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

Cheng Li[1], Jiaqi Ma[1], Xiaoxiao Guo[2], Qiaozhu Mei[1,2]
[1]School of Information, University of Michigan, Ann Arbor, MI, USA
[2]Department of EECS, University of Michigan, Ann Arbor, MI, USA
{lichengz, jiaqima, guoxiao, qmei}@umich.edu

ABSTRACT

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 structures. 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 or heuristics. We find that node embeddings fall short of 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 including feature based methods, node embedding methods, and graph kernel methods. Our results also provide interesting implications for cascade prediction in general.

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 agents, 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 [10]. Indeed, cascades of microblogs/Tweets [43, 39, 44, 20, 11, 17, 37], photos [10], videos [2] and academic papers [31] are proved to be predictable to some extent. In most of these studies, cascade prediction is cast as classification or regression problems and is solved with machine learning techniques that incorporate many features [39, 10, 11, 20]. On one hand, many of these features are specific to the particular platform or to 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 [10]; specific wording on Tweets is shown to help them gain more retweets [33]. 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 [10, 43, 39].

Many of these features are carefully designed based on the prior knowledge from both 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. [10] 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 predictions.

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-

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052643

level representations to update based on the feedback from the higher levels. A deep neural network is particularly good at learning a non-linear 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 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. 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. To ease the reproduction of our results, we have made the source code of DeepCas publicly available[1].

2. RELATED WORK

In a networked environment, people tend to be influenced by their neighbors' behaviors and decisions [13]. Opinions, product advertisements, or political propagandas could spread over the network through a chain reaction of such influence, a process known as the *information cascade* [38, 5, 1]. We present the first deep learning method to predict the future size of information cascades.

2.1 Cascade Prediction

Cascades of particular types of information are empirically proved to be predictable to some extent, including Tweets/microblogs [43, 39, 20, 11, 17, 44], photos [10], videos [2] and academic papers [31]. In literature, cascade prediction is mainly formulated in two ways. One treats cascade prediction as a classification problem [39, 20, 10, 11], 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 [39, 35]. This line of work can be further categorized by whether it outputs the final size of a cascade [44] or the size as a function of time (i.e., the growth of the cascade) [43]. Either way, most of the methods identified temporal properties, topological structure of the cascade at the early stage, source 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) [23, 3] or stochastic processes that explain the microscopic actions of passing along the information [43]. 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 hand-crafted features, which are extracted from the data, combined, and weighted by discriminative machine learning algorithms to perform the classification or the regression tasks [39, 10, 20, 11]. Most work in this fashion uses learning methods without end-to-end training, whose performance heavily relies on the quality of the features. In general, there is no principled and systematic way to design these features. Some most predictive features are tied to particular platforms or particular cascades and are hard to be generalized, such as the ones mentioned in the introduction. Some features are closely related to the structural properties of the social network, such as degree [10, 39], density [10, 17], and community structures [39]. 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 [14] and how to detect structural holes [42]. 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 arbitrary feature design. We focus on the structures (including node identities) of cascades as content features are reported to be much weaker predictors than structural features [10]. Using temporal signals to predict future trend is a standard problem in time series, and these signals might not be available under certain circumstances, where only cascade snapshots can be observed.

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., [29, 34, 16]). Rather than learning the representation of each node, recent work also attempts to learn the representation of subgraph structures [28, 26, 40, 41]. Much of this work is inspired by the huge success of representation learning and deep learning applied to various domains such as text [4] and image [21]. For example, DeepWalk [29] 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 [25]. 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 [7, 15, 32]. For example, the Weisfeiler-Lehman subtree kernel (WL) [32] computes the graph similarity based on the sub-trees in each graph. Some studies have applied deep learning techniques to improving graph kernels [40, 27]. 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 [28], spectral methods [12], and heat kernel signatures [24], which could then 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. As the

[1] https://github.com/chengli-um/DeepCas, retrieved on February 19, 2017.

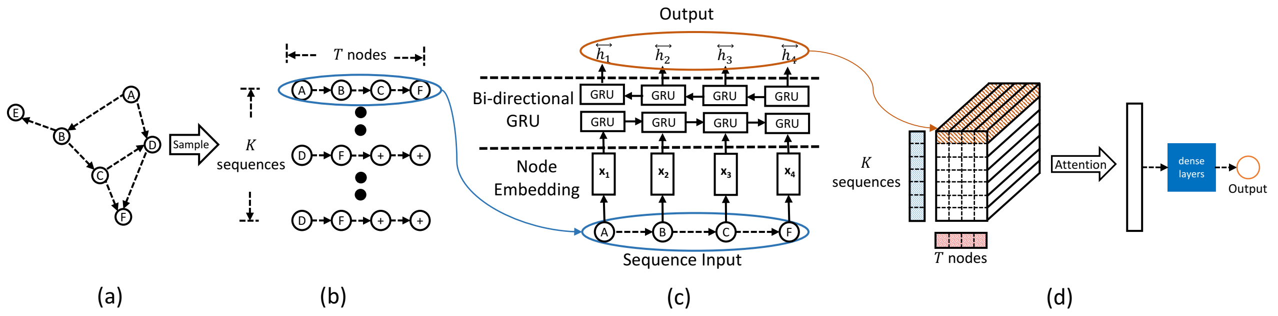


Figure 1: The end-to-end pipeline of DeepCas.

expense, it is hard to incorporate the identities of nodes, which are critical to the prediction of cascades. Indeed, a remarkable study recently shows that mentioning certain users in a Tweet could lead to a bursting diffusion of that Tweet [37].

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.

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 [10] (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.

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.

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 is to find a function f that maps g_c^t to Δs_c, f: $g_c^t \to \Delta s_c$. 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.

3.2 DeepCas: the End-to-End Pipeline

We propose an end-to-end neural network framework that takes as input of the cascade graph g_c and predicts the increment of cascade size Δs_c. The framework (shown in figure 1) first samples node sequences from cascade graphs and then feeds the sequences into a gated recurrent neural network, where attention mechanisms are specifically designed to learn how to assemble sequences into "documents," so that the future cascade size could be predicted.

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 [10]. 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 2. The random walk for each diffusion graph starts from the starting state S, which is always followed by the state N, where the walker transits to a neighbor of the current node. With probability $1 - p_j$, it goes on walking to the neighbor. With a jumping probability p_j, it jumps to an arbitrary node in the cascade graph, leading the walker to the jump state J. With continuation probability p_o, it walks to a neighbor of the current node, thus going back to state N. With probability $1 - p_o$, it goes to the terminal state T, terminating the entire random walk process.

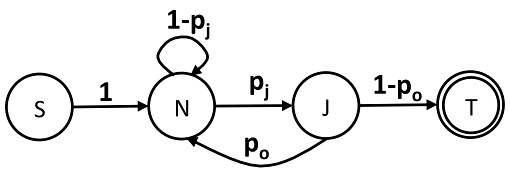


Figure 2: The Markov chain of random walk.

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 sampled in proportion to its score:

$$p(u \in N_c(v)|v) = \frac{\mathrm{sc_t}(u) + \alpha}{\sum_{s \in N_c(v)}(\mathrm{sc_t}(s) + \alpha)}, \tag{1}$$

where α is a smoother. The scoring function $\mathrm{sc_t}(u)$ could be instantiated by but is 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:

$$p(u) = \frac{\mathrm{sc_j}(u) + \alpha}{\sum_{s \in V_c}(\mathrm{sc_j}(s) + \alpha)}, \tag{2}$$

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

3.4 Sampling Sequences from a Graph

The probability p_o of whether to perform another random jump or to 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 to 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 into our deep learning framework. To do this, as Figure 1 (b) shows, for all cascade graphs we sample a sufficient number of sequences that are long enough. 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 that are actually needed for each graph g_c, essentially a different parameterization of p_o and p_j.

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.

An example showing how to sample from a graph is displayed in Figure 1 (a) and (b). For each sequence, the starting node is sampled with probability according to Equation 2 without replacement. When all nodes are sampled once as the starting node, a new iteration of sampling starts. Following the starting node, neighbors are sampled iteratively with probability according to Equation 1. The sampling of one sequence is stopped either when we reach the predefined length T, or when we reach a node without any outgoing neighbors. In this case, sequences with length less than T nodes are padded by a special node '$+$.' This process of sampling sequences continues until we sample K sequences.

3.5 Neural Network Models

Once we have sampled K sequences with T nodes for each diffusion graph, any effective neural network 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) [19], a specific type of recurrent neural network (RNN), which is known to be effective for modeling sequences. 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 $h_i = \mathrm{GRU}(x_i, h_{i-1}), h_i \in \mathbf{R}^H$ by:

$$\begin{aligned} u_i &= \sigma(W^{(u)}x_i + U^{(u)}h_{i-1} + b^{(u)}), \\ r_i &= \sigma(W^{(r)}x_i + U^{(r)}h_{i-1} + b^{(r)}), \\ \hat{h}_i &= \tanh(W^{(h)}x_i + r_i \cdot U^{(h)}h_{i-1} + b^{(h)}), \\ h_i &= u_i \cdot \hat{h}_{i-1} + (1 - u_i) \cdot h_{i-1}, \end{aligned} \tag{3}$$

where $\sigma(.)$ is the sigmoid activation function, $\cdot$ represents an element-wise product. $W^{(u)}, W^{(r)}, W^{(h)}, U^{(u)}, U^{(r)}, U^{(h)} \in \mathbf{R}^{H \times H}$ and $b^{(u)}, b^{(r)}, b^{(h)} \in \mathbf{R}^H$ are GRU parameters that are learned during training and H is the hidden size.

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_{fwd} that reads the sequence from left to right, and a backward GRU_{bwd} that reads from right to left. As Figure 1 (c) shows, the presentation 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:

$$\begin{aligned} \overrightarrow{h}_i^k &= \mathrm{GRU}_{fwd}(x_i, \overrightarrow{h}_{i-1}^k), \\ \overleftarrow{h}_i^k &= \mathrm{GRU}_{bwd}(x_i, \overleftarrow{h}_{i+1}^k), \\ \overleftrightarrow{h}_i^k &= \overrightarrow{h}_i^k \oplus \overleftarrow{h}_i^k, \end{aligned} \tag{4}$$

where $\oplus$ denotes the concatenation operation.

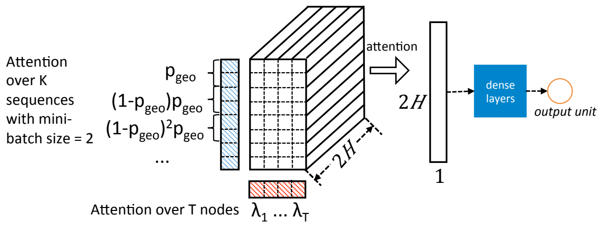

Attention over K sequences with mini-batch size = 2
p_{geo}
$(1\text{-}p_{geo})p_{geo}$
$(1\text{-}p_{geo})^2p_{geo}$
...
attention
$2H$
$2H$
1
dense layers
output unit
Attention over T nodes
$\lambda_1 \dots \lambda_T$

Figure 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 $[\overleftrightarrow{h}_1^k, ..., \overleftrightarrow{h}_i^k, ... , \overleftrightarrow{h}_T^k]$, as displayed in Figure 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 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_{geo}^c, the next mini-batch will have attention $(1-p_{geo}^c)p_{geo}^c$, so on and so forth as Figure 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 $1/p_{geo}^c$. With this expectation, learning the parameter p_{geo}^c 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_{geo}^c 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_{geo}^c on the size of graph $\mathrm{sz}(g_c)$, more specifically $\lfloor \log_2(\mathrm{sz}(g_c) + 1) \rfloor$, where $\lfloor\cdot\rfloor$ takes the floor of a floating number. As a result, p_{geo}^c is replaced with $p_{geo}^{\lfloor \log_2(\mathrm{sz}(g_c)+1)\rfloor}$.

We could apply similar procedure to learn sequence length. In practice, we found that the standard multinomial distribution of attentions already works 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 the $(\lfloor k/B \rfloor + 1)$-th mini-batch. The attention mechanism then outputs the representation for graph g_c, a vector of length $2H$:

$$h(g_c) = \sum_{k=1}^{K}\sum_{i=1}^{T}\left((1-a_c)^{\lfloor k/B\rfloor} a_c\right)\lambda_i \overleftrightarrow{h}_i^t, \qquad (5)$$

where the term in the big parentheses 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 one fully connected layer with one final output unit: $f(g_c) = \mathrm{MLP}(h(g_c))$, where MLP stands for a multi-layer perceptron.

4. EXPERIMENT SETUP

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

4.1 Data Sets

Most existing work evaluates their methods of predicting diffusions on a single social network data set (e.g., [10, 11, 18]). 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 [30], 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 retweet 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.

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 [30] 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 [2]. 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. 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 1) and apply a logarithm transformation of the outcome variable (increment of cascade size), following existing literature [22, 35].

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 [35, 43, 22]. Denote $\hat{y}$ a prediction value, and y the ground truth value, the MSE is:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \qquad (6)$$

As noted in Section 3.1, we predict a scaled version of the actual increment of the cascade size, i.e., $y_i = \log_2(\Delta s_i + 1)$.

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 [10, 18, 11, 36]. These features include:

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

Table 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
Avg. scaled growth Δt_1	train	1.1	0.6	0.5	1.8	2.2	2.0
	val	1.4	0.9	0.7	1.9	2.0	1.8
	test	1.3	0.8	0.7	1.7	2.0	1.8
Avg. scaled growth Δt_2	train	1.6	1.2	1.1	2.5	3.0	2.8
	val	2.1	1.6	1.3	2.7	2.8	2.6
	test	1.9	1.4	1.3	2.4	2.8	2.5
Avg. scaled growth Δt_3	train	1.9	1.5	1.3	3.0	3.5	3.2
	val	2.4	1.9	1.7	3.1	3.3	3.0
	test	2.2	1.8	1.6	2.9	3.2	2.8

Avg. scaled growth scales the actual increment of the cascade size s to $\log_2(\Delta s + 1)$
Δt_1, Δt_2, Δt_3 are respectively 1,3,5 days for Tweets, and 1,2,3 years for papers.

Centrality and Density. Degree of nodes in the cascade graph g and the global network $\mathcal{G}$, the mean and the 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 [6], the overlap of communities, and Gini impurity of communities [18].

Substructures. We count the frequency of k-node substructures ($k \leq 4$) [36]. 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 worse on our task. To obtain an even stronger baseline, we feed the feature vectors to a multi-layer perceptron (MLP), denoted as **Features-deep**.

OSLOR selects important nodes as sensors, and predict the outbreaks based on the cascading behaviors of these sensors [11].

Node2vec [16] is selected as a representative of node embedding methods. Node2vec is a generalization of DeepWalk [29], which is reported to be outperforming alternative methods such as DeepWalk and LINE [34]. 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 [8] 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 [28]. We apply PSCN to both the cascade 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 [28]: the shortest-path kernel (SP) [7], the random walk kernel (RW) [15], and the Weisfeiler-Lehman subtree kernel (WL) [32]. The RW kernel and the SP kernel are too computationally expensive, 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 in [28, 41]. 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 graph 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 regressions, 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 [28] 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 [16], 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 $\{5, 10, 15, 20\}$.

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 these parameters. As described in Section 3.4 and 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 100 and 50 respectively. The embeddings are initialized by concatenating embedding learned by Node2Vec from both all cascade graphs $\{g\}$ in training set and the global network $\mathcal{G}$. The node2vec hyper-parameters p and q are 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 the root of the cascade, 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. EXPERIMENT RESULTS

5.1 Overall performance

The overall performance of all competing methods across data sets is displayed in Table 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 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 have already found a set of good features. The benefit of deep learning really comes from end-to-end learning from the data to the predictions.

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 ids 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 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 better. In a way, this is related to the "mysterious" feature used in Cheng et al. [10], 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 earliness 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.

5.2 Computational Cost

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 for the DeepCas model to converge.

5.3 Error Analysis

We also investigate cascades for which DeepCas makes more mistakes than the baselines, and also the other way around. Here we use the strongest baseline, Features-linear as our reference. The procedure is as follows: among graphs where DeepCas has smaller MSE than the baseline (and the other way around), we select the top 100 with the largest MSE differences between the two methods. For these top graphs, we compute the average of certain network properties of the cascade graphs. This average is taken across configurations, as we do not observe significant differences between configurations.

Table 3: Statistics of graphs for which DeepCas outperforms Features-linear (first rows), and vise versa (second rows). The third row of each data set shows the average statistics on its test set.

Data set	Method	# nodes	# edges	Avg. degree	Edge density
	DeepCas	113.27	689.81	6.09	0.36
TWITTER	Features-linear	89.94	455.78	4.22	0.25
	Avg.	63.16	231.26	3.18	0.30
	DeepCas	13.89	36.97	2.25	0.48
AMINER	Features-linear	12.72	35.51	2.23	0.48
	Avg.	12.53	33.35	2.21	0.49

As Table 3 shows, DeepCas tends 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. On the other hand, both methods perform reasonably well on smaller graphs.

5.4 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 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 [6], 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 dis-

Table 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	**1.480*** **	1.997***	2.074***	1.013***	**1.567*** **	1.735***	0.854***	**1.322*** **	1.422***
DeepCas-deg	1.492***	**1.933*** **	**2.033*** **	1.039***	1.597***	**1.707*** **	0.854***	1.330***	**1.412*** **
DeepCas-DEG	1.487***	2.124***	2.081***	**1.012*** **	1.644***	1.724***	**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	**1.668* **	**2.016** **	**2.084* **	1.545***	**1.693*** **	1.799***	**1.402*** **	1.477***	1.548***
DeepCas-deg	1.684*	2.043*	2.113*	1.544***	1.716***	**1.792*** **	1.407***	**1.469*** **	1.545***
DeepCas-DEG	1.685*	2.036*	2.107*	**1.540*** **	1.700***	1.788***	1.404***	1.480***	**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).

tribution 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 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 [9], 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 4(g) with 4(i), we can see that the number of communities in a cascade graph is indeed positively correlated with its growth.

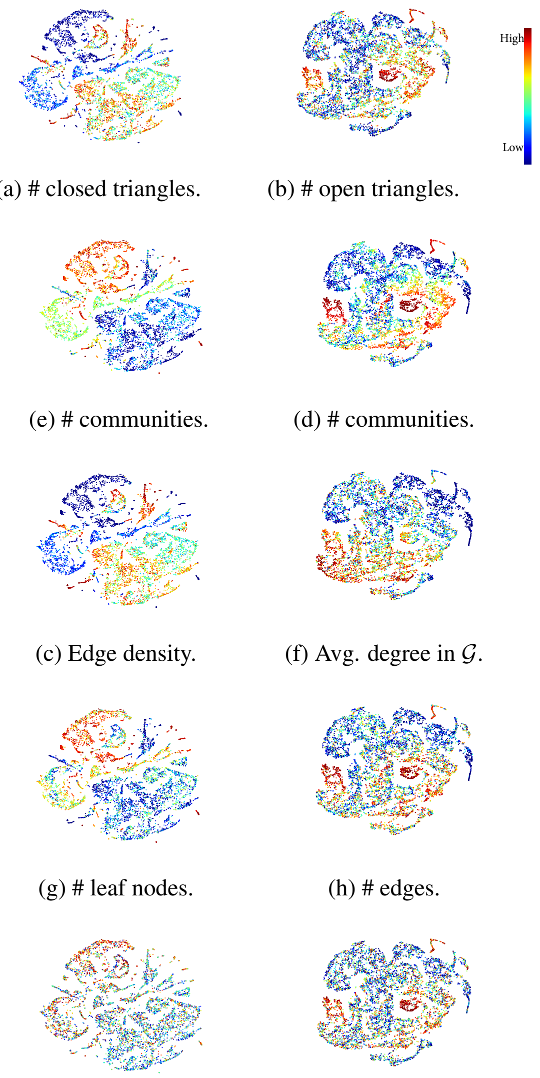
Figure 4 (f) plots the average global degree of nodes in each cascade graph. The pattern suggests that DeepCas not only captures the structural information from individual cascade graphs, but also incorporates the global information into the graph representation. How did this happen? Although we did 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 4. First, as the number of open and closed triangles are actually important features used for graph prediction tasks [36], 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 4, but this is true), but also their none-linear relationship that involves division.

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 domain, social networks, we do wish to point the readers to a few more interesting implications. First, we find that linearly combined, hand-crafted features per-



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

(e) # communities. (d) # communities.

(c) 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 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.

form 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 worse 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.

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.

Acknowledgment

This work is partially supported by the National Science Foundation under grant numbers IIS-1054199 and SES-1131500.

7. REFERENCES

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

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

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

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

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

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

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

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

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

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

[11] 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.

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

[13] D. Easley and J. Kleinberg. Networks, crowds, and markets. *Cambridge University Press*, 2010.

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

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

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

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

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

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

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

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

[22] 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.

[23] 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.

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

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

[26] 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.

[27] 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.

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

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

[30] 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.

[31] 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.

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

[33] 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.

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

[35] 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.

[36] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *PNAS*, 2012.

[37] B. Wang, C. Wang, J. Bu, C. Chen, W. V. Zhang, D. Cai, and X. He. Whom to mention: expand the diffusion of tweets by @ recommendation on micro-blogging systems. In *22nd International World Wide Web Conference*, pages 1331–1340, 2013.

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

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

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

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

[42] 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.

[43] 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.

[44] 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.