network literature, showing that a deep learning model can automatically discover meaningful and predictive structural patterns in networks.

Our study reassures the predictive power of network structures and suggests a way to effectively utilize this power. A meaningful future direction is to integrate network

structure with other types of information, such as the content of information cascades in the network. A joint representation of multi-modal information may maximize the performance of particular prediction tasks.

Despite that DeepGraph is outperforming competing methods, the prediction errors are still not negligible in their absolute sense. However, the improvement is still beneficial in many practical scenarios, e.g., when ranking information is more important than the absolute scale. Consider a scenario where a research can only afford time to read three papers per day. In this case, our DeepGraph is able to assist her to identify the top three most influential papers.

CHAPTER V

DeepCas: an End-to-end Predictor of Information Cascades

Information cascades, effectively facilitated by most social network platforms, are recognized as a major factor in almost every social success and disaster in these networks. Can cascades be predicted? While many believe that they are inherently unpredictable, recent work has shown that some key properties of information cascades, such as size, growth, and shape, can be predicted by a machine learning algorithm that combines many features. These predictors all depend on a bag of hand-crafting features to represent the cascade network and the global network structure. Such features, always carefully and sometimes mysteriously designed, are not easy to extend or to generalize to a different platform or domain.

Inspired by the recent successes of deep learning in multiple data mining tasks, we investigate whether an end-to-end deep learning approach could effectively predict the future size of cascades. Such a method automatically learns the representation of individual cascade graphs in the context of the global network structure, without hand-crafted features and heuristics. We find that node embeddings fall short on predictive power, and it is critical to learn the representation of a cascade graph as a whole. We present algorithms that learn the representation of cascade graphs in an end-to-end manner, which significantly improve the performance of cascade

prediction over strong baselines that include feature based methods, node embedding methods, and graph kernel methods. Our results also provide interesting implications for cascade prediction in general.

5.1 Introduction

Most modern social network platforms are designed to facilitate fast diffusion of information. Information cascades are identified to be a major factor in almost every plausible or disastrous social network phenomenon, ranging from viral marketing, diffusion of innovation, crowdsourcing, rumor spread, cyber violence, and various types of persuasion campaigns.

If cascades can be predicted, one can make wiser decisions in all these scenarios. For example, understanding which types of Tweets will go viral helps marketing specialists to design their strategies; predicting the potential influence of a rumor enables administrators to make early interventions to avoid serious consequences. A prediction of cascade size benefits business owners, investors, journalists, policy makers, national security, and many others.

Can cascades be predicted? While many believe that cascades are inherently unpredictable, recent work has shown that some key properties of information cascades, such as size, growth, and shape, can be predicted through a mixture of signals [19]. Indeed, cascades of microblogs/Tweets [129, 118, 131, 47, 23, 39], photos [19], videos [5] and academic papers [92] are proved to be predictable to some extent. In most of these studies, cascade prediction is cast as classification or regression problems and be solved with machine learning techniques that incorporate many features [118, 19, 23, 47]. On one hand, many of these features are specific to the particular platform or the particular type of information being diffused. For example, whether a photo was posted with a caption is shown to be predictive of how widely it spread on Facebook [19]; specific wording on Tweets is shown to help them gain more retweets

[103]. These features are indicative but cannot be generalized to other platforms or to other types of cascades. On the other hand, a common set of features, those extracted from the network structure of the cascade, are reported to be predictive by multiple studies [19, 129, 118].

Many of these features are carefully designed based on the prior knowledge from network theory and empirical analyses, such as centrality of nodes, community structures, tie strength, and structural holes. There are also ad hoc features that appear very predictive, but their success is intriguing and sometimes magical. For example, Cheng et al. [19] found that one of the most indicative feature to the growth of a cascade is whether any of the first a few reshares are not directly connected to the root of the diffusion.

We consider this as a major deficiency of these machine learning approaches: their performance heavily depends on the feature representations, yet there is no common principle of how to design and measure the features. Is degree the correct measure of centrality? Which algorithm should we use to extract communities, out of the hundreds available? How accurately can we detect and measure structural holes? How do we systematically design those "magical" features, and how do we know we are not missing anything important? Chances are whichever decisions we make we'll be losing information and making mistakes, and these mistakes will be accumulated and carried through to the prediction.

Can one overcome this deficiency? The recent success of *deep learning* in different fields inspires us to investigate an end-to-end learning system for cascade prediction, a system that pipes all the way through the network structures to the final predictions without making arbitrary decisions about feature design. Such a deep learning pipeline is expected to automatically learn the representations of the input data (cascade graphs in our case) that are the most predictive of the output (cascade growth), from a finer-granularity to increasingly more abstract representations, and allow the

lower-level representations to update based on the feedback from the higher levels. A deep neural network is particularly good at learning a nonlinear function that maps these representations to the prediction, in our case the future size of a cascade. While deep learning models have shown their great power of dealing with image, text, and speech data, how to design a suitable architecture to learn the representations of *graphs* remains a major challenge. In the context of cascade prediction, the particular barrier is how to go from representations of nodes to representing a cascade graph as a whole.

We present a novel, end-to-end deep learning architecture named the *DeepCas*, which first represents a snapshot of a cascade graph as a set of cascade paths that are sampled through multiple random walks processes. Such a representation not only preserves node identities but also bounds the loss of structural information. Analogically, cascade graphs are represented as documents, with nodes as words and paths as sentences. The challenge is how to sample the paths from a graph to assemble the "document," which is also automatically learned through the end-to-end model to optimize the prediction of cascade growth. Once we have such a "document" assembled, deep learning techniques for text data could be applied in a similar way here. We evaluate the performance of the proposed method using real world information cascades in two different domains, Tweets and scientific papers. DeepCas is compared with multiple strong baselines, including feature based methods, node embedding methods, and graph kernel methods. DeepCas significantly improves the prediction accuracy over these baselines, which provides interesting implications to the understanding of information cascades.

5.2 Related work

In a networked environment, people tend to be influenced by their neighbors' behavior and decisions [29]. Opinions, product advertisements, or political propaganda

could spread over the network through a chain reaction of such influence, a process known as the *information cascade* [117, 10, 4]. We present the first deep learning method to predict the future size of information cascades.

5.2.1 Cascade Prediction

Cascades of particular types of information are empirically proved to be predictable to some extent, including Tweets/microblogs [129, 118, 47, 23, 39, 131], photos [19], videos [5] and academic papers [92]. In literature, cascade prediction is mainly formulated in two ways. One treats cascade prediction as a classification problem [118, 47, 19, 23], which predicts whether or not a piece of information will become popular and wide-spread (above a certain threshold). The other formulates cascade prediction as a regression problem, which predicts the numerical properties (e.g., size) of a cascade in the future [118, 108]. This line of work can be further categorized by whether it outputs the final size of a cascade [131] or the size as a function of time (i.e., the growth of the cascade) [129]. Either way, most of the methods identified temporal properties, topological structure of the cascade at the early stage, root and early adopters of the information, and the content being spread as the most predictive factors.

These factors are utilized for cascade prediction in two fashions. The first mainly designs generative models of the cascade process based on temporal or structural features, which can be as simple as certain macroscopic distributions (e.g., of cascade size over time) [58, 6], or stochastic processes that explain the microscopic actions of passing along the information [129]. These generative models make various strong assumptions and oversimplify the reality. As a result, they generally underperform in real prediction tasks.

Alternatively, these factors may be represented through handcrafted features, which are extracted from the data, combined, and weighted by discriminative ma-

chine learning algorithms to perform the classification or the regression tasks [118, 19, 47, 23]. Most work in this fashion uses standard supervised learning models (e.g. logistic regression, SVM, or random forests), the performance of which heavily rely on the quality of the features. In general, there is not a principled and systematic way to design these features. Some of the most predictive features are tied to particular platforms or particular cascades and are hard to be generalized, such as the ones mentioned in the Section 5.1. Some features are closely related to the structural properties of the social network, such as degree[19, 118], density[19, 39], and community structures [118]. These features could generalize over domains and platforms, but many may still involve arbitrary and hard decisions in computation, such as what to choose from hundreds of community detection algorithms available [33] and how to detect structural holes [128]. Besides, there are also heuristic features that perform very well in particular scenarios but it is hard to explain why they are designed as is.

Our work differs from this literature as we take an end-to-end view of cascade prediction and directly learn the representations of a cascade without arbitary feature design. We focus on the structures (including node identities) of cascades as temporal and content information is not always available. In fact, content features are reported to be much weaker predictors than structural features [19]. Using temporal signals to predict future trend is a standard problem in time series, which is less interesting in this scope.

5.2.2 Learning the Representation of Graphs

Our work is also related to the literature of representation learning for graphs. Networks are traditionally represented as affiliation matrices or discrete sets of nodes and edges. Modern representation learning methods attempt to represent nodes as high-dimensional vectors in a continuous space (a.k.a., node embeddings) so that nodes with similar embedding vectors share similar structural properties (e.g., [81, 107, 38]).

Rather than learning the representation of each node, recent work also attempts to learn the representation of subgraph structures [78, 76, 122, 123]. Much of this work is inspired by the huge success of representation learning and deep learning applied to various domains such as text [7] and image [54]. For example, DeepWalk [81] makes an analogy between the nodes in networks and the words in natural language and uses fixed-length random walk paths to stimulate the "context" of a node so that node representations can be learned using the same method of learning word representations [69]. The representation of a graph can then be calculated by averaging the embeddings of all nodes.

Another line of related work comes from the domain of graph kernels, which computes pairwise similarities between graphs [12, 34, 94]. For example, the Weisfeiler-Lehman subtree kernel (**WL**) [94] computes the graph similarity based on the subtrees in each graph. Some studies have applied deep learning techniques to improve graph kernels [122, 75]. Though graph kernels are good at extracting structural information from a graph, it is hard for them to incorporate node identity information.

Another analogy connects graph structures to images. Motivated by representation learning of images, the topological structures of networks are first represented using locally connected regions [78], spectral methods [26], and heat kernel signatures [59], which could be passed through convolutional neural networks. These approaches are insensitive to orders of nodes and have an advantage of generating the same representation for isomorphic graphs. This nice property however comes at a price that it is hard to incorporate the identities of nodes.

Starting in next section, we present a novel end-to-end architecture that learns the representation of cascade graphs to optimize the prediction accuracy of their future sizes.

5.3 Method

In reality, we observe snapshots of the social network but may or may not observe the exact time when nodes and edges are introduced. Similarly, we may observe snapshots of a cascade but not its complete history. In other words, at a given time we know who have adopted the information but not when or through whom the information was passed through [19] (e.g., we know who cited a paper but not when and where she found the paper). Below we define the problem so that it is closely tied to the reality.

5.3.1 Problem Definition

Given a snapshot of a social network at time t_0, denote it as $\mathcal{G} = (V, E)$ where V is the set of nodes and $E \subset V \times V$ is the set of edges. A node $i \in V$ represents an actor (e.g., a user in Twitter or an author in the academic paper network) and an edge $(i, j) \in E$ represents a relationship tie (e.g., retweeting or citation) between node i and j up to t_0.

Let C be the set of cascades which start in $\mathcal{G}$ after time t_0. A snapshot of cascade $c \in C$ with a duration t after its origination is characterized by a *cascade graph* $g_c^t = (V_c^t, E_c^t)$, where V_c^t is a subset of nodes in V that have adopted the cascade c within duration t after its origination and $E_c^t = E \cap (V_c^t \times V_c^t)$, which is the set of edges in E with both ends inside V_c^t. These are the edges that are potentially used for information diffusion, as the cascade graph does not capture which edges were actually used for diffusion.

We consider the problem of predicting the **increment of the size** of cascade c after a given time interval Δt, which is denoted as $\Delta s_c = |V_c^{t+\Delta t}| - |V_c^t|$. The cascade prediction can then be formulated as, given $\mathcal{G}$, t, Δt, and $\{(g_c^t, \Delta s_c)\}_{c \in C}$, finding an optimal mapping function f that minimizes the following objective

$$\mathcal{O} = \frac{1}{|C|} \sum_{c} (f(g_c^t) - \Delta s_c)^2 \tag{5.1}$$

In the definition, t indicates the earliness of the prediction and Δt indicates the horizon of the prediction. When t is smaller, we are making predictions at the early stage of a cascade; when Δt is larger, we are predicting the size of cascade that is closer to its final status. These scenarios are particularly valuable but inherently harder in reality. It is worth noting that we consider the social network structure $\mathcal{G}$ as static in the prediction task. While in reality the global network does change over time, we are doing this to control for the effect of cascades on the network structure in this study - new edges may form due to a particular information cascade.

5.3.2 DeepCas: the End-to-End Pipeline

We propose an end-to-end neural network framework that takes as input the cascade graph g_c and predicts the increment of cascade size Δs_c. The framework (shown in figure 5.1) first samples node sequences from a cascade graph and then feeds the sequences into a gated recurrent neural network, where attention mechanisms are specifically designed to learn how to assemble sequences into a "document", so that the future cascade size could be predicted.

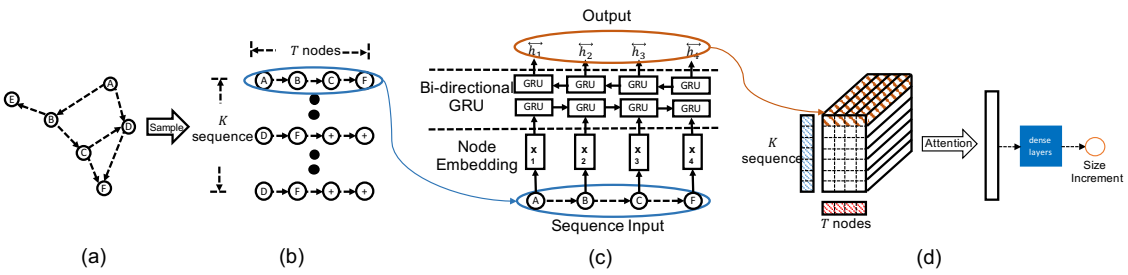


Figure 5.1: The end-to-end pipeline of DeepCas.

5.3.3 Cascade Graph as Random Walk Paths

Given a cascade graph g_c, the first component in DeepCas generates an initial representation of g_c using a set of node sequences.

Naturally, the future size of a cascade highly depends on who the information "propagators" are, which are the nodes in the current cascade graph. Therefore, a straightforward way to represent a graph is to treat it as a bag of nodes. However, this method apparently ignores both local and global structural information in g_c, which have been proven to be critical in the prediction of diffusions [19]. To remedy this issue, we sample from each graph a set of paths, instead of individual nodes. If we make a analogy between nodes and words, paths would be analogous to sentences, cascade graphs to documents, and a set of graphs to a document collection.

Similar to DeepWalk, the sampling process could be generalized as performing a random walk over a cascade graph g_c, the Markov chain of which is shown in Figure 5.2. At each step there is a current node and a state for the random walk of each diffusion graph. The starting state is S and the starting node is randomly sampled. State S is always followed by state N, with the node becoming a randomly selected neighbor of the current node. In state N, with probability $1 - p_j$, it stays in state N and the node transitions to a randomly selected neighbor of the current node. With probability p_j, it moves to a jump state J. With continue probability p_o, it jumps to a node randomly selected from the entire cascade graph, thus going back to state N. With probability $1 - p_o$, it goes to the terminal state T, terminating the entire random walk process.

Suppose the walker is at state N in the Markov chain and is currently visiting a node v, it follows a transition probability $p(u \in N_c(v)|v)$ to go to one of its outgoing neighbor $u \in N_c(v)$, where $N_c(v)$ denotes the set of v's outgoing neighbors in diffusion graph g_c. There are multiple strategies for setting transition probabilities. Given a specific choice of scoring function $\mathrm{sc_t}(u)$ to transit to node u, the neighbor u could be

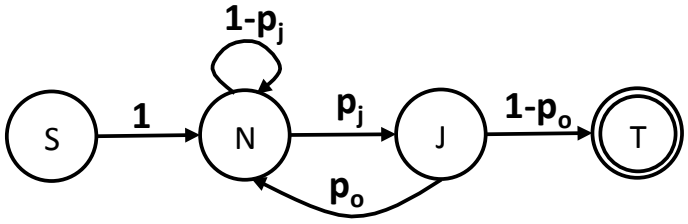


Figure 5.2: The Markov chain of random walk.

sampled in proportion to its score:

$$p(u \in N_c(v)|v) = \frac{\mathrm{sc_t}(u) + \alpha}{\sum_{s \in N_c(v)} (\mathrm{sc_t}(s) + \alpha)} \tag{5.2}$$

where α is a smoother. The scoring function $\mathrm{sc_t}(u)$ could be instantiated by but not limited to (1) $deg_c(u)$, the out-degree of node u in g_c, (2) $deg_{\mathcal{G}}(u)$, the degree of u in the global graph $\mathcal{G}$, or (3) $weight(v, u)$, the weight of the edge between the current node v and its neighbor u. Likewise, when the walker is at state J and is to select a node to jump to, the scoring function $\mathrm{sc_j}(u)$ could be set correspondingly.

$$p(u) = \frac{\mathrm{sc_t}(u) + \alpha}{\sum_{s \in V_c} (\mathrm{sc_t}(s) + \alpha)} \tag{5.3}$$

where V_c is the node set of g_c, and $\mathrm{sc_t}(u)$ could be (1) $deg_c(u)$, (2) $deg_{\mathcal{G}}(u)$, or (3) $\sum_{s \in N_c(u)} weight(u, s)$.

5.3.4 Sampling sequences from a graph

The probability p_o of whether to perform another random jump or go to the terminal state essentially determines the expected number of sampled sequences, while the probability p_j of whether to perform a random jump or transit to neighbors corresponds to the sequence length. The two factors play a key role in determining the representations of cascade graphs.

Naturally, different cascade graphs may require different parameters p_o and p_j,

as some are intrinsically more complex than others. Instead of fixing or manually tuning these two hyper-parameters, we propose to learn the two probabilities in an end-to-end manner by incorporating them to our deep learning framework. To do this, as Figure 5.1 (b) shows, we sample long enough sequences and sufficient number of sequences for all diffusion graphs. Denote T the sampled sequence length, K the sampled number of sequences, where T and K are the same for all diffusion graphs, we want to learn the actual length t_c and the actual number of sequences k_c we needed for each graph g_c, essentially a different parameterization of p_o and p_j. If a sampled sequence is shorter than the predefined length T, we pad this sequence with null nodes in the end till length T.

Note that existing work of using random walk paths to represent graphs such as DeepWalk and Node2Vec use fixed, predefined T and K. Automatically learning graph-specific path counts and lengths is a major technical contribution. We leave the learning of t_c and k_c to the next subsection.

5.3.5 Neural Network Models

Once we have sampled K sequences with T nodes for each diffusion graph, any effective neural networks for sequences could be applied to the random walk paths in a similar way as to text documents. The output of the neural network gives us the hidden representation of individual sequences. Unlike documents whose sentences are already written, we have to learn how to "assemble" these individual sequences into a "document," so that it can best represent the graph and predict its growth.

Node Embedding Each node in a sequence is represented as a one-hot vector, $q \in \mathbf{R}^{N_{node}}$, where N_{node} is the number of nodes in $\mathcal{G}$. All nodes share an embedding matrix $A \in \mathbf{R}^{H \times N_{node}}$, which converts a node into its embedding vector $x = Aq, x \in \mathbf{R}^H$.

GRU-based Sequence Encoding The sampled sequences represent the flow of information of a specific diffusion item. To capture this information flow, we use a Gated Recurrent Unite (GRU) [20], a specific type of recurrent neural network (RNN). When applying GRU recursively to a sequence from left to right, the sequence representation will be more and more enriched by information from later nodes in this sequence, with the gating mechanism deciding the amount of new information to be added and the amount of history to be preserved, which simulates the process of information flow during a diffusion. Specifically, denote step i the i-th node in a sequence, for each step i with input node embedding $x_i \in \mathbf{R}^H$ and previous hidden state $h_{i-1} \in \mathbf{R}^H$ as inputs, GRU computes the updated hidden state $h_i = \text{GRU}(x_i, h_{i-1}), h_i \in \mathbf{R}^H$.

For now we have read the sequence from left to right. We could also read the sequence from right to left, so that earlier nodes in the sequence could be informed by which nodes have been affected by a cascading item passed from them. To this end, we adopt the bi-directional GRU, which applies a forward GRU that reads the sequence from left to right, and a backward GRU from right to left. We denote the forward GRU as GRU_{fwd} and backward as GRU_{bwd}. As Figure 5.1 (c) shows, the representation of the i-th node in k-th sequence, $\overleftrightarrow{h}_i^k \in \mathbf{R}^{2H}$, is computed as the concatenation of the forward and backward hidden vectors.

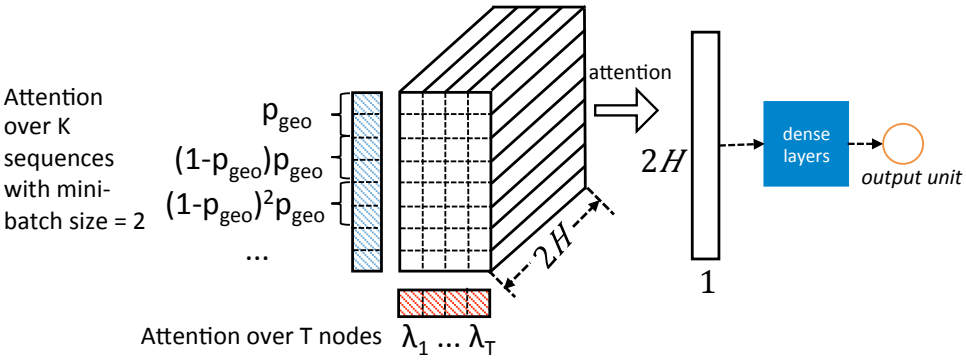


Figure 5.3: Attention to assemble the representation of the graph.

From sequence to graph representation Given a collection of sequence representations, where the k-th sequence with length T is represented as $[\overleftrightarrow{h}^k_1, ..., \overleftrightarrow{h}^k_i, ..., \overleftrightarrow{h}^k_T]$, as displayed in Figure 5.1 (d), we attempt to learn the representation of the cascade graph as a whole, so that it best predicts its future size. Analogically, we are assembling a document (graph) from a large number of very long sentences. We do this by learning the number of sentences and length of sentences per document, through an *attention* mechanism in deep learning.

In particular, the random walk on a graph terminates with probability $1 - p_o$. From the learning perspective, we could learn the value of p_o by examining whether the sampled number of sequences could represent the graph well, which in turn decides whether the prediction task is well performed. Intuitively, we could partition the sampled K sequences into "mini-batches." We want to read in more mini-batches until we could learn the graph well, simulating the action of jumping to the terminal state in the random walk. To implement this intuition, we assume a geometric distribution of *attentions* over mini-batches. If sequences in the first mini-batch of cascade g_c share attention weight p^c_{geo}, the next mini-batch will have attention $(1 - p^c_{geo})p^c_{geo}$, so on and so forth as Figure 5.3 shows. In theory, if we sample infinite number of sequences with the geometric distribution so that $K \to \infty$, the number of expected mini-batches to learn will be $1/p^c_{geo}$. With this expectation, learning the parameter p^c_{geo} could help us decide how many sequences to read in. Note that the degree of freedom is too high if we fit a free parameter p^c_{geo} per cascade. Instead, we rely on an observation that the number of sequences we need to represent a cascade graph is correlated with its size. Therefore, we condition p^c_{geo} on the size of graph $\text{sz}(g_c)$, more specifically $\lfloor \log_2(\text{sz}(g_c) + 1) \rfloor$. As a result, p^c_{geo} is replaced with $p_{geo}^{\lfloor \log_2(\text{sz}(g_c)+1) \rfloor}$.

We could apply similar procedure to learn sequence length. In practice, we found that the standard multinomial distribution of attentions already work well. So we simply assume multinomial distribution $\lambda_1, ..., \lambda_T$ over T nodes so that $\sum_i(\lambda_i) = 1$,

where $\{\lambda_i\}$ are shared across all cascade graphs.

To sum up and to give a mathematical representation, suppose the mini-batch size is B sequences, then the k-th sequence will fall into $(\lfloor k/B \rfloor + 1)$-th mini-batch, the attention mechanism then outputs the representation for graph g_c, a vector of length $2H$:

$$h(g_c) = \sum_{k=1}^{K} \sum_{i=1}^{T} \left((1 - a_c)^{\lfloor k/B \rfloor} a_c \right) \lambda_i \overleftrightarrow{h}_i^t, \tag{5.4}$$

where the first term corresponds to the attention over sequences with geometric distribution, and $a_c = p_{geo}^{\lfloor \log_2(\mathrm{sz}(g_c)+1) \rfloor}$. Both a_c and λ_i are learned through the deep learning process.

Output module Our output module consists of a fully connected layer with one final output unit: $f(g_c) = \mathrm{MLP}(h(g_c))$, where MLP stands for a multi-layer perceptron.

5.4 Experiment setup

We present comprehensive empirical experiments using real world data sets to evaluate the performance of DeepCas.

5.4.1 Data Sets

Most existing work evaluates their methods of predicting diffusions on a single social network data set (e.g., [19, 23, 41]. We add another completely different, publicly available data set to demonstrate the effectiveness and generalizability of DeepCas and to allow readers to reproduce our results.

One of the scenario is the cascade of Tweets on Twitter. Following the practice in existing work [87], we collect the TWITTER data set which contains the cascades of

Tweets (i.e., through retweeting) in June, 2016 from the official Decahose API (10% sample of the entire Tweet stream). All original English tweets that are published from June 1 to June 15 and retweeted at least once in 10 days are used for training. Those with only one retweets are downsampled to 5%. Cascades originated on June 16 are used for validation, and cascades originated from June 17 to June 20 are used for testing. A cascade contains the authors of the original Tweet and its retweets. Following [13, 23], we only consider users that appear in training stage, and they form the node set V.

We construct the global social network $\mathcal{G}$ using the same Tweet stream in April and May 2016. As the follower/followee relations are not available in the data and Twitter does not disclose the retweet paths, we follow existing work [87] and draw an edge from Twitter user A to B if either B retweeted a message of A or A mentioned B in a Tweet. Comparing to a follower/followee network, this network structure accumulates all information cascades and reflects the truly active connections between Twitter users. We weigh an edge based on the number of retweeting/mentioning events between the two users. To construct cascade graphs, we choose t, the duration of cascade since the original Tweet was posted, from a range of $t = 1, 3, 5$ days. We compute the increment of cascade size after t for the next Δt days, where $\Delta t = 1, 3, 5$ days. The combination of t and Δt yields a total of $3 \times 3 = 9$ configurations.

In the second scenario, we evaluate the prediction of the cascades of scientific papers. We collect the AMiner data set using the DBLP citation network released by ArnetMiner [1]. We construct the global network $\mathcal{G}$ based on citations between 1992 and 2002. That is, an edge draws from node A to B if author A is ever cited by B (which indicates that B might have found a reference from reading A's papers). A cascade of a given paper thus involves all authors who have written or cited that paper. Papers published between 2003 and 2007 are included in the training set.

[1] `https://aminer.org/citation`, DBLP-Citation-network V8, retrieved in August 2016.

Papers published in 2008 and 2009 are used for validation and testing, respectively. For the earliness and horizon of predictions, we set $t = 1, 2, 3$ years and $\Delta t = 1, 2, 3$ years respectively.

In both scenarios, we notice that the growth of all the cascades follows a power-law distribution, where a large number of cascades did not grow at all after t. Therefore we downsample 50% graphs with zero growth (to the numbers shown in Table 5.1) and apply a logarithm transformation of the outcome variable (increment of cascade size), following existing literature [55, 108].

Table 5.1: Statistics of the data sets.

	Set	Twitter			AMiner		
# nodes in $\mathcal{G}$	All	354,634			131,415		
# edges in $\mathcal{G}$	All	27,929,863			842,542		
t		1 day	3 days	5 days	1 year	2 years	3 years
# cascades	train	25,720	26,621	26,871	3,044	17,023	34,347
	val	1,540	1,563	1,574	509	3,665	7,428
	test	6,574	6,656	6,663	517	3,512	7,337
Avg. nodes per g_c	train	26.2	34.9	39.1	16.4	16.8	19.7
	val	46.1	62.1	69.7	10.6	13.6	17.2
	test	50.8	65.8	72.8	8.8	12.6	16.2
Avg. edges per g_c	train	99.0	153.8	188.3	56.8	54.9	68.5
	val	167.0	241.4	296.5	29.5	40.9	55.3
	test	162.3	242.2	289.0	22.6	32.9	44.5

5.4.2 Evaluation Metric

We use the mean squared error (MSE) to evaluate the accuracy of predictions, which is a common choice for regression tasks and used in previous work of cascade prediction [108, 129, 55]. As noted in Section 5.3.1, we predict a scaled version of the actual increment of the cascade size, i.e., $y_i = \log_2(\Delta s_i + 1)$.

5.4.3 Baseline methods

We compare DeepCas with a set of strong baselines, including feature-based methods used for cascade prediction, methods based on nodes embeddings, and alternative deep learning methods to learn graph representations.

Features-. We include all structural features that could be generalized across data sets from recent studies of cascade prediction [19, 41, 23, 110]. These features include:

Centrality and Density. Degree of nodes in the cascade graph g and the global network $\mathcal{G}$, average and 90th percentile of the local and global degrees of nodes in g, number of leaf nodes in g, edge density of g, and the number of nodes and edges in the *frontier graph* of the cascade, which is composed of nodes that are not in g but are neighbors of nodes in g.

Node Identity. The presence of node ids in g is used as features.

Communities. From both the cascade graph and the frontier graph, we compute the number of communities [11], the overlap of communities, and Gini impurity of communities [41].

Substructures. We count the frequency of k-node substructures ($k \leq 4$) [110]. These include nodes ($k = 1$), edges ($k = 2$), triads (e.g., the number of closed and open triangles) and quads from both the cascade graph and the frontier graph.

-linear and **-deep**. Once the cascade is represented as a set of features above, they are blended together using linear regression (denoted as **Features-linear**) with L2 regularization, as other linear regressors such as SVR empirically perform worth on our task. To obtain an even stronger baseline, we feed the feature vectors to MLP (denoted as **Features-deep**).

OSLOR selects important nodes as sensors, and predict the outbreaks based on the cascading behaviors of these sensors [23].

Node2vec [38] is selected as a representative of node embedding methods. As a

generalization of DeepWalk [81], node2vec is reported to be outperforming alternative methods such as DeepWalk and LINE [107]. We generate walks from two sources: (1) the set of cascade graphs $\{g\}$ (2) the global network $\mathcal{G}$. The two sources lead to two embedding vectors per node, which are concatenated to form the final embedding of each node. The average of embeddings of all nodes in a cascade graph is fed through MLP to make the prediction.

Embedded-IC [13] represents nodes by two types of embeddings: as a sender or as a receiver. For prediction, the original paper used Monte-Carlo simulations to estimate infections probabilities of each individual user. To predict cascade size, we experiment with two settings: (1) learn a linear mapping function between the number of infected users and the cascade size; (2) follow the setting of Node2Vec by using the average of embeddings of all nodes in the cascade graph, which is then piped through MLP. We find that the second setting empirically performs better than the first one. We therefore report the performance of the latter.

PSCN applies convolutional neural networks (CNN) to locally connected regions from graphs [78]. We apply PSCN to both the diffusion graphs and the frontier graphs. The last hidden layer of the cascade graph and that of the frontier graph are concatenated to make the final prediction.

Graph kernels. There are a set of state-of-the-art graph kernels [78]: the shortest-path kernel (SP) [12], the random walk kernel (RW) [34], and the Weisfeiler-Lehman subtree kernel (WL) [94]. The RW kernel and the SP kernel are too computationally inefficient, which did not complete after 10 days for a single data set in our experiment. We therefore exclude them from the comparison, a decision also made by in [78, 123]. For the WL kernel, we experiment with two settings: **WL-degree**, where node degree is used as the node attribute to build subgraphs for each cascade and frontier graph; **WL-id**, where node id is used as the attribute. The second setting is to test whether node identity information could be incorporated into graph

kernel methods.

Hyper-parameters. All together we have 8 baselines. All their hyper-parameters are tuned to obtain the best results on validation set for each configuration (9 in total) of each data set. For linear regression, we chose the L2-coefficient from $\{1, 0.5, 0.1, 0.05, ..., 10^{-8}\}$. For neural network regression, the initial learning rate is selected from $\{0.1, 0.05, 0.01, ..., 10^{-4}\}$, the number of hidden layers from $\{1, 2, ..., 4\}$, the hidden layer size from $\{32, 64, ..., 1024\}$, and L1- and L2-coefficient both from $\{1, 0.5, 0.1, 0.05, ..., 10^{-8}\}$. Following [78] for PSCN, the width is set to the average number of nodes, and the receptive field size is chosen between 5 and 10. The height parameter of WL is chosen from $\{2, 3, 4\}$. The candidate embedding size set is selected from {50, 100, 200, 300} for all methods that learn embeddings for nodes. For node2vec, we follow [38], p, q are selected from $\{0.25, 0.50, 1, 2, 4\}$, the length of walk is chosen from $\{10, 25, 50, 75, 100\}$, and the number of walks per node is chosen from $\{5, 10, 15, 20\}$.

5.4.4 DeepCas and the Variants

We compare a few variants of DeepCas with the 8 baselines. We sample $K = 200$ paths each with length $T = 10$ from the cascade graph without tuning the parameters. As described in Section 5.3.4 and 5.3.5, the attention mechanism will automatically decide when and where to stop using the sequences. The mini-batch size is set to 5. The smoother α is set to 0.01. The embedding sizes for the Twitter and AMiner data set are set to 150 and 50 respectively. The embeddings are initialized by concatenating embedding learned by Node2Vec from both all diffusion graphs $\{g\}$ in training set and the global network $\mathcal{G}$. The node2vec hyper-parameters p and q are simply set to 1.

We use **DeepCas-edge**, **DeepCas-deg**, and **DeepCas-DEG** to denote three version of DeepCas, which randomly walk with transition probabilities proportional

to edge weights, node degree in the cascade graph, and node degree in the global network. For comparison, we also include three simplified versions of DeepCas:

GRU-bag represents a cascade graph as a bag of nodes and feeds them through our GRU model. This is similar to setting the length of random walk paths to 1, which examines whether sequential information is important for cascade prediction.

GRU-fixed uses a fixed path length t and a fixed number of sequences k, without using the attention mechanism to learn them adaptively. Hyper-parameters t and k are tuned to optimal on the validation sets, the values of which are selected from $\{2, 3, 5, 7, 10\}$ and from $\{50, 100, 150, 200\}$, respectively.

GRU-root uses the attention mechanism, but starts sampling a random walk path only from roots, which are nodes who started the diffusion. If there are multiple roots, we take turns to sample from them. This examines whether it is important to perform random jumps in the walks over the graph.

5.5 Experiment results

In this section, we present the results of the experiments as designed in Section 5.4.

5.5.1 Overall performance

The overall performance of all competing methods across data sets are displayed in Table 5.2. The last three rows of each table show the performance of the complete versions of our methods, which outperform all eight baseline methods with a statistically significant drop of MSE. Please note that the numbers in Table 5.2 are errors of log-transformed outcomes. If we translate them back to raw sizes, the numerical differences between the methods would look larger.

The difference between Features-deep and Features-linear is intriguing, which shows that deep learning does not always perform better than linear methods if we

Table 5.2: Performance measured by MSE (the lower the better), where original label Δs is scaled to $y = \log_2(\Delta s + 1)$.

(a) Twitter

t	1 day			3 days			5 days		
Δt	1 day	3 days	5 days	1 day	3 days	5 days	1 day	3 days	5 days
Features-deep	1.644	2.253	2.432	1.116	1.687	2.133	0.884	1.406	1.492
Features-linear	1.665^{**}	2.256	2.464^{**}	1.123	1.706^{*}	2.137	0.885	1.425^{*}	1.505
OSLOR	1.791^{***}	2.485^{***}	2.606^{***}	1.179^{***}	1.875^{***}	2.181^{***}	0.990^{***}	1.539^{***}	1.778^{***}
node2vec	1.759^{***}	2.384^{***}	2.562^{***}	1.145^{**}	1.760^{***}	2.143	0.895	1.460^{***}	1.544^{***}
Embedded-IC	2.079^{***}	2.706^{***}	2.944^{***}	1.277^{***}	2.072^{***}	2.316^{***}	1.012^{***}	1.743^{***}	1.955^{***}
PSCN	1.735^{***}	2.862^{***}	2.911^{***}	1.134^{*}	1.784^{***}	2.411^{***}	0.893	1.461^{***}	1.566^{***}
WL-degree	1.778^{***}	2.568^{***}	2.691^{***}	1.177^{***}	1.890^{***}	2.205^{***}	0.939^{***}	1.568^{***}	1.825^{***}
WL-id	1.805^{***}	2.611^{***}	2.745^{***}	1.357^{***}	1.967^{***}	2.197^{***}	0.945^{***}	1.602^{***}	1.853^{***}
Proposed methods									
GRU-bag	1.769^{***}	2.374^{***}	2.565^{***}	1.172^{***}	1.822^{***}	2.159	0.932^{***}	1.472^{***}	1.594^{***}
GRU-fixed	1.606^{**}	2.149^{***}	2.286^{***}	1.132^{*}	1.675	1.825^{***}	0.891	1.376^{***}	1.513^{*}
GRU-root	1.572^{***}	2.202^{**}	2.147^{***}	1.097	1.726^{***}	1.762^{***}	0.874	1.406	1.489
DeepCas-edge	$\mathbf{1.480^{***}}$	1.997^{***}	2.074^{***}	1.013^{***}	$\mathbf{1.567^{***}}$	1.735^{***}	0.854^{***}	$\mathbf{1.322^{***}}$	1.422^{***}
DeepCas-deg	1.492^{***}	$\mathbf{1.933^{***}}$	$\mathbf{2.033^{***}}$	1.039^{***}	1.597^{***}	$\mathbf{1.707^{***}}$	0.854^{***}	1.330^{***}	$\mathbf{1.412^{***}}$
DeepCas-DEG	1.487^{***}	2.124^{***}	2.081^{***}	$\mathbf{1.012^{***}}$	1.644^{***}	1.724^{***}	$\mathbf{0.849^{***}}$	1.409	1.457^{***}

(b) AMiner

t	1 year			2 years			3 years		
Δt	1 year	2 years	3 years	1 year	2 years	3 years	1 year	2 years	3 years
Features-deep	1.748	2.148	2.199	1.686	1.876	1.954	1.504	1.617	1.686
Features-linear	1.737	2.145	2.205	1.690	1.887	1.964	1.529^{**}	1.626	1.697
OSLOR	1.768	2.173	2.225	1.897^{***}	1.964^{***}	2.057^{***}	1.706^{***}	1.738^{***}	1.871^{***}
node2vec	1.743	2.153	2.209	1.702	1.921^{***}	1.999^{***}	1.563^{***}	1.708^{***}	1.816^{***}
Embedded-IC	2.117^{***}	2.576^{***}	2.751^{***}	2.113^{***}	2.429^{***}	2.551^{***}	1.947^{***}	2.183^{***}	2.285^{***}
PSCN	1.880^{**}	2.332^{***}	2.424^{***}	1.853^{***}	2.164^{***}	2.092^{***}	1.770^{***}	1.822^{***}	1.857^{***}
WL-degree	1.742	2.234^{*}	2.350^{**}	1.780	2.037^{***}	2.079^{***}	1.586^{***}	1.762^{***}	1.864^{***}
WL-id	2.566^{***}	2.779^{***}	2.900^{***}	2.100^{***}	2.259^{***}	2.297^{***}	2.029^{***}	2.076^{***}	2.086^{***}
Proposed methods									
GRU-bag	1.783	2.217	2.242	1.712^{*}	1.982^{***}	1.988^{**}	1.614^{***}	1.743^{***}	1.856^{***}
GRU-fixed	1.703	2.064	2.151	1.569^{***}	1.735^{***}	1.805^{***}	1.430^{***}	1.537^{***}	1.564^{***}
GRU-root	1.816^{*}	2.222^{*}	2.331^{**}	1.890^{***}	1.972^{***}	2.146^{***}	1.660^{***}	1.778^{***}	1.813^{***}
DeepCas-edge	$\mathbf{1.668^{*}}$	$\mathbf{2.016^{**}}$	$\mathbf{2.084^{*}}$	1.545^{***}	$\mathbf{1.693^{***}}$	1.799^{***}	$\mathbf{1.402^{***}}$	1.477^{***}	1.548^{***}
DeepCas-deg	1.684^{*}	2.043^{*}	2.113^{*}	1.544^{***}	1.716^{***}	$\mathbf{1.792^{***}}$	1.407^{***}	$\mathbf{1.469^{***}}$	1.545^{***}
DeepCas-DEG	1.685^{*}	2.036^{*}	2.107^{*}	$\mathbf{1.540^{***}}$	1.700^{***}	1.788^{***}	1.404^{***}	1.480^{***}	$\mathbf{1.527^{***}}$

"***(**)" means the result is significantly better or worse over *Features-deep* according to paired t-test test at level 0.01(0.1).

have already found a set of good features. It is more important to learn end-to-end from the data.

Node2Vec and Embedded-IC do not perform well in cascade prediction. Taking the average of node embeddings as the graph representation is not as informative as representing the graph as a set of paths, even if the node embeddings are also fed into a deep neural net to make predictions. By comparing WL-degree and WL-id, we can see that it is hard for graph kernels to incorporate node identities. Simply using identities as node labels degenerates performance. This is because graph kernels rely on node labels to compute similarity between graphs. Using node id to measure

similarity could cause serious sparsity problem.

The three simplified versions of DeepCas, GRU-bag, GRU-fixed, and GRU-root all lead to certain degradation of performance, comparing to the three DeepCas models. This empirically proves the effectiveness of the three important components of DeepCas. First, sampling a set of paths to represent a graph instead of averaging the representations of nodes is critical, as it facilitates the learning of structural information. Second, learning the random walks by adaptively deciding when to stop sampling from a particular path and when to stop sampling more paths is more effective than using a fixed number of fixed-length paths (which is what DeepWalk does). The suitable numbers and lengths might be associated with the complexity and the influence power of a cascade graph. If a cascade graph is more complex and more "influential," it needs more paths and longer paths to represent its power. Third, sampling paths only from the root is not adequate (which is what most generative models do). Randomly jumping to other nodes could make the graph representation carry more information of the cascade structure and handle missing data. In a way, this is related to the "mysterious" feature used in Cheng et al. [19], i.e., whether some early adopters are not directly connected to the root.

Comparing the performance of using different t and Δt, we see a general pattern that can be applied to all methods: the larger the snapshot time t is, the easier to make a good prediction. This is because longer t makes more information available. While for Δt, it is the opposite, as it is always harder to make long-term predictions.

Training DeepCas is quite efficient. On a machine with 2.40 GHz CPU, 120G RAM and a single Titan X GPU, it takes less than 20 minutes to generate random walk paths for a complete data set and less than 10 minutes to train the deep neural network.

We also investigate cascades for which DeepCas makes more mistakes than the baselines, and also the other way around. DeepCas tend to perform better on larger

and denser graphs. These structures are more complex and harder to be represented as a bag of hand-crafted features. An end-to-end predictor without explicit feature design works very well in these cases. For the sake of space, we omit the detailed statistics here.

5.5.2 Interpreting the Representations

We have empirically shown that DeepCas could learn the representation of cascade graphs that incorporates both structures and node identities. Qualitatively, we have not assessed what the learned representation actually captures from these information. Indeed, one concern of applying deep learning to particular domains is that the models are black-boxes and not easy to interpret. For us, it is intriguing to know whether the learned representation corresponds to well-known network properties and structural patterns in literature.

To do this, we select a few hand-crafted features which are computed for the feature based baselines. These features characterize either global or local network properties, and are listed in Figure 5.4. In each subfigure, we layout the cascade graphs as data points in the test set to a 2-D space by feeding their vector representations output by the last hidden layer of DeepCas to t-SNE [11], a commonly used visualization algorithm. Cascade graphs with similar vector representations are placed closely. To connect the hand-crafted features with the learned representations, we color each cascade graph (a point in the 2-D visualization) by the values of each feature (e.g., network density). If we eyeball a pattern of the distribution of colors in this 2-D layout, it suggests a connection between the learned representation and that network property. We also color the layout by the ground-truth labels (increment of cascade size). If the color distribution of labels looks somewhat correlated with the color distribution of a network property, we know this property attributes to cascade prediction, although not through a hand-crafted feature.

As we observe, DeepCas could capture structural properties like the number of open, closed triangles, and the number of communities. For example, in the Figure 5.4 (e), the points (cascade graphs) clustered to the bottom right have the fewest communities, while graphs in the top left have the most. Cascade graph with a larger number of communities implies that many early adopters may lie in between bigger communities, which are likely to be structural holes in the global network. In literature [14], nodes spanning structural holes are likely to gain social capital, promoting the growth of its ego-net. Indeed, when we compare the color scheme of 5.4(g) with 5.4(i), we can see that the number of communities in a cascade graph is indeed positively correlated with its growth.

Figure 5.4 (f) plots the average global degree of nodes in each cascade graph. The pattern suggests that DeepCas not only captures the structural information from individual cascade graphs, but also incorporates the global information into the graph representation. How did this happen? Although we did not explicitly represent the global network $\mathcal{G}$ (or the frontier graphs), DeepCas is likely to learn useful global network information from the many cascade graphs in training (similar to a model that captures collection-level information from the input of many individual documents), and incorporate it into the high-level representation of a cascade graph.

Some additional observations can be made from Figure 5.4. First, as the number of open and closed triangles are actually important features used for graph prediction tasks [110], we can see that DeepCas has automatically learned these useful features without human input. Second, since edge density is a function of the number of edges and nodes, DeepCas learns not only the number of edges and nodes (we do not show the node property in Figure 5.4, but this is true), but also their none-linear relationship that involves division.

5.6 Discussion and Conclusion

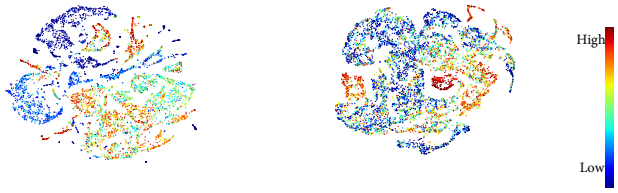
We present the first end-to-end, deep learning based predictor of information cascades. A cascade graph is first represented as a set of random walk paths, which are piped through a carefully designed GRU neural network structure and an attention mechanism to predict the future size of the cascade. The end-to-end predictor, DeepCas, outperforms feature-based machine learning methods and alternative node embedding and graph embedding methods.

While the study adds another evidence to the recent successes of deep learning in a new application, social networks, we do wish to point the readers to a few more interesting implications. First, we find that linearly combined, hand-crafted features perform reasonably well in cascade prediction, which outperform a series of node embedding, graph embedding, and suboptimal deep learning methods. Comparing to other data mining domains, social network is a field where there exists rich theoretical and empirical domain knowledge. Carefully designed features inherited from the literature are already very powerful in capturing the critical properties of networks. The benefit of deep learning in this case really comes from the end-to-end procedure, which is likely to have learned high-level features that just better represent these network properties. Comparing to deep learning methods, feature-based methods do have their advantages (if the right features are identified), as both the results and the importance of features are easier to interpret. For social network researchers, it is perhaps a good idea to interpret DeepCas as a way to test the potential room to improve cascade prediction, instead of as a complete overturn of the existing practice. Indeed, it is intriguing to pursue how to design better measurements of the classical network concepts (e.g., communities and centrality), based on the results of DeepCas.

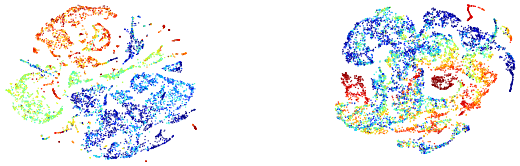
Another interesting finding is that different random walk strategies perform better and worth in different scenarios, and all better than bag of node embeddings. This is where prior knowledge in social networks literature may kick in, by incorporating

various contagion/diffusion processes to generate initial representations of cascade networks. How to choose from multiple cascading processes itself is an interesting question of reinforcement learning.

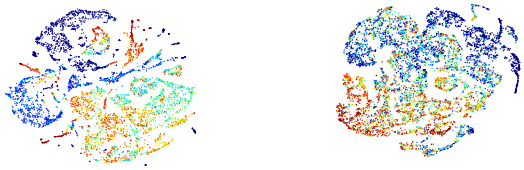
Finally, to make our conclusion clean and generalizable, we only utilized the network structure and node identities in the prediction. It is interesting to incorporate DeepCas with other types of information when they are available, e.g., content and time series, to optimize the prediction accuracy on a particular domain.



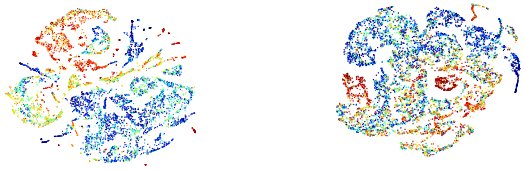
(a) # closed triangles. (b) # open triangles.



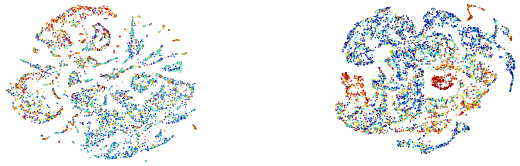
(c) # communities. (d) # communities.



(e) Edge density. (f) Avg. degree in $\mathcal{G}$.



(g) # leaf nodes. (h) # edges.



(i) Increment of diffusion size on TWITTER. (j) Increment of diffusion size on AMINER.

Figure 5.4: Feature visualization. Every point is a cascade graph in *test* set. Every layout is colored (red: high, blue: low) using *hand-crafted* network properties or the ground-truth, labeled under each subfigures. The left column displays graphs from TWITTER, while the right column shows AMINER.

CHAPTER VI

Joint Modeling of Text and Networks for Cascade Prediction

The DeepCas model takes into account the global network structure for the representation learning of cascade graphs. In addition to the global context, another important source of information that resides in cascade graphs is text. A diffusion item can be described by a text message, and nodes participating diffusions are also associated with text. Text provides valuable information for the learning of graphs, especially when nodes of incoming cascades rarely participate in previous diffusions, leading to a lack of structural information. In the extreme case, new nodes that are absent in the training stage has no structural information available in the test stage. Under these circumstances, text information could be leveraged to learn the representation of nodes, which assembles the graph representation. We introduce a new end-to-end learning method that joint models of text and networks for cascade prediction. A gating mechanism is introduced to dynamically fuse the structural and textual representations of nodes based on their respective properties. To incorporate the text information associated with both the item being diffused and the nodes in the cascade, attentions are employed over the nodes' content based on their interactions with the item's content. Empirical evaluations demonstrate that incorporating text information benefits the cascade prediction task significantly, and that our pro-

posed model outperforms alternative methods that combines structural information and text information.

6.1 Introduction

DeepCas introduced in Chapter V has successfully approached the problem of learning network representation in the context of the global network structure. What has been left out is the content of the cascade. Indeed, textual content is an important source of information that often presents in a cascade. The item being diffused is usually in the form of a text message or can be described by text – Tweets, posts, and scientific papers. On the other hand, users who are passing these items along are also associated with rich text information, which often indicates their preferences or interests. For example, a Twitter user has a history of Tweets and retweets; every researcher has a list of publications.

In a machine learning perspective, text could greatly complement structures, especially when the nodes of a cascade (a test example) have rarely participated in previous cascades (training examples), causing a lack of structural information for these nodes. In the extreme case, incoming cascades may introduce new nodes that are absent from the history (training cascades). If the representation of a cascade is only learned from the structural relationships of nodes, which is exactly what DeepCas does, the learned embedding vectors will not be available for the new nodes. Text information helps in these cases, based on the intuition that nodes with similar text content are likely to be close to each other in the embedding space and nodes with similar content to the diffused item are likely to adopt the item. This motivates us to jointly model text and networks, so that we can effectively embed all nodes into the hidden space, which forms the basis of learning good representations of graphs. In this work, we extend DeepCas by equipping it with the capability to handle text.

In literature, researchers have explored various approaches to incorporating text

content into the learning of node representations [124, 79]. How to go beyond node to learn representation for the entire graph with the presence of text content is not well studied. There is another line of research called multimodal learning, which learns from multiple modalities (e.g., text, images, and audio). Common approaches of multimodal learning aim to learn a shared representation from different modalities [77, 98]. In our context, however, simply treating graph structure as one modality and the aggregated text content from graph nodes as another leads to significant loss of information. This is due to the complexity of information networks – text isn't present as a single document for a cascade, but resides in individual nodes, and a cascade is a complex system of nodes and edges. Moreover, independent from the graph structure is the item being diffused, which is also associated with a piece of text. To better handle the cascade prediction task, a model considering the special nature of information networks is needed.

To utilize the structural and textual information of nodes in a better way, we introduce a gating mechanism to dynamically fuse the node representations from two sources, based on how well each representation is learned. To incorporate the text information from both the item being diffused and the nodes involved in the cascade, an attention mechanism is employed over the content of nodes, which is conditioned on their interactions with the content of the item. Empirical evaluations demonstrate that incorporating text information benefits the cascade prediction task significantly, and that our proposed model outperforms alternative combination methods, methods that use manually designed features, methods that use standard concatenation of text and structure information, and methods using multimodal learning.

6.2 Related work

A considerable amount of attention has been devoted to joint modeling of text and networks, especially for the area of topic modeling. Mei et al. [67] propose to

regularize topic models based on network structure. Following this line, various topic models considering both factors are developed [64, 16].

More recently, researchers begin to explore the possibility of incorporating text contents to learn node representations. Yang et al. [124] propose text-associated DeepWalk [81] to incorporate text features into the matrix factorization framework. TriDNR proposed in [79] captures the relationships between node and its words by maximizing the co-occurrence of word sequences given this node. These studies focus on learning node representations. How to learn representation for the entire graph with the presence of text information is still not well studied.

A general line of research related to our work is multimodal deep learning, which focuses on learning from different modalities (e.g., text, images and audio). Instead of concatenating representations learned from different sources, common approaches of multimodal learning aim to learn a shared representation from different modalities [77, 98]. This practice incorporates correlations across the modalities, and is robust to situations where some modalities are absent. To improve model's ability to predict missing modalities, Sohn et al. [97] proposed to minimize the information distance between data modalities. Our problem setting is more complicated than the general setting of multimodal deep learning. If we simply treat graph structure as one modality and aggregated text content from all graph nodes as another modality, there will be a significant loss of information. A model specific to the task has to be designed to account for the fact that text resides in each node, while nodes and edges consist of the graph, and there is text on the graph level – diffusion items can be described by text.

6.3 Method

In this section, we will first describe notations in addition to the ones used in DeepCas, followed by the proposed method.

6.3.1 Notations for Text Content

We have given the problem definition of cascade prediction in Section 5.3.1, which remains the same in this chapter. Here we introduce additional notations for text information on both nodes and diffusion items. Specifically, each node $v \in V$ at time t_0 is associated with a text document $d_v^{t_0} = \{w_1, w_2, ..., w_{|d_v^{t_0}|}\}$, where w_k is k-th word tokens in document $d_v^{t_0}$. For example, a Twitter user can have a document that is a concatenation of all her history tweets. If a node has no text information at time t_0, this list will simply be empty. Without confusion, we will omit t_0, and simply use d_v in the following discussion. For each cascade $c \in C$, there is a text document $d_c = \{w_1, w_2, ..., w_{|d_c|}\}$.

Originally, a cascade graph g_c^t is characterized by V_c^t and E_c^t, where V_c^t are nodes that have adopted the cascade c within duration t after its origination and E_c^t denotes edges. When text is considered, a cascade graph can be represented as $g_c^t = (V_c^t, E_c^t, \{d_v^t, v \in V_c^t\}, d_c)$, where $\{d_v^t, v \in V_c^t\}$ is a set of text documents of the cascade adopters.

6.3.2 The pipeline to jointly model structure and text

The proposed framework to model structure and text jointly is shown in Figure 6.1. It takes as input the cascade graph g_c and predicts the increment of cascade size Δs_c. The first two steps of the framework is the same as DeepCas, which first samples node sequences from cascade graphs and then feeds the sequences into a gated recurrent neural network. Figure 6.1 (c) shows the major difference, where a gating mechanism is employed to fuse the structural and textural representation of nodes. After that, the same attention mechanism is used to assemble sequences into graph representation, so that the future cascade size could be predicted.

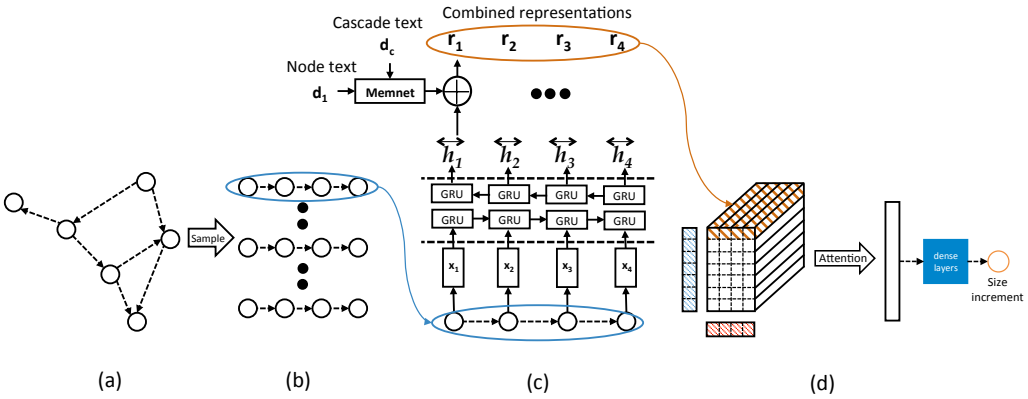


Figure 6.1: The end-to-end pipeline to jointly model structure and text.

6.3.3 Modeling structure

The structural representation of nodes could be directly obtained from the original DeepCas model. In Figure 6.1 (c), the representation of node v output by GRU is $\overleftrightarrow{h_v} \in \mathbf{R}^{2H}$, where H is the size of embedding vectors. This representation, calculated based on its relationships to neighboring nodes, captures the structural information of each node. Therefore, $\overleftrightarrow{h_v}$ could be treated as the structural representation of node v.

6.3.4 Modeling text

Text representation of nodes can be simply computed as $m_v = \phi(d_v)$, where d_v is the text document of node v. The function $\phi(\cdot)$, which could be any one like CNN or RNN, abstracts the representation of text.

Modeling node text in this way does not account for its interaction with the item being diffused. Consider an example of a researcher who is an expert in natural language processing. This researcher might have a big influence when propagating a machine translation paper, but probably not an image recognition paper. In other words, the text representation of nodes could change dynamically with respect to the cascade text d_c, and thus need to be calculated as $m_v = \phi(d_v, d_c)$.

To account for this dynamism, we apply an attention mechanism over the node

text. By conditioning attention on the cascade text, different parts of the node text could be attended given the cascade text. The structure of the attention mechanism could be implemented by a memory network [99], where each memory cell stores one word of node text, and cascade text is the input question.

6.3.5 Fusing structure and text representation

Jointly learning the representation of nodes could be as simple as a concatenation of its structural and textual representation. However, this concatenation learns to assign a global weight of importance to each source, failing to consider the uniqueness of nodes. On one hand, some nodes appear so frequently in cascades that structural embedding is already sufficient to represent the node. On the other hand, some nodes have rich text information to be utilized. Finally, there are nodes with scarce information from both sources.

In order to take into account specific properties of each node, we design a gating mechanism to dynamically fuse the structural and textual representations of nodes. The gates control how information flows by measuring the informativeness of each source. The informativeness of structural information, $i_v^{(s)}$, can be simply measured by $\text{fq}(v)$, the frequency of node v occurred in the training set, scaled by logarithm:

$$i_v^{(s)} = \log(\text{fq}(v + 1)). \tag{6.1}$$

To measure the informativeness of text, we compute the match between the node text and the cascade text, as users who are constantly promoting certain topics are likely to have larger influence in those topics. One thing to be noted is that there are also some general topics that match well with a large population of users. To account for this factor, we are actually measuring how much better the node text matches the cascade text than average. Similar to negative sampling [69], the match of average

nodes can be approximately computed by randomly sampling a node set $V_s \subseteq V$. In this way, the informativeness of text $i_v^{(t)}$ is calculated as:

$$i_v^{(t)} = \psi(d_v)^T \psi(d_c) - \frac{1}{|V_s|} \sum_{v' \in V_s} \psi(d_{v'})^T \psi(d_c), \tag{6.2}$$

where $\psi(\cdot)$ computes the document representation simply by taking the average of word embeddings.

Given the two measures $i_v^{(s)}$ and $i_v^{(t)}$, we can now use two gates $g_v^{(1)}$ and $g_v^{(2)}$ to distinguish the three cases mentioned above:

$$g_v^{(1)} = \sigma(W^{(1)}[i_v^{(s)}; i_v^{(t)}]) \tag{6.3}$$

$$g_v^{(2)} = \sigma(W^{(2)}[i_v^{(s)}; i_v^{(t)}]) \tag{6.4}$$

$$r_v = (1 - g_v^{(1)}) \cdot \left((1 - g_v^{(2)}) \cdot \overleftrightarrow{h_v} + g_v^{(2)} \cdot m_v\right) + g_v^{(1)} \cdot e, \tag{6.5}$$

where $\sigma(x) = 1/(1 + \exp(x))$ is the sigmoid function, $\cdot$ represents an element-wise product, and e is an embedding vector learned globally that represents nodes with neither rich structural nor textual information. If gate $g_v^{(1)}$ is close to one, it chooses information more from the global embedding e. If $g_v^{(2)}$ is close to one, it allows more information from text, rather than structure, to flow through.

6.3.6 From sequence to graph representation

The rest of the framework basically follows DeepCas, except that the formation of the final graph representation is based on the fused representation r_v, rather than the structural representation $\overleftrightarrow{h_v}$.

6.3.7 Multi-task learning

So we have had a workable pipeline to joint model structure and text. As an addition, we explore here whether additional training signals could help for the task of cascade prediction. We do this in an unsupervised manner, without requiring additional ground-truth labels.

Remember that when we compute the informativeness of text $i_v^{(t)}$, we are comparing the match score of nodes in the cascade with a random set of nodes. This could be further formulated as a pairwise ranking task, based on the intuition that the text of cascade adopters is likely to have a better match with the cascade text than random nodes. Specifically, denote $\text{sc}_v = \psi(d_v)^T\psi(d_c)$ the score of node v, and $\text{sc}_{V_s} = \frac{1}{|V_s|}\sum_{v'\in V_s}\psi(d_{v'})^T\psi(d_c)$ the average score of node set V_s. Given a pair of scores $(\text{sc}_v, \text{sc}_{V_s})$, we train our model in a way that it could rank sc_v higher than sc_{V_s}. To this end, for half of the nodes $v \in V_c$ in cascade, we generate (data, label) tuples as $((\text{sc}_v, \text{sc}_{V_s}), 1)$; for the rest nodes, both the score pair and the label are reversed: $((\text{sc}_{V_s}, \text{sc}_v), -1)$.

We train this ranking task together with the main task of cascade prediction.

6.4 Experiment setup

6.4.1 Data Sets

Following Section 5.4, we continue to use Twitter and AMiner as our data sets. To avoid using future information, we only use a piece of text of a node if the text is generated before training time. Specifically for Twitter, we collect user tweets and retweets in April and May 2016. All tweets of a user in this period is considered as the node text. For AMiner, we gather all titles of each author's publications between 1992 and 2002.

In previous work [61, 23], new nodes that only appear in test stage are not in-

cluded. This time these new nodes are added back, which is more like the real setting.

6.4.2 Baseline methods

Apart from strong baselines in Chapter V, we additionally compare with node embedding methods that model text content of nodes, and variants of DeepCas that combine textual information.

Features-deep. In additional to features used in Chapter V, we include text based features, including ngrams ($n = 1, 2, 3$), the average of word embeddings, and topic distribution of text, which is found by Latent Semantic Analysis (LSA) [25], from both nodes and cascades. The features are fed to a deep MLP network.

Node2vec [38]. We concatenate the average of node embeddings and word embeddings of both node and cascade text, which is fed through MLP to make the prediction.

TriDNR [79]. TriDNR is a node embedding method that captures the relationships between node and its words by maximizing the co-occurrence of word sequences given this node. Its original objective function includes three components: network structure, text, and node labels. Since we do not have node labels, we only optimize for the first two components. As node text is already incorporated into the node embedding, we only use the average of node embedding and embedding of cascade text for prediction.

DeepCas proposed in Chapter V, which does not consider text information. Since the three versions of DeepCas, DeepCas-edge, DeepCas-deg, and DeepCas-DEG, perform similarly, we simply choose DeepCas-edge for comparison. All the newly proposed methods will be based on DeepCas-edge, which is shortened as DeepCas for simplicity.

Hyper-parameters. Following Chapter V, the hyper-parameters of baselines are tuned to obtain the best results on validation set for each configuration of each

data set. For neural network regression, the initial learning rate is selected from $\{0.1, 0.05, 0.01, ..., 10^{-4}\}$, the number of hidden layers from $\{1, 2, ..., 4\}$, the hidden layer size from $\{32, 64, ..., 1024\}$, and L1- and L2-coefficient both from $\{1, 0.5, 0.1, 0.05, ..., 10^{-8}\}$. The candidate embedding size set is selected from {50, 100, 200} for all methods that learn embeddings for nodes and text. For node2vec, we follow [38], p, q are selected from $\{0.25, 0.50, 1, 2, 4\}$, the length of walk is chosen from $\{10, 25, 50, 75, 100\}$, and the number of walks per node is chosen from $\{5, 10, 15, 20\}$.

6.4.3 The proposed methods

We compare a few variants of DeepCas with the baselines. The same setting as Chapter V is used. We sample $K = 200$ paths each with length $T = 10$ from the cascade graph without tuning the parameters. The mini-batch size is set to 5. The smoother α is set to 0.01. The node embedding sizes for the TWITTER and AMINER data set are set to 100 and 50 respectively. The embeddings are initialized by concatenating embedding learned by Node2Vec from both all diffusion graphs $\{g\}$ in training set and the global network $\mathcal{G}$. The node2vec hyper-parameters p and q are simply set to 1. All text embeddings are set to size of 50, and are initialized by training all the text documents of each data set using word2vec. The number of hops of memory networks is chosen from $1, 2, 3$.

DeepCas-cat. Instead of using gates to fuse structural and textual representation of nodes as proposed in Section 6.3.5, they are simply concatenated to form the node representation.

DeepCas-multimodal. This is another approach to combine structural and textual representation of nodes, based on multimodal deep learning. We employ a classical multimodal learning method developed by Srivastava et al. [98] to learn a shared representation for structure and text.

DeepCas-gate. The gating mechanism introduced in Section 6.3.5 is used to fuse

structural and textual representation of nodes.

DeepCas-multitask. Based on DeepCas-gate, we form a multitask setting by including additional training signals, as proposed in Section 6.3.7.

6.5 Experiment results

We present the results of the experiments as designed in Section 6.4.

6.5.1 Overall performance

Table 6.1: Performance measured by MSE (the lower the better), where original label Δs is scaled to $y = \log_2(\Delta s + 1)$.

(a) TWITTER

t	1 day			3 days			5 days		
Δt	1 day	3 days	5 days	1 day	3 days	5 days	1 day	3 days	5 days
Features-Deep	2.894^{***}	3.514^{***}	3.501^{***}	2.113^{***}	3.026^{***}	3.109^{***}	0.956^{***}	1.542^{***}	1.723^{***}
node2vec	2.387^{***}	2.936^{***}	2.930^{***}	1.980^{***}	2.731^{***}	2.706^{***}	1.053^{***}	1.687^{***}	1.855^{***}
TriDNR	2.678^{***}	3.439^{***}	3.510^{***}	2.131^{***}	3.062^{***}	3.143^{***}	1.109^{***}	1.887^{***}	2.161^{***}
DeepCas	2.102	2.758	2.772	1.465	2.004	2.020	0.907	1.410	1.494
Proposed methods									
DeepCas-cat	1.990^{***}	2.415^{***}	2.454^{***}	1.430^{***}	1.907^{***}	1.960^{***}	0.925	1.372^{***}	1.468^{**}
DeepCas-multimodal	1.982^{***}	2.433^{***}	2.493^{***}	1.432^{***}	1.931^{***}	1.968^{***}	0.942^{*}	1.391^{**}	1.475^{**}
DeepCas-gate	$1.945^{***}_{\triangledown\triangledown\triangledown}$	$2.144^{***}_{\triangledown\triangledown\triangledown}$	$2.397^{***}_{\triangledown\triangledown\triangledown}$	$1.371^{***}_{\triangledown\triangledown\triangledown}$	$1.862^{***}_{\triangledown\triangledown\triangledown}$	$1.871^{***}_{\triangledown\triangledown\triangledown}$	$0.890^{***}_{\triangledown\triangledown\triangledown}$	$1.312^{***}_{\triangledown\triangledown\triangledown}$	$1.392^{***}_{\triangledown\triangledown\triangledown}$
DeepCas-multitask	1.942^{***}	2.139^{***}	2.402^{***}	1.375^{***}	1.855^{***}	$1.842^{***}_{\diamond\diamond}$	0.885^{***}	1.314^{***}	$1.377^{***}_{\diamond}$

(b) AMINER

t	1 year			2 years			3 years		
Δt	1 year	2 years	3 years	1 year	2 years	3 years	1 year	2 years	3 years
Features-Deep	2.678^{*}	2.902^{**}	2.922^{***}	1.908^{***}	1.990^{***}	2.032^{***}	1.608^{**}	1.683^{***}	1.748^{***}
node2vec	2.466	2.663^{*}	2.706^{**}	1.902^{***}	2.046^{***}	2.073^{***}	1.697^{***}	1.786^{***}	1.832^{***}
TriDNR	2.586^{*}	2.821^{**}	2.866^{**}	1.971^{***}	2.110^{***}	2.130^{***}	1.678^{***}	1.763^{***}	1.806^{***}
DeepCas	2.425	2.556	2.576	1.826	1.898	1.914	1.575	1.607	1.643
Proposed methods									
DeepCas-cat	2.327^{*}	2.439^{**}	2.488^{*}	1.811	1.861^{**}	1.865^{***}	1.531^{***}	1.568^{*}	1.563^{***}
DeepCas-multimodal	2.395	2.513	2.529	1.805	1.867^{**}	1.891^{*}	1.552^{*}	1.592	1.622^{**}
DeepCas-gate	$2.301^{**}_{\triangledown}$	2.412^{**}	2.494^{*}	$1.742^{***}_{\triangledown\triangledown\triangledown}$	$1.818^{***}_{\triangledown\triangledown\triangledown}$	$1.820^{***}_{\triangledown\triangledown\triangledown}$	$1.482^{***}_{\triangledown\triangledown\triangledown}$	$1.529^{***}_{\triangledown\triangledown\triangledown}$	$1.502^{***}_{\triangledown\triangledown\triangledown}$
DeepCas-multitask	2.293^{**}	2.414^{**}	2.479^{*}	$1.729^{***}_{\diamond}$	1.816^{***}	1.823^{***}	1.477^{***}	1.524^{***}	1.508^{***}

"***(**, *)" means the result is significantly better or worse over *DeepCas* according to paired t-test test at level 0.01(0.05, 0.1). "$\triangledown$" means *DeepCas-gate* is significantly better over *DeepCas-cat*, and "$\diamond$" means *DeepCas-multitask* is significantly better over *DeepCas-gate*.

The overall performance of all competing methods across data sets are displayed in Table 6.1. The last row of each table shows the performance of the complete version of our methods, which outperform all baseline methods, including DeepCas, with a statistically significant drop of MSE. Please note that the numbers in Table 6.1

are errors of log-transformed outcomes. If we translate them back to raw sizes, the numerical differences between the methods would look larger.

Unlike the results presented in Table 5.2 of the previous Chapter, where Features-Deep is mostly the strongest baseline, both node2vec and TriDNR could now outperform feature-based method in some of the configurations. This could result from the lack of end-to-end learning procedure, while a large number of designed features, including the newly incorporated text-based features, are used as input.

Node2vec and TriDNR underperform DeepCas, which does not consider any text input. This confirms that taking the average of embeddings, both structural and textual, as the graph representation is not as informative as representing the graph as a set of paths.

DeepCas-cat, which simply concatenates structural and textual representations, can already beat DeepCas. This shows the benefits of incorporating text information, which helps when the structural information of nodes are hard to come by. In this case, text provides supplementary information to embed nodes into the hidden space.

DeepCas-cat and DeepCas-multimodal do not perform better than DeepCas-gate. The problem of DeepCas-cat might be that simple concatenation fails to learn the importance of structural and textual information for each node individually. For DeepCas-multimodal, it tries to learn a shared representation over structure and text. Due to the large amount of text, the shared representation might be overwhelmed by text, and consequently the knowledge from structure is not well represented. By considering specific properties of each node, DeepCas-gate, which applies the proposed gating mechanism, could learn to adjust the information flow from each source more dynamically and smartly.

DeepCas-multitask performs comparable to DeepCas-gate, and outperforms it in two configurations. This suggests that adding additional training signals does no harm to cascade prediction, and can bring benefits in some of the cases.

6.5.2 Error analysis

To analyze how text information could help with the prediction task, we compare the behavior of DeepCas and DeepCas-gate for networks of different properties. Specifically, we use two properties to examine whether text could really benefit graphs with more new nodes, nodes that only occur in test stage: the average of node occurrences in training stage (*avg node freq*), and the percentage of old nodes that appear in training stage (*old node pct*).

For each property, we sort networks by its value, and take 100 networks with the largest/smallest values. Based on these networks with extreme property values, we evaluate the performance of both methods, which is averaged across all configurations.

Table 6.2: MSE for 100 networks with extreme property values.

	Property	smallest avg node freq	smallest old node pct	largest avg node freq	largest old node pct
Twitter	DeepCas	2.045	2.132	3.334	2.485
	DeepCas-gate	1.494	1.540	3.358	2.495
AMiner	DeepCas	2.451	2.792	1.698	1.828
	DeepCas-gate	1.891	2.109	1.721	1.747

From Table 6.2, we see that DeepCas-gate outperforms DeepCas by a large margin for networks with the smallest values of the two properties *avg node freq* and *old node pct*, while performs similar for networks with the largest values. When the two properties are small, the networks consist of many nodes that appear rarely, lacking enough information to learn these nodes from structure. Therefore, this confirms our assumption that text could provide supplementary information when we are faced with scarce structural information.

6.5.3 Relationship between text content and prediction difficulty

To further analyze the role of text content, we study whether cascades of some text topics are much easier to predict than others. To this end, we apply LSA to produce the topic distribution of each cascade in the test set. Based on the distribution,

cascades are assigned to the topic with the highest probability score. In this way, each topic is associated with a list of cascades. To quantify the prediction difficulty, we calculate the average of errors for cascades belonging to each topic.

Table 6.3: Topics with highest/lowest average errors.

Twitter	Lowest err.	day having tryna beach today night favorite smiling hope enjoying
		pokemon catch rare game playing play feelings catching bio spot
		thing seen video way realest prettiest realize laugh funniest trying
		good morning night friends times looking sounds things giveaway work
	Highest err.	heart way mind breaks eyes win broken feel hurts scene
		hate things signs miss used drama publicly privately etah lot
		girl baby ask boy wants picture needs gets favorite date
		retweet vote free following win follows friends follow star timeline
AMiner	Lowest err.	graph mining patterns multiple matching problems graphs pattern object frequent
		models topic process probabilistic business language unsupervised document joint latent
		fuzzy adaptive clustering sets selection method decision type rough making
		mobile phones devices location services ad hoc interaction application adaptive
	Highest err.	web services content ranking automatic integration clustering extraction composition pages
		semantic ontology discovery role syntactic similarity latent parsing annotation matching
		recognition face activity human expression facial object representation entity named
		classification selection support feature vector text machines machine method pattern

Table 6.3 displays topics either with highest or lowest average errors. On Twitter, it seems like cascade prediction is easier when the content is about sharing enjoyable moments or items, or recent popular games. In the field of computer science, the topics of graph mining, topic modeling, fuzzy methods, and mobile computing can be predicted with less errors.

6.6 Conclusion

In this chapter, we explored how to model structure and text jointly for cascade prediction. Text is present at different levels of the cascade graphs. A diffusion item can be described by text message, and nodes participating diffusions are also associated with text. Text provides valuable information for the learning of graphs, especially when their node members rarely participate in diffusions, leading to difficulties in learning from structure. In these cases, text could come to rescue by providing supplementary information. This motivates the joint modeling of text and networks for cascade prediction. To this end, a gating mechanism is introduced to dynamically

fuse the structural and textual representations of nodes based on their respective properties. To incorporate the text information associated with both diffusion items and nodes, attentions are employed over node text based on their interactions with item text. Empirical evaluations demonstrate that incorporating text information benefits the cascade prediction task, and that the proposed gating mechanism is superior to alternatives, including a simple combination of text and structure information, and standard multimodal learning.

CHAPTER VII

Conclusions

In this thesis, we presented approaches to mining text and network data in an end-to-end manner. In this chapter, we conclude this thesis and discuss potential future directions.

7.1 Summary

Text and networks convey a large amount of information in today's online communities and social media, and are thus vital data sources for researchers to study users and understand a variety of phenomena. Modeling text and networks for various tasks has become increasingly important for people from different fields, including social scientists, online marketers, government officers, scientific researchers, and daily users.

Studying user behavior is an important research direction in online communities and social media. We address it by mining both text and network data. More specifically, we study attitude identification in the text mining domain, and network growth prediction in the network domain. The advantage of studying both text and networks can also be viewed from the perspective of data types. Unlike images and time series, text and networks are both discrete, thus sharing similar challenges and solutions. Indeed, our DeepCas model, which learns to predict cascade growth, is

inspired by the learning of document representations.

Conventional methods could easily fall short of resolving the challenges rooted in the tasks of text and network mining. The performance of these methods hinges on features designed by experts, who find it hard to encode every piece of necessary knowledge for a particular task. On the other hand, the existence of subtasks poses more challenges for conventional methods, which train a separate model for each subtask. In this case, the interactions among subtasks are ignored, and the information is not well shared.

To resolve these issues, we focus on designing end-to-end learning algorithms for text and information network mining. We employ deep learning techniques to directly learn from raw data, by using deeply layered, hierarchical concepts, with complicated concepts built upon simpler ones. With the ability to learn the nonlinear mapping from input to output in an end-to-end manner, deep learning relieves humans from the burden of hard coding world knowledge by, e.g., designing features or rules. The power of deep learning enables us to design effective and efficient end-to-end algorithms for both attitude identification and network prediction tasks.

Attitude identification aims to identify people's attitudes towards a given set of entities. Conventionally, the task is decomposed into two separate subtasks: target detection and polarity classification, which neglects intrinsic interactions between the two subtasks. Apart from subtask interaction, interactions among targets exist – certain targets and their sentiments may share some important semantic dimensions with each other while differing on other dimensions. To resolve these issues, we propose an end-to-end machine learning architecture, where the two subtasks are interleaved by a deep memory network that directly learns from the raw text input. The proposed model also allows different targets to interact with each other, which share a common semantic space and simultaneously keep their own space, making it possible for all targets to be learned in a unified model. The proposed deep memory

network markedly outperforms models that do not consider the subtask or target interactions, including conventional supervised learning methods and state-of-the-art deep learning models.

In the domain of network mining, a major challenge is to find good representations of graph structures, which form the basis for all downstream tasks. In the social network literature, graphs are represented by a bag of manually designed structural features, e.g., network density, clustering coefficients, and triadic profiles. However, it is hard for these features to fully represent both the local and the global structure of a graph and the complex interaction between them. On the other hand, these features usually have a limited characterization power of networks, as many different networks may share the same feature representation. To remedy these issues, we introduce a graph descriptor that is based on the Heat Kernel Signature (HKS) [100], which serves as a universal low-level representation of the topological structures of networks. By modeling the amount of heat flow over the nodes of a network over time, HKS successfully stores both the global and the local structural information of the entire network, and networks with the same topological structure can be mapped to a unique representation of the little loss of structural information. However, unlike 3D objects that are composed of polygon meshes, the structures of networks vary in shape, size, and complex local structures. To address this issue, some computations of HKS need to be approximated carefully. Inspired by the semantics of the HKS-based graph descriptors, we propose a multicolumn, multiresolution neural network that learns latent hierarchical representations of graphs on top of the HKS-based graph descriptor. The proposed deep neural network, named DeepGraph, predicts network growth in an end-to-end process. We conduct extensive experiments to evaluate the effectiveness of DeepGraph. Different growing properties are predicted for four genres of real-world networks. Empirical results show that our method significantly outperforms baseline approaches that use alternative graph representations, handcrafted

features, or existing deep learning architectures.

DeepGraph has successfully resolved the challenge of learning graph structures. We then turn to the challenge of jointly learning different information sources that reside in information networks, including the global network context, and text content. To address this challenge, we specifically consider the task of learning network representation for cascade prediction, whose objective is to predict the future size of a cascade network. In order to take into account the context of the global network structure, we present a novel, end-to-end deep learning architecture named the *DeepCas*, which first represents a cascade graph as a set of cascade paths that are sampled through multiple random walk processes. Such a representation not only preserves node identities, but also bounds the loss of structural information. Analogically, cascade graphs are represented as documents, with nodes as words and paths as sentences. The challenge is how to sample the paths from a graph to assemble the "document," which is also automatically learned through the end-to-end model to optimize the prediction of cascade growth. We evaluate the performance of the proposed method using real-world information cascades in two different domains, Tweets, and scientific papers. DeepCas is compared with multiple strong baselines, including feature-based methods, node-embedding methods, and graph kernel methods. DeepCas significantly improves the prediction accuracy over these baselines, which provides interesting implications for the understanding of information cascades.

After modeling global context, we approach the problem of joint modeling text and networks. Text could greatly complement structural information, especially when node members of cascades rarely participate in previous diffusions, causing lack of structural information. In the extreme case, new nodes that are absent in the training stage could appear in the test stage. If the graph representation is only learned from the structural relationships between nodes, which is exactly what DeepCas does, embedding vectors learned from the structure will not be available for these new

nodes.

Text could help in these cases, based on the intuition that nodes with similar text content might be close to each other in the embedding space. To better utilize both the structural and textual information of nodes, a gating mechanism is designed to dynamically fuse the node representations from two sources, based on how well each representation is learned. To incorporate the text information from both diffusion items and nodes, an attention mechanism is employed over node text, which is conditioned on their interactions with item text. Empirical evaluations demonstrate that incorporating text information benefits the cascade prediction task and that the proposed gating mechanism outperforms other combination methods, including a simple combination of text and structure information, and standard multimodal learning.

This thesis demonstrates the effectiveness of end-to-end learning for mining text and network data. We show that end-to-end learning techniques can be more powerfully employed when we are able to first construct a good, low-level representation of the raw format of the discrete data, and when the interactions among subtasks are correctly identified and considered. These findings are not domain-specific and can be applied more generally. Of course, many challenges remain to be solved, which will be discussed below.

7.2 Future directions

Our approaches to end-to-end learning for mining text and network data open up new opportunities for many directions of future research, including:

Scalability: There are, at least, two directions worth exploration. The first direction is how to efficiently and effectively represent the topology of a graph when the size of the graph is large. How to efficiently compute HKS when the graph size is large? Are there any approximation methods? Will the approximation lead to

significant loss of information? Successfully solving these issues could help us apply DeepGraph to more general domains with more efficiency. The second direction is how to employ DeepCas when there are a large number of nodes in the global context. Currently, we are learning a vector representation for each node, which might cause out of memory issues when there are billions of nodes. How to tackle this problem is an interesting direction to explore.

Modeling temporal properties of graphs: Many networks change dynamically. New nodes join a network from time to time, and new relationships are formed, while old nodes and edges might disappear. Incorporating these dynamics into the learning of graph representation could help us understand the networks better, and can greatly benefit many valuable mining tasks.

Incorporate network information for text mining tasks: Chapter VI tackles the problem of joint modeling text and networks for cascade prediction, which is a network-mining task. In fact, the paradigm of joint modeling text and networks can also be applied to many text-mining tasks. For example, in the task of attitude identification, neighboring users might share similar attitudes towards certain entities. How to incorporate network information for these text-mining tasks is intriguing.

Incorporating domain knowledge: End-to-end learning does not mean that it excludes the incorporation of domain knowledge. End-to-end helps when it is hard for humans to encode world knowledge. Conversely, domain knowledge could also contribute to learning in many scenarios. How to better utilize existing knowledge from domain experts is critical in that it might help our learning systems learn more efficiently and effectively with less training data, while enjoying better generalization capability.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proc. of SIGKDD*, 2006.

[2] L. Bai, L. Rossi, A. Torsello, and E. R. Hancock. A quantum jensen–shannon graph kernel for unattributed graphs. *Pattern Recognition*, 2015.

[3] X. Bai and E. R. Hancock. Heat kernels, manifolds and graph embedding. In *Structural, Syntactic, and Statistical Pattern Recognition*. Springer, 2004.

[4] A. V. Banerjee. A simple model of herd behavior. *The Quarterly Journal of Economics*, 1992.

[5] C. Bauckhage, F. Hadiji, and K. Kersting. How viral are viral videos. In *Proc. of ICWSM*, 2015.

[6] C. Bauckhage, K. Kersting, and F. Hadiji. Mathematical models of fads explain the temporal dynamics of internet memes. In *Proc. of ICWSM*, 2013.

[7] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *JMLR*, 2003.

[8] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 1994.

[9] D. Bespalov, B. Bai, Y. Qi, and A. Shokoufandeh. Sentiment classification based on supervised latent n-gram analysis. In *Proc. of CIKM*, 2011.

[10] S. Bikhchandani, D. Hirshleifer, and I. Welch. A theory of fads, fashion, custom, and cultural change as informational cascades. *Journal of political Economy*, 1992.

[11] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *JSTAT*, 2008.

[12] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proc. of ICDM*, 2005.

[13] S. Bourigault, S. Lamprier, and P. Gallinari. Representation learning for information diffusion through social networks: an embedded cascade model. In *Proc. of WSDM*, 2016.

[14] R. S. Burt. The network structure of social capital. *Research in organizational behavior*, 2000.

[15] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *TIST*, 2011.

[16] J. Chang, J. Boyd-Graber, and D. M. Blei. Connections between the lines: augmenting social networks with text. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. ACM, 2009.

[17] N. V. Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*. 2005.

[18] R. Chen, Y. Chen, Y. Liu, and Q. Mei. Does team competition increase pro-social lending? evidence from online microfinance. *Games and Economic Behavior*, 2015.

[19] J. Cheng, L. Adamic, P. A. Dow, J. M. Kleinberg, and J. Leskovec. Can cascades be predicted? In *Proc. of WWW*, 2014.

[20] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[21] F. Chung. Laplacians and the cheeger inequality for directed graphs. *Annals of Combinatorics*, 2005.

[22] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *JMLR*, 2011.

[23] P. Cui, S. Jin, L. Yu, F. Wang, W. Zhu, and S. Yang. Cascading outbreak prediction in networks: a data-driven approach. In *Proc. of SIGKDD*, 2013.

[24] M.-C. De Marneffe, B. MacCartney, C. D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proc. of LREC*, 2006.

[25] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[26] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

[27] L. Dong, F. Wei, C. Tan, D. Tang, M. Zhou, and K. Xu. Adaptive recursive neural network for target-dependent twitter sentiment classification. In *ACL*, 2014.

[28] B. G. P. M. Dragomir R. Radev, Mark Thomas Joseph. A Bibliometric and Network Analysis of the field of Computational Linguistics. *JASIST*, 2009.

[29] D. Easley and J. Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[30] Y. Fang, M. Sun, and K. Ramani. Heat-passing framework for robust interpretation of data in networks. *PloS one*, 2015.

[31] Y. Fang, J. Xie, G. Dai, M. Wang, F. Zhu, T. Xu, and E. Wong. 3d deep shape descriptor. In *Proc. of CVPR*, 2015.

[32] A. Faulkner. Automated classification of stance in student essays: An approach using stance target information and the wikipedia link-based measure. In *FLAIRS*, 2014.

[33] S. Fortunato. Community detection in graphs. *Physics reports*, 2010.

[34] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*. 2003.

[35] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proc. of ACL*, 2011.

[36] X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proc. of ICML*, 2011.

[37] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[38] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proc. of SIGKDD*, 2016.

[39] A. Guille and H. Hacid. A predictive model for the temporal dynamics of information diffusion in online social networks. In *Proc. of WWW*, 2012.

[40] R. Guimera, B. Uzzi, J. Spiro, and L. A. N. Amaral. Team assembly mechanisms determine collaboration network structure and team performance. *Science*, 2005.

[41] R. Guo, E. Shaabani, A. Bhatnagar, and P. Shakarian. Toward order-of-magnitude cascade prediction. In *Proc. of ASONAM*, 2015.

[42] K. S. Hasan and V. Ng. Stance classification of ideological debates: Data, models, features, and constraints. In *IJCNLP*, 2013.

[43] K. M. Hermann and P. Blunsom. The role of syntax in vector space models of compositional semantics. In *ACL*, 2013.

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

[45] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proc. of SIGKDD*, 2004.

[46] O. Irsoy and C. Cardie. Deep recursive neural networks for compositionality in language. In *NIPS*, 2014.

[47] M. Jenders, G. Kasneci, and F. Naumann. Analyzing and predicting viral tweets. In *Proc. of WWW*, 2013.

[48] L. Jiang, M. Yu, M. Zhou, X. Liu, and T. Zhao. Target-dependent twitter sentiment classification. In *Proc. of ACL*, 2011.

[49] H. Kashima, K. Tsuda, and A. Inokuchi. Kernels for graphs. *Kernel methods in computational biology*, 2004.

[50] Y. Kim. Convolutional neural networks for sentence classification. In *Proc. of EMNLP*, 2014.

[51] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proc. of ICLR*, 2015.

[52] N. Kobayashi, R. Iida, K. Inui, and Y. Matsumoto. Opinion mining on the web by extracting subject-aspect-evaluation relations. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, 2006.

[53] G. Kossinets and D. J. Watts. Empirical analysis of an evolving social network. *science*, 2006.

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. of NIPS*, 2012.

[55] A. Kupavskii, L. Ostroumova, A. Umnov, S. Usachev, P. Serdyukov, G. Gusev, and A. Kustarev. Prediction of retweet cascade size over time. In *Proc. of CIKM*, 2012.

[56] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014.

[57] Y. Le Cun, L. Jackel, B. Boser, J. Denker, H. Graf, I. Guyon, D. Henderson, R. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 1989.

[58] K. Lerman and R. Ghosh. Information contagion: An empirical study of the spread of news on digg and twitter social networks. *ICWSM*, 10:90–97, 2010.

[59] C. Li, X. Guo, and Q. Mei. Deepgraph: Graph structure predicts network growth. *arXiv preprint arXiv:1610.06251*, 2016.

[60] C. Li, X. Guo, and Q. Mei. Deep memory networks for attitude identification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 671–680. ACM, 2017.

[61] C. Li, J. Ma, X. Guo, and Q. Mei. Deepcas: an end-to-end predictor of information cascades. In *Proceedings of the 26th International Conference on World Wide Web*, pages 577–586. International World Wide Web Conferences Steering Committee, 2017.

[62] F. Li, C. Han, M. Huang, X. Zhu, Y.-J. Xia, S. Zhang, and H. Yu. Structure-aware review mining and summarization. In *Proc. of ACL*, 2010.

[63] P. Liu, S. Joty, and H. Meng. Fine-grained opinion mining with recurrent neural networks and word embeddings. In *EMNLP*, 2015.

[64] Y. Liu, A. Niculescu-Mizil, and W. Gryc. Topic-link lda: joint models of topic and author community. In *proceedings of the 26th annual international conference on machine learning*, pages 665–672. ACM, 2009.

[65] H. Ma, H. Yang, M. R. Lyu, and I. King. Mining social networks using heat diffusion processes for marketing candidates selection. In *Proc. of CIKM*, 2008.

[66] D. Marcheggiani, O. Täckström, A. Esuli, and F. Sebastiani. Hierarchical multi-label conditional random fields for aspect-oriented opinion mining. In *ECIR*, 2014.

[67] Q. Mei, D. Cai, D. Zhang, and C. Zhai. Topic modeling with network regularization. In *Proceedings of the 17th international conference on World Wide Web*, pages 101–110. ACM, 2008.

[68] Q. Mei, X. Ling, M. Wondra, H. Su, and C. Zhai. Topic sentiment mixture: modeling facets and opinions in weblogs. In *Proc. of WWW*, 2007.

[69] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[70] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[71] A. Mislove. *Online Social Networks: Measurement, Analysis, and Applications to Distributed Information Systems*. PhD thesis, 2009.

[72] S. M. Mohammad, S. Kiritchenko, P. Sobhani, X. Zhu, and C. Cherry. Semeval-2016 task 6: Detecting stance in tweets. In *Proc. of SemEval*, 2016.

[73] S. M. Mohammad, P. Sobhani, and S. Kiritchenko. Stance and sentiment in tweets. *arXiv preprint arXiv:1605.01655*, 2016.

[74] S. M. Mohammad and P. D. Turney. Emotions evoked by common words and phrases: Using mechanical turk to create an emotion lexicon. In *Proc. of NAACL*, 2010.

[75] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. In *Workshop on Mining and Learning with Graphs*, 2016.

[76] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*, 2016.

[77] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[78] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. 2016.

[79] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang. Tri-party deep network representation. *Network*, 11(9):12, 2016.

[80] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2008.

[81] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proc. of SIGKDD*, 2014.

[82] M. Pontiki, D. Galanis, H. Papageorgiou, S. Manandhar, and I. Androutsopoulos. Semeval-2015 task 12: Aspect based sentiment analysis. In *Proc. of SemEval*, 2015.

[83] M. Pontiki, D. Galanis, J. Pavlopoulos, H. Papageorgiou, I. Androutsopoulos, and S. Manandhar. Semeval-2014 task 4: Aspect based sentiment analysis. In *Proc. of SemEval*, 2014.

[84] A. Popescu and M. Pennacchiotti. Dancing with the stars, nba games, politics: An exploration of twitter users response to events. In *Proc. of ICWSM*, 2011.

[85] A. Rajadesingan and H. Liu. Identifying users with opposing opinions in twitter debates. In *Social Computing, Behavioral-Cultural Modeling and Prediction*. 2014.

[86] S. Ranu and A. K. Singh. Graphsig: A scalable approach to mining significant subgraphs in large graph databases. In *Proc. of ICDE*, 2009.

[87] J. Ratkiewicz, M. Conover, M. Meiss, B. Gonçalves, S. Patil, A. Flammini, and F. Menczer. Truthy: mapping the spread of astroturf in microblog streams. In *Proc. of WWW*, 2011.

[88] D. M. Romero, C. Tan, and J. Ugander. On the interplay between social and topical structure. *Proc. of ICWSM*, 2013.

[89] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[90] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 2009.

[91] C. Sauper and R. Barzilay. Automatic aggregation by joint modeling of aspects and values. *JAIR*, 2013.

[92] H.-W. Shen, D. Wang, C. Song, and A.-L. Barabási. Modeling and predicting popularity dynamics via reinforced poisson processes. *arXiv preprint arXiv:1401.0778*, 2014.

[93] N. Shervashidze, T. Petri, K. Mehlhorn, K. M. Borgwardt, and S. Vishwanathan. Efficient graphlet kernels for large graph comparison. In *International conference on artificial intelligence and statistics*, pages 488–495, 2009.

[94] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 2011.

[95] R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proc. of EMNLP-CoNLL*, 2012.

[96] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*, 2013.

[97] K. Sohn, W. Shang, and H. Lee. Improved multimodal deep learning with variation of information. In *Advances in Neural Information Processing Systems*, pages 2141–2149, 2014.

[98] N. Srivastava and R. R. Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012.

[99] S. Sukhbaatar, J. Weston, R. Fergus, et al. End-to-end memory networks. In *NIPS*, 2015.

[100] J. Sun, M. Ovsjanikov, and L. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, 2009.

[101] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. In *Proc. of VLDB*, 2011.

[102] G. Szabo and B. A. Huberman. Predicting the popularity of online content. *Communications of the ACM*, 2010.

[103] C. Tan, L. Lee, and B. Pang. The effect of wording on message propagation: Topic-and author-controlled natural experiments on twitter. *arXiv preprint arXiv:1405.1438*, 2014.

[104] D. Tang, B. Qin, and T. Liu. Aspect level sentiment classification with deep memory network. In *EMNLP*, 2016.

[105] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *ACL*, 2014.

[106] J. Tang, M. Qu, and Q. Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proc. of SIGKDD*, 2015.

[107] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proc. of WWW*, 2015.

[108] O. Tsur and A. Rappoport. What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities. In *Proc. of WSDM*, 2012.

[109] J. Ugander, L. Backstrom, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proc. of WWW*, 2013.

[110] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *Proc. of the National Academy of Sciences*, 2012.

[111] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *Proc. of WOSN*, 2009.

[112] D.-T. Vo and Y. Zhang. Target-dependent twitter sentiment classification with rich automatic features. In *IJCAI*, 2015.

[113] M. A. Walker, P. Anand, R. Abbott, and R. Grant. Stance classification using dialogic properties of persuasion. In *Proc. of NAACL*, 2012.

[114] M. A. Walker, P. Anand, R. Abbott, J. E. F. Tree, C. Martell, and J. King. That is your evidence?: Classifying stance in online political debate. *Decision Support Systems*, 2012.

[115] H. Wang, Y. Lu, and C. Zhai. Latent aspect rating analysis without aspect keyword supervision. In *Proc. of SIGKDD*, 2011.

[116] S. Wang and C. D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proc. of ACL*, 2012.

[117] I. Welch. Sequential sales, learning, and cascades. *The Journal of finance*, 1992.

[118] L. Weng, F. Menczer, and Y.-Y. Ahn. Predicting successful memes using network and community structure. *arXiv preprint arXiv:1403.6199*, 2014.

[119] J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[120] T. Wilson, J. Wiebe, and P. Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proc. of HLT/EMNLP*, 2005.

[121] J. Xie, Y. Fang, F. Zhu, and E. Wong. Deepshape: Deep learned shape descriptor for 3d shape matching and retrieval. In *Proc. of CVPR*, 2015.

[122] P. Yanardag and S. Vishwanathan. Deep graph kernels. In *Proc. of SIGKDD*, 2015.

[123] P. Yanardag and S. Vishwanathan. A structural smoothing framework for robust graph comparison. In *NIPS*, 2015.

[124] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang. Network representation learning with rich text information. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina*, pages 2111–2117, 2015.

[125] J. Yang and J. Leskovec. Modeling information diffusion in implicit networks. In *Proc. of ICDM*, 2010.

[126] L. Yang, T. Sun, M. Zhang, and Q. Mei. We know what@ you# tag: does the dual role affect hashtag adoption? In *Proc. of WWW*, 2012.

[127] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proc. of SIGIR*, 1999.

[128] Y. Yang, J. Tang, C. W.-k. Leung, Y. Sun, Q. Chen, J. Li, and Q. Yang. Rain: Social role-aware information diffusion. In *Proc. of AAAI*, 2015.

[129] L. Yu, P. Cui, F. Wang, C. Song, and S. Yang. From micro to macro: Uncovering and predicting information cascading process with behavioral dynamics. In *Proc. of ICDM*, 2015.

[130] M. Zhang, Y. Zhang, and D.-T. Vo. Gated neural networks for targeted sentiment analysis. In *Proc. of AAAI*, 2016.

[131] Q. Zhao, M. A. Erdogdu, H. Y. He, A. Rajaraman, and J. Leskovec. Seismic: A self-exciting point process model for predicting tweet popularity. In *Proc. of SIGKDD*, 2015.

[132] C. Zirn, M. Niepert, H. Stuckenschmidt, and M. Strube. Fine-grained sentiment analysis with structural features. In *IJCNLP*, 2011.