A Method for Learning Representations of Signed Networks

Inzamam Rahaman
Department of Computer Science
The University of the West Indies
St. Augustine, Trinidad and Tobago
inzamam@lab.tt

Patrick Hosein
Department of Computer Science
The University of the West Indies
St. Augustine, Trinidad and Tobago
patrick.hosein@sta.uwi.edu

ABSTRACT

Recently there has been significant interest in low dimensional representations of graphs that can then be exploited by machine learning and data mining techniques. The geometric relationships within these learned representations should reflect those between nodes in the original graph. Most work has concentrated on unsigned graphs, which only model positive relationships. However, such techniques can be inadequate for signed graphs, which model both positive and negative relationships. In this work in progress paper, we present a method - StEM (Signed neTwork Embedding Model) - for learning representations of signed networks that achieves improved performance on tasks such as visualization, node classification and signed link prediction.

KEYWORDS

Network Embeddings, Latent Representations, Feature Learning, Signed Networks

ACM Reference Format:
Inzamam Rahaman and Patrick Hosein. 2018. A Method for Learning Representations of Signed Networks. In *Proceedings of International Workshop on Mining and Learning on Graphs (MLG2018).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

There are several biological [14, 39], social [12, 29], and technological [6, 35] systems that can be expressed as graphs. In such systems the individual actors or agents comprise the nodes and their relationships/interactions comprise the edges of the graph. Given the ubiquity of data that can be represented as a graph, there has been growing interest in developing data mining and machine learning techniques that can operate on data that has been structured as a graph or a network. (Note that the terms graph and network are used interchangeably in this paper.)

Given the diversity of systems that can be decomposed into actors and their interactions, there are several graphical models that can be applied. Unsigned networks are the most well-studied of these models. In unsigned networks, all nodes are drawn from the same overall set of objects and all edges denote the same type of bi-directional relationship. The quintessential unsigned network is the Online Social Network known as Facebook [13] where users are nodes and bidirectional friendship relationships are represented by edges. Although the edges of an unsigned network can be assigned numeric weights to express the strength of the relationships they represent, the edges all denote the same sort of relationship - typically a positive one.

While many datasets can be adequately modeled as unsigned networks, others require more nuance to be represented than simply the existence of a relationship. In many systems agents can have interactions with one another that can be deemed either positive or negative [48] and these are better represented as a signed network. In a signed network, nodes are drawn from a single set of objects, but edges may be denoted as either positive - indicating a friendly relationship - or negative - indicating an antagonistic relationship.

Graphs are becoming an increasingly popular way to model structured data in order to perform machine learning and data mining tasks. These tasks include link prediction [33], clustering [38], node label classification [4], anomaly detection [2], visualization [34], and recommendation [23, 32]. To perform machine learning and data mining on graphs, we compute features of the nodes and edges that constitute the graph. These computed features can then be fed into techniques such as logistic regression and K-means clustering to predict and analyze properties and structures of the graphical data under consideration. However, this process of feature/representation engineering is a time-consuming and tedious process that can require the input of a domain expert. Furthermore, the features obtained during manual feature detection can be tightly coupled to the downstream task and even the data itself.

Representation learning is an alternative to manual feature engineering. In representation learning, instead of manually designing features, we design a procedure that can learn features of the objects under study for a particular type of dataset. The methodology of representation learning has yielded good results in domains such as natural language processing [36].

Representation learning on graphs usually takes one of two approaches. The first involves factorizing a matrix encoding of the graph. The second approach involves designing a neural embedding model that solves an auxiliary task related to the observed topology of the graph. Many unsigned graphs exhibit a property called homophily whereby edges are formed between neighbours having similar properties. Consequently, the representation learned for a particular node should reside close in the vector space to the representations of its neighbors. However, homophily in a signed graph is primarily observed among positive neighbors [49]. Hence, if an embedding technique designed for unsigned networks is applied to a signed network, then nodes may be placed in close proximity with its foes - this is undesirable. We first present a neural embedding model for signed networks - StEM. We then compare this model with another state of the art model called SiNE [52]. Comparisons

MLG2018, August 2018, London, UK

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

are made for graph visualization, node classification, and signed link prediction tasks. Advantages include similar or better performance, reduced training time and fewer hyper-parameters.

2 PROBLEM DEFINITION

In this section we provide the signed networks model first introduced by Wang et al. [52] and Dong et al.[9]. We also draw from Tang et al.'s [48] definition of a signed network.

Definition 2.1. ***Signed Network:*** *A signed network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an ordered pair of sets, where $\mathcal{V}$ is the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \{-1, 1\}$ is the set of edges. Note that for every triple in $\mathcal{E}$, the last component represents the sign of the edge where -1 indicates a negative edge and 1 indicates a positive edge. In addition, we let $\mathcal{E}_- = \{(u, v) \mid (u, v, s) \in \mathcal{E} \text{ and } s = -1\}$ be the set of negative edges and likewise we let $\mathcal{E}_+ = \{(u, v) \mid (u, v, s) \in \mathcal{E} \text{ and } s = 1\}$ be the set of positive edges. We let $|\cdot|$ be the standard cardinality operator, and as such as we let $N = |V|$ be the number of nodes in $\mathcal{G}$.*

By considering a signed network as input, we formally define the problem of feature/representation learning in signed networks as follows.

Definition 2.2. ***Representation Learning in Signed Networks:*** *Given a signed network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and number of dimensions $d \in \mathbb{Z}_+$, our task is to learn a function $f : \mathcal{V} \to \mathbb{R}^d$, where $d << N$, such that the structural relationships contained in $\mathcal{E}$ are reflected in the representations output by f.*

Note that the function f can take many forms. In StEM, we let f be the output of a hidden layer of a neural embedding model. We define a neural embedding model below:

Definition 2.3. ***Neural Embedding Model:*** *Given a set $\mathcal{D}$ of size N indexed on $[1, N]$ and $d \in \mathbb{Z}_+$, a neural embedding model is a neural network designed to learn a weight matrix $W \in \mathbb{R}^{N \times d}$, where $d << N$. In W, $W_{n,:}$ (the n^{th} row of the matrix W) is the representation learned for item with index n. We can learn W by solving some sort of auxiliary task using relationships observed in $\mathcal{D}$.*

Hence, our objective is to design a neural embedding model that can be used to learn representations for signed networks. In the following section, we describe our proposed approach: StEM.

3 DESCRIPTION OF PROPOSED METHOD

Drawing from the our insights on homophily, we would expect that most properties of interest in a signed network would be related to the separation of opposing subgroups in said signed network. Consequently, a good representation learning method for signed networks should result in these subgroups being well separated from one another. Suppose, in addition, we also learn a decision boundary for each node that separates the node's friends from its foes. Jointly learning these representations with associated decision boundaries would allow us learn representations that capture global information related to the separation of opposing groups. We suspect that the representations learned by such an approach would be of higher quality than those representations learned by using a distance based ranking approach that considers only local information.

3.1 Mathematical Formulation

Recall that, in a signed network, we have two edge types: positive and negative. Given a signed network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we can formulate the task of learning our representation function f^* as a maximum likelihood problem as described below:

$$\max_{f,\theta} \prod_{(u,v,s)\in\mathcal{E}} Pr(y = \mathbb{I}_{\{s=1\}} \mid u, v, \theta, f) \tag{1}$$

where $\mathbb{I}$ is the indicator function, $f : \mathcal{V} \to \mathbb{R}^d$ is our representation function that maps nodes to points in a d-dimensional vector space, and θ is the set parameters of the probability function. By taking logs and multiplying by -1, Equation (1) becomes:

$$\min_{f,\theta} \sum_{(u,v,s)\in\mathcal{E}} -log\ Pr(y = \mathbb{I}_{\{s=1\}} \mid u, v, \theta, f) \tag{2}$$

To make (2) tractable, we need to impose restrictions on both the forms of Pr and f. First, consider the form of Pr. Since the codomain of Pr is $[0, 1]$, we let Pr be the following:

$$Pr(x \mid u, v, \theta, f) = \begin{cases} \sigma(\zeta(u, v; \theta, f)) & x = 1 \\ 1 - \sigma(\zeta(u, v; \theta, f)) & x = 0 \end{cases} \tag{3}$$

where σ is the logit function. Note that we introduce a function $\zeta : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ that acts as a "measure" of relative similarity of two nodes. We let ζ be a function of the following form:

$$\zeta(u, v; \theta, f) = \varphi(f(u); \beta) M_2 \varphi(f(v); \beta)^T + b_2 \tag{4}$$

where $u, v \in \mathcal{V}$, $f(u), f(v) \in \mathbb{R}^d$ are feature (row) vectors for u and v respectively, $M_2 \in \mathbb{R}^{d \times d}$ is a learnable matrix of weights, $b_2 \in \mathbb{R}$ is a learnable bias, $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ is a parameterizable activation function, and β is a learnable parameter of φ. Note that $b_2, M_2,$ and $\beta \in \theta$, and that (4) may not be symmetric with respect to its arguments, i.e $\varphi(f(u); \beta) M_2 \varphi(f(v); \beta)^T + b_2 \neq \varphi(f(v); \beta) M_2 \varphi(f(u); \beta)^T + b_2$. Equation (4) captures the intuitions we expressed at the start of Section 3. We interpret $\varphi(f(u); \beta) M_2$ as computing a decision boundary that is specific to u that allows us to separate u's friends from u's foes. If u and v are friends, $\zeta(u, v; \theta, f)$ should be positive, thereby leading to $Pr(x = 1 \mid u, v, \theta, f) > Pr(x = 0 \mid u, v, \theta, f)$. Likewise, if u and v are enemies, $\zeta(u, v; \theta, f)$ should be negative, thereby leading to $Pr(x = 1 \mid u, v, \theta, f) < Pr(x = 0 \mid u, v, \theta, f)$. We let φ be a single-channeled parameterized leaky rectified linear unit as defined by He et al. [22]. Its definition is shown below:

$$\varphi(x[i]; \beta) = \max(0, x[i]) + \beta \min(0, x[i]) \tag{5}$$

where $x[i]$ is the i^{th} component of vector x. When describing passes through the neural network, we call this function $PLeakyReLU(x, \beta)$.

With the structure of the probability function outlined, we can now turn our attention to the form of the representation function f. We define the structure of $f : \mathcal{V} \to \mathbb{R}^d$ as follows:

$$f(u) = (M_1 g(u)^T + b_1)^T \tag{6}$$

$$g(u) = W_{u,:} \tag{7}$$

where $u \in \mathcal{V}$, $M_1 \in \mathbb{R}^{d \times 2d}$, $W \in \mathbb{R}^{N \times 2d}$, and $b_1 \in \mathbb{R}$ are all learnable parameters. We let $W_{u,:}$ be the u^{th} row of matrix W.

Given the above, we can design a feed-forward neural network that accepts the indices of two nodes, u and v, and outputs the

probability of a positive edge existing from u to v. We describe a forward pass through the network as below. Note that we have highlighted the learnable parameters of the model. From this point on, we shall refer to the set of learnable parameters in our model as Θ.

$$\begin{aligned} r_u &= \boldsymbol{W}_{u,:} & r_v &= \boldsymbol{W}_{v,:} \\ x_u &= (\boldsymbol{M_1} r_u^T + \boldsymbol{b_1})^T & x_v &= (\boldsymbol{M_1} r_v^T + \boldsymbol{b_1})^T \\ q_u &= PLeakyReLU(x_u, \boldsymbol{\beta}) & q_v &= PLeakyReLU(x_v, \boldsymbol{\beta}) \\ z &= q_u \boldsymbol{M_2} q_v^T + \boldsymbol{b_2} & p &= \sigma(z) \end{aligned}$$

Recall that our representation function is expressed in the above forward pass. To extract the representation for a node u, we only need to make a partial forward pass through the network as follows:

$$\begin{aligned} r_u &= \boldsymbol{W}_{u,:} \\ x_u &= (\boldsymbol{M_1} r_u^T + \boldsymbol{b_1})^T \end{aligned}$$

3.2 Training StEM

3.2.1 The Loss function. With the structure of our neural network defined, we can now restate our objective in Equation (2) as a loss function in terms of the output of our neural network. Note that in the following, p denotes a paramaterizable function encoding the structure of the neural embedding model described above, and $p(u, v; \Theta)$ is the application of neural network p with the parameter list Θ to the edge from u to v to determine the probability of it denoting a positive relationship; consequently, since we have only two types of edges, $1 - p(u, v; \Theta)$ is the probability of the edge from u to v denoting a negative relationship.

$$\mathcal{L}(\mathcal{E}, p, \Theta) = -\frac{1}{|\mathcal{E}|} \sum_{(u,v,s)\in\mathcal{E}} \mathbb{I}_{\{s=1\}} \log(p(u, v; \Theta)) + (1 - \mathbb{I}_{\{s=1\}}) \log(1 - p(u, v; \Theta)) \tag{8}$$

To prevent overfitting, we apply standard L2 regularizations on the parameters contained in Θ, to obtain:

$$\mathcal{L}(\mathcal{E}, p, \Theta) = -\frac{1}{|\mathcal{E}|} \sum_{(u,v,s)\in\mathcal{E}} \mathbb{I}_{\{s=1\}} \log(p(u, v; \Theta)) + (1 - \mathbb{I}_{\{s=1\}}) \log(1 - p(u, v; \Theta)) + \lambda \sum_{\theta\in\Theta} ||\theta||_2 \tag{9}$$

where $0 < \lambda << 1$ is our regularization constant that controls the degree of regularization we apply during training.

3.2.2 Initialization and Mini-batching. Prior to training the network, we need to randomly generate values for its parameters. During our development of StEM, we found that Xavier initialization [15] worked well in practice for W, M_1, M_2, b_1, and b_2. We initialized β to 0.

As noted by Leskovec et al. [30], there can be an imbalance in the number of positive edges versus the number of negative edges in a signed network. Consequently, we used undersampling during each epoch to ensure that we sampled equal numbers of positive and negative edges to prevent the signal contributed by one edge type drowning out the signal contributed by the edges of the opposite type.

3.2.3 The Training Algorithm. There are several variants of stochastic gradient [46] that can be used to train neural networks, including ADAM [25], AdaDelta [55], and AdaGrad [11]. We empirically tested these methods and found that our neural networks encoding StEM converged faster using AdaGrad.

3.3 Complexity of StEM

Consider that we are learning vectors of size d. Making a single pass through the network would have computational complexity $O(d^3)$. Backpropagation would also incur a computational complexity of $O(d^3)$. In addition, we would make both forward and backward passes through the neural network element in a batch. Since we undersample our edges then the size of each mini-batch is $E = \min\{|\mathcal{E}_+|, |\mathcal{E}_-|\}$. Hence, a single epoch has complexity $O(Ed^3)$. If we iterate for t epochs, then the computational complexity of training our model is $O(tEd^3)$.

4 EXPERIMENTS

We learn graph representations for use in downstream machine learning and data mining tasks. In this section, we describe the experiments we conducted to examine the effectiveness of StEM in learning useful representations for signed graphs. We consider three tasks: visualization, node classification, and signed link prediction.

To benchmark our results, we compare representations learned by StEM with representations learned by Wang et al.'s SiNE [52]. In our experiments, we set the hyperparamters to the values indicated by Wang et al. Note that several datasets used in our experiments were complete graphs, i.e. every node in the graph is connected to every other node. This would obviate the need to incorporate the virtual node used by SiNE. For such datasets, we used SiNE/P_0 and indicate accordingly.

We implemented both StEM and SiNE in Python 3.6 using PyTorch 0.30 [40] and the standard scientific Python stack [18, 24, 41, 50]. Both our implementation of StEM and SiNE are publicly available[1].All experiments were run on a 2015 MacBook Pro with a Core i5 processor and 8 GB of RAM. No GPUs were used in our experiments.

4.1 Datasets and Data Processing

We used several datasets throughout our experiments. The BitcoinOtc and BitcoinAlpha [28], Epinions [30], Wiki-Rfa [54] and Slashdot [30] datasets were downloaded from the SNAP [2] [31] repository. The US Senate and House roll-call votes [53] dataset were downloaded through the ICON [3] [8] repository. The Tribes [43] dataset was reconstructed from tables presented by Doreian and Mrvar [10].

While many of the datasets were structured as explicit signed networks, Wiki-Rfa, Senate-104, House-102, House-103, House-104, House-105, House-106, and House-107 were not originally encoded as explicit signed networks. We transformed the Wiki-Rfa dataset into a signed network by creating a positive edge between user u

[1] https://github.com/InzamamRahaman/StEMPublic
[2] snap.stanford.edu
[3] icon.colorado.edu

Dataset	N	$\lvert\mathcal{E}_+\rvert$	$\lvert\mathcal{E}_-\rvert$
BitcoinOtc	5881	32029 (90%)	3563 (10%)
BitcoinAlpha	3783	22650 (96%)	1536 (4%)
Senate-104	103	4876 (46%)	5630 (54%)
Epinions	131828	717667 (85%)	123705 (15%)
Slashdot	77350	396378 (77%)	120197 (23%)
Wiki-Rfa	11368	144451 (78%)	41176 (22%)
Tribes	16	61 (50%)	61 (50%)
House-102	441	69338 (38%)	124702 (64%)
House-103	442	86302 (44%)	108620 (56%)
House-104	445	90410 (46%)	107170 (54%)
House-105	444	73034 (37%)	123658 (63%)
House-106	440	49588 (26%)	143572 (74%)
House-107	444	38362 (20%)	158330 (80%)

Table 1: Statistics for Signed Network Datasets Used In Experiments

and user v if u supported v's request for adminship and creating a negative edge if u opposed v's request for adminship. We ignored neutral votes.

The voting record datasets, i.e. House-# and Senate-104, can initially be conceived as a bipartite, heterogeneous signed network. In our paper, we consider only homogeneous signed networks. Consequently, the voting record datasets required more processing than the Wiki-Rfa datasets to transform them into signed networks. Each voting dataset provided a matrix encoding every representatives' support or rejection for each bill; moreover, the matrix also encoded whether a representative abstained from voting on a bill. We transformed this matrix into a complete signed network by constructing an undirected complete graph. The sign for an edge between nodes i and j is denoted as $label_{ij}$; $label_{ij}$ is computed as follows:

$$u_i[a] = \begin{cases} v_i[a] & \text{if } v_j[a] \neq 0 \\ 0 & \text{otherwise} \end{cases} \qquad u_j[a] = \begin{cases} v_j[a] & \text{if } v_i[a] \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$label_{ij} = \begin{cases} 1 & \text{if } H(u_i, u_j) \leq \delta \\ -1 & \text{otherwise} \end{cases}$$

where $v_y[x]$ is the vote of senator y on bill x (either 1, 0, or -1), and H is the hamming distance [20] between two vectors. For the House-# datasets, we set $\delta = 0.5$, and for the Senate-104 dataset we set $\delta = 0.45$. We summarize statistics for all datasets in Table 1

Aside from structuring the data as a signed network, StEM requires no extra preprocessing of data. However, SiNE requires the extraction of node triples from the dataset such that each triple contains a positive edge and a negative edge. These triples are computed from $\mathcal{E}$. During some of our experiments, we train both StEM and SiNE on a subgraph of the original signed network. In such cases, the triples extracted for SiNE were extracted from the edges contained in the subgraph of the original graph.

4.2 Data Visualization

A common task in exploratory data analysis is visualization. Using visualization, we can develop insight into the underlying patterns in the data under interrogation. In the case of visualizing graphical data, a challenge arises in deciding how nodes ought to be projected into a 2D space in relation to one another. This is the problem of determining the graph layout [27].

One strategy is to project the nodes of a graph onto a high dimensional space and to then use manifold learning/dimensionality reduction techniques such as MDS [5] and tSNE [34] [21] to project this high dimensional space into a 2D space. Since we would expect related nodes to be close to one another in the feature spaces learned by both StEM and SiNE(or SiNE/P_0), we assert that these learned representations can be used as the high dimensional input into a dimensionality reduction technique for the purposes of visualizing the global relationships between the nodes of signed networks. Note that in a signed network dataset, we should see clusters emerge when the data is projected onto a 2D space. The nodes within these visibly identifiable clusters should be allies. During our visualization experiments, we considered two datasets: Senate-104, and Tribes.

When learning representations for data visualization, we consumed all of the edges. Both methods were run for 50 epochs, with a learning rate of 0.1, a regularization constant of 0.00055, and an embedding dimension of 16. In the case of both SiNE and SiNE/P_0, we used the empirically set parameters outlined by Wang et al.[52].

As noted by Hage and Harray [19], there are three communities within the Tribes dataset. Consequently, when the nodes representing these tribes are visualized in $\mathbb{R}^2$, we should see a clear separation between these three groups. We ran both StEM and SiNE, applied MDS to the resultant representations, and then plotted the results in Figures 1 and 2. Each of the three groups identified by Hage and Harray are color coded differently. Notice that both the representations StEM and SiNE facilitate the projection the data onto a $\mathbb{R}^2$ vector space well, thereby allowing us to discern between the three opposing groups.

In Senate-104, there are two opposing groups of users: Republicans and Democrats. Consequently, when visualized, we expect that we should observe two conspicuous clusters of nodes. To visualize Senate-104 we used the same hyperparameters used to visualize the Tribes dataset; however, to visualize Senate-104, we used tSNE instead of MDS, and SiNE/P_0 instead of SiNE as Senate-104 is a complete graph. The visualizations produced using StEM and SiNE/P_0 are seen in Figures 3 and 4 respectively. While both StEM SiNE/P_0's representations achieve a noticeable separation between Republicans and Democrats, the separation achieved by StEM's representations is more pronounced. We attribute this to Equation (4) encouraging linear separability between a node's friends and foes, which in turn lends itself to the linear separability of different communities in the network.

In addition, we also considered the quality of the visualizations produced as we vary the percentage of data available for training. These visualizations are plotted in Figure 5. We considered three random samples: 60% (first row), 80% (second row), and 100% (third row) of the edges. The plots on the left are the representations learned by StEM, and the plots on the right are the representations learned by SiNE. Notice that with 60% of the data, StEM achieves a good separation between Republicans and Democrats. In contrast, SiNE/P_0 required 100% of the available data to achieve a comparable outcome. This may indicate that StEM might be more robust to missing data than SiNE/P_0.

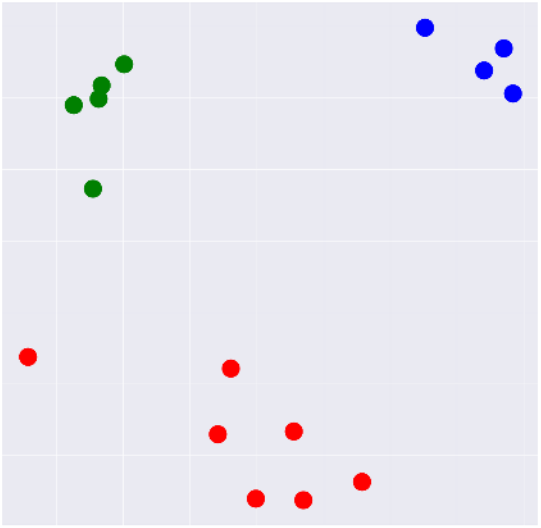

Figure 1: Representations Learned by StEM of Tribes dataset projected onto $\mathbb{R}^2$ by MDS. Each color denotes a different subgroup identified by Hage and Harray [19]

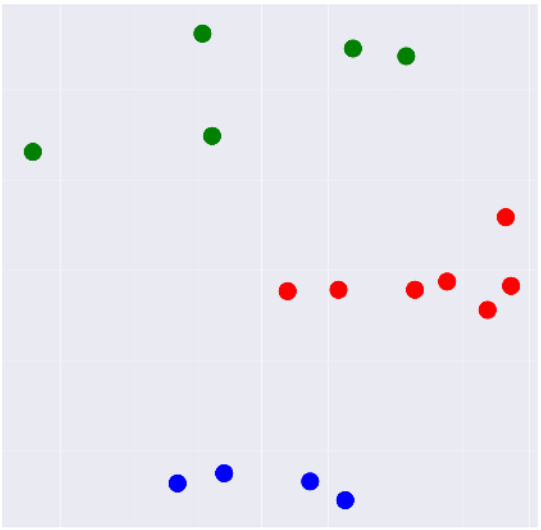

Figure 2: Representations Learned by SiNE of Tribes dataset projected onto $\mathbb{R}^2$ by MDS. Each color denotes a different subgroup identified by Hage and Harray [19]

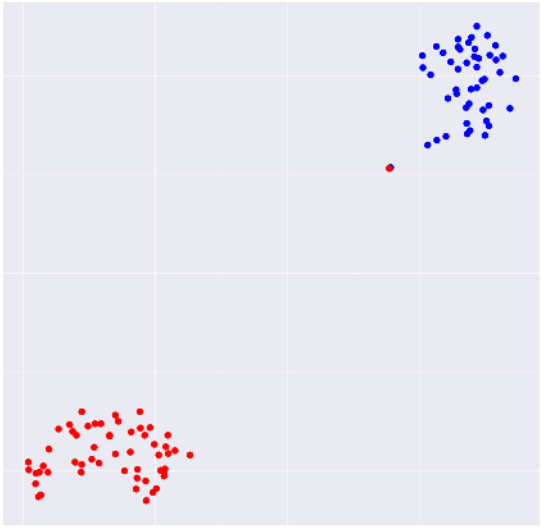

Figure 3: Representations Learned by StEM of Senate-104 dataset projected onto $\mathbb{R}^2$ by tSNE. Red dots are Republicans and blue dots are Democrats.

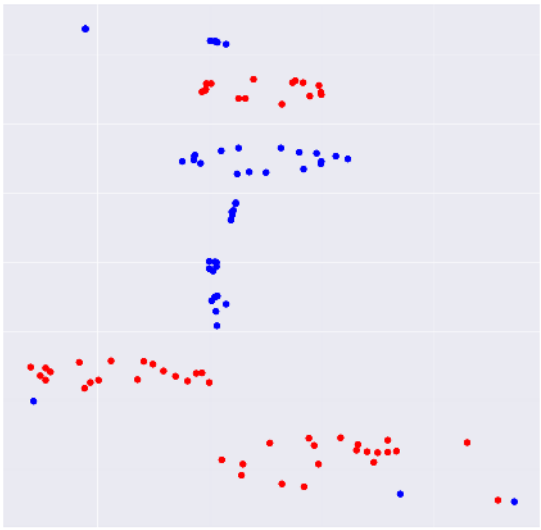

Figure 4: Representations Learned by SiNE of Senate-104 dataset projected onto $\mathbb{R}^2$ by tSNE. Red dots are Republicans and blue dots are Democrats.

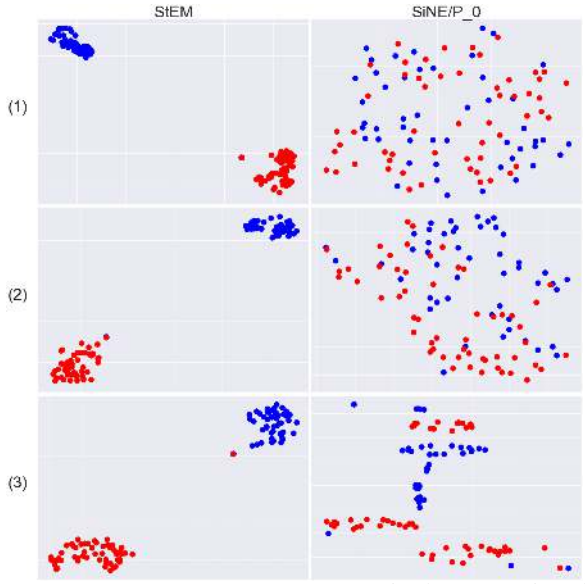

Figure 5: Representations Learned by StEM (left) and SiNE (right) of Senate-104 dataset projected onto $\mathbb{R}^2$ by tSNE for Different Fractions of the Dataset. (1): 60% of dataset; (2) : 80% of dataset; (3): 100% of dataset. Red dots are Republicans and blue dots are Democrats.

Dataset\Method	StEM	SINE/P_0	% improvement
House-102	0.977 (0.012)	0.868 (0.059)	12.6
House-103	0.984 (0.014)	0.909 (0.031)	7.5
House-104	0.982 (0.015)	0.883 (0.054)	11.2
House-105	0.980 (0.085)	0.849 (0.069)	15.4
House-106	0.964 (0.013)	0.528 (0.084)	82.6
House-107	0.977 (0.021)	0.731 (0.083)	24.6

Table 2: Mean (and Standard Deviation) of Micro-F1 Scores Obtained from 5-fold Cross Validation on Different Datasets

4.3 Node Classification

Similar to unsigned networks, the nodes of a signed network may be assigned two or more mutually exclusive labels/classes. The task of node classification is to use the network topology along with a subset of nodes whose labels are known to predict the labels of nodes for whom we lack labels [47]. Since the learned representations would capture information of the network topology, we can train a classifier that takes these representations of their input.

For each dataset, we consumed the entire signed network toplogy to train their representations. The representations learned for nodes were then fed as input into a logistic regression classifier along with a portion of the class labels. We used 5-fold cross validation on our node-labels pairs and averaged their performance as measured using both micro-F1 and macro-F1 scores. We report these results in Tables 2 and 3 respectively. In addition, we also report the standard deviation in the micro-F1 and macro-F1 scores for each case in parenthesis. We trained both methods at a learning rate of 0.15 for 50 epochs to learn representations comprising 16 dimensions. Since all of our datasets usable for node classification were complete graphs, we used SiNE/P_0 instead of SiNE.

As seen in both Tables 2 and 3, both the representations learned by StEM and SiNE/P_0 tend to perform well on node classification.

Dataset\Method	StEM	SINE/P_0	% improvement
House-102	0.976 (0.013)	0.842 (0.077)	15.9
House-103	0.983 (0.014)	0.909 (0.032)	8.1
House-104	0.981 (0.016)	0.880 (0.054)	11.5
House-105	0.979 (0.008)	0.845 (0.070)	15.9
House-106	0.962 (0.014)	0.512 (0.090)	87.9
House-107	0.977 (0.070)	0.723 (0.086)	35.1

Table 3: Mean (and Standard Deviation) of Macro-F1 Scores Obtained from 5-fold Cross Validation on Different Datasets

Operation	Output
Hadamard	$y[i] = x_a[i] \times x_b[i]$
L1	$y[i] =\mid x_a[i] - x_b[i] \mid$
L2	$y[i] = (x_a[i] - x_b[i])^2$
Average	$y[i] = 0.5 \times (x_a[i] + x_b[i])$
Concatenation	$y = x_a \oplus x_b$

Table 4: Binary Operations for Composing Edge Features from Node Features

That being said, StEM outperforms SiNE/P_0 across all the datasets we considered. Excluding the results on the House-106 dataset, StEM's representations performed an average of 14.6% and 17.3% better as measured by Micro-F1 and Macro-F1 scores respectively. Moreover, as can be seen by the low standard deviations we observed, both StEM and SiNE learn representations that lend themselves to robust classifiers.

4.4 Signed Link Prediction

In signed link prediction, we are given a signed network with the signs on several edges unobserved or missing and we would like to use the available annotated edges to predict the missing annotations. This can be formulated as a binary classification problem.

In Grover and Leskovec [17], representations for edges are computed by composing their incident nodes using binary operations. These edge representations are then used to train a logistic regression classifier that predicts whether an edge would form between two arbitrary nodes. We adopted a similar workflow for signed link prediction. The binary operations used are described in Table 4.

Given a signed network dataset, we partitioned the dataset's edges, $\mathcal{E}$, into a training set and a testing set. The training set was used to learn representations using StEM and SiNE. These representations were then composed using a binary function from Table 4 to generate representations for the edges in $\mathcal{E}$. Using the representations for the edges in the training set, we trained logistic regression classifiers to predict edge signs from edge representations. Due to the imbalance in signs in $\mathcal{E}$, we used undersampling when training the logistic regression classifiers. We then evaluated the performance of the trained classifiers on the representations of the edges in the testing set. For every dataset, we performed 5-fold cross-validation using the above procedure. We report the results in Table 5.

As seen in Table 5, StEM outperformed SiNE on all datasets. For the BitcoinOtc, BitcoinAlpha, Slashdot, Epinions, and Wiki-Rfa datasets, we observed an improvement of 18.3%, 20.8%, 10.0%, 24.7%, and 13.8% respectively of StEM's over SiNE. The concatenation operator proved to be the best performer across nearly all datasets

Op	Method	Dataset				
		(1)	(2)	(3)	(4)	(5)
a	StEM	**0.934**	0.950	**0.914**	**0.948**	**0.889**
	SiNE	*0.805*	*0.787*	*0.733*	*0.858*	*0.781*
b	StEM	0.940	**0.951**	0.898	0.916	0.818
	SiNE	0.671	0.675	0.591	0.718	0.624
c	StEM	0.836	0.852	0.703	0.826	0.695
	SiNE	0.784	0.761	0.668	0.779	0.680
d	StEM	0.853	0.859	0.741	0.832	0.701
	SiNE	0.773	0.757	0.665	0.771	0.668
e	StEM	0.894	0.892	0.879	0.874	0.801
	SiNE	0.721	0.652	0.658	0.647	0.711

Table 5: Mean AUC from 5-fold Cross Validation for Comparison between StEM and SiNE. Binary operators used: (a) Concatenation, (b) Hadamard, (c) L1, (d) L2, (e) Average. Datasets Used: (1) BitcoinOtc, (2) BitcoinAlpha, (3) Slashdot, (4) Epinions, (5) Wiki-Rfa

Dataset\Method	StEM	SINE	% improvement
BitcoinOtc	1.57	37.43	95.8
BitcoinAlpha	1.12	17.41	93.6
Slashdot	41.47	97.92	57.7
Epinions	33.27	899.59	96.3
Wiki-Rfa	11.11	181.08	93.9

Table 6: Time Taken (ms) for Training for Different Datasets

for both StEM and SiNE. Hadamard was second best for StEM, and the L1 operator was the second best for SiNE. We believe that Hadamard composed representations performing better than the L1 composed representations for StEM to be a result Equation (4) effects on the representations learned by StEM. We believe that Hadamard composed representations performing worse than the L1 composed representations for SiNE to be a result of SiNE explicitly aiming to ensure that a node's friends are closer to it than its enemies.

4.5 Runtime of StEM vs SiNE

Aside from accuracy, runtime is also an important consideration when evaluating representation learning methods. We ran StEM and SiNE on several datasets and recorded the the time taken to train representations with 16 dimensions for 50 epochs. As seen in Table 6, StEM takes substantially less time than SiNE, and achieved, on average, an 87.5% faster runtimes. Note that we excluded the time incurred in preprocessing the networks into triples for SiNE.

5 RELATED WORK

There has been much interest in developing automated representation learning techniques on networks in the past decade. As noted by by Goyal and Ferrara [16], many of these methods approach the task of representation learning from the angle of matrix factorization, random walk modeling, or deep learning. In factorization methods such as LLE [45], Laplacian Eigenmaps [3], and Graph Factorization [1], the relationships between nodes are encoded in a matrix such as the adjacency matrix or Laplacian matrix. In such methods, the matrix is then factorized while trying to solve an optimization problem, thereby learning representations in the process.

In random walk based approaches, we perform several random walks through the network. These random walks are taken as encoding centrality, proximity, and structural information about the graph that can be exploited to learn representations. Many of these techniques draw inspiration from Milkolov et al.'s word2vec [36]. Notable techniques in this space are Perozzi et al.'s DeepWalk [42], which pioneered modeling networks as linguistic objects for representation learning, and Grover and Leskovec's node2vec [17], which advanced understanding of how different types random walks affect the quality of the representation learned. Some techniques such as Ribeiro et al.'s struct2vec [44] have considered more advanced re-imaginings of a random walk. In particular, struct2vec uses a random walk on a layered multi-graph to better capture information related to the structural roles of nodes rather than their proximity relationships.

Deep Learning based techniques, such as GCNs [26] and SDNE [51] adapt representation learning techniques originally created for other domains to the task of learning representations on networks. GCNs and SDNE exploit convolutions and deep autoencoders respectively.

Given common peculiarities of graphical structures, there has also been interest in moving from representing nodes of a network in Euclidean space and targeting representation spaces such as Hyperbolic space [7, 37].

There has also been interest in learning representations for more nuanced network types, such as Dong et al.'s metapath2vec [9], and most relevantly to this paper, Wang et al.'s [52] SiNE. SiNE addresses the problem of representing signed networks as a ranking problem - the endpoints of positive edges should rank more highly with respect to one another than the endpoints of negative edges in terms of a learned similarity function.

6 DISCUSSION

In this paper, we present ongoing work on StEM, a method for learning representations of nodes in signed networks. We have shown preliminary results that StEM separates opposing groups in networks well. In addition, we have produced experimental results that indicate that StEM performs better than SiNE on both measures of accuracy and runtime.

As we continue work on StEM, we plan to evaluate benchmark against more recent work in addition to SiNE. Moreover, we plan to using larger datasets to also demonstrate that the representations learned by StEM are useful for community detection in signed networks. We also plan to further investigate the effects of the hyperparameters d and λ on StEM's performance. In addition, we plan to extend StEM to handle cases where extra node attribute information is observed and available to inform the representations learned.

ACKNOWLEDGMENTS

This work has been supported by the Trinidad and Tobago Network Information Centre.

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 37–48.

[2] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery* 29, 3 (2015), 626–688.

[3] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*. 585–591.

[4] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. 2011. Node classification in social networks. In *Social network data analytics*. Springer, 115–148.

[5] Ingwer Borg and Patrick JF Groenen. 2005. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media.

[6] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. 2000. Graph structure in the web. *Computer networks* 33, 1-6 (2000), 309–320.

[7] Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. 2017. Neural Embeddings of Graphs in Hyperbolic Space. *arXiv preprint arXiv:1705.10359* (2017).

[8] Ellen Tucker Aaron Clauset and Matthias Sainz. 2016. The Colorado index of complex networks. (2016).

[9] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 135–144.

[10] Patrick Doreian and Andrej Mrvar. 1996. Structural balance and partitioning signed graphs. *Developments in data analysis* (1996), 195–208.

[11] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[12] David Easley and Jon Kleinberg. 2010. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press.

[13] Facebook. 2018. Facebook Home Page. (2018). http://facebook.com.

[14] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.

[15] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 249–256.

[16] Palash Goyal and Emilio Ferrara. 2017. Graph embedding techniques, applications, and performance: A survey. *arXiv preprint arXiv:1705.02801* (2017).

[17] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[18] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[19] Per Hage and Frank Harary. 1983. *Structural models in anthropology*. Cambridge University Press.

[20] Richard W Hamming. 1950. Error detecting and error correcting codes. *Bell Labs Technical Journal* 29, 2 (1950), 147–160.

[21] David Harel and Yehuda Koren. 2002. Graph drawing by high-dimensional embedding. In *International symposium on graph drawing*. Springer, 207–219.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[23] Zan Huang, Xin Li, and Hsinchun Chen. 2005. Link prediction approach to collaborative filtering. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 141–142.

[24] John D Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in science & engineering* 9, 3 (2007), 90–95.

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

[26] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[27] JF Kruiger, PE Rauber, Rafael Messias Martins, Andreas Kerren, Stephen Kobourov, and Alexandru C Telea. 2017. Graph Layouts by t-SNE. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 283–294.

[28] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. 2016. Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 221–230.

[29] David Burth Kurka, Alan Godoy, and Fernando J Von Zuben. 2015. Online social network analysis: A survey of research applications in computer science. *arXiv preprint arXiv:1504.05655* (2015).

[30] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 1361–1370.

[31] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. (June 2014).

[32] Xin Li and Hsinchun Chen. 2013. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems* 54, 2 (2013), 880–890.

[33] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *journal of the Association for Information Science and Technology* 58, 7 (2007), 1019–1031.

[34] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[35] Matthew Malloy, Paul Barford, Enis Ceyhun Alp, Jonathan Koller, and Adria Jewell. 2017. Internet Device Graphs. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 1913–1921. https://doi.org/10.1145/3097983.3098114

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

[37] Maximillian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*. 6341–6350.

[38] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, and Ploutarchos Spyridonos. 2012. Community detection in social media. *Data Mining and Knowledge Discovery* 24, 3 (2012), 515–554.

[39] Srinivasan Parthasarathy, Shirish Tatikonda, and Duygu Ucar. 2010. A survey of graph mining techniques for biological datasets. In *Managing and mining graph data*. Springer, 547–580.

[40] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[42] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[43] Kenneth E Read. 1954. Cultures of the central highlands, New Guinea. *Southwestern Journal of Anthropology* 10, 1 (1954), 1–43.

[44] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 385–394.

[45] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[46] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*. 1139–1147.

[47] Jiliang Tang, Charu Aggarwal, and Huan Liu. 2016. Node classification in signed social networks. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 54–62.

[48] Jiliang Tang, Yi Chang, Charu Aggarwal, and Huan Liu. 2016. A survey of signed network mining in social media. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 42.

[49] Jiliang Tang, Xia Hu, and Huan Liu. 2014. Is distrust the negation of trust?: the value of distrust in social media. In *Proceedings of the 25th ACM conference on Hypertext and social media*. ACM, 148–157.

[50] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering* 13, 2 (2011), 22–30.

[51] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.

[52] Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. 2017. Signed network embedding in social media. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 327–335.

[53] Andrew Scott Waugh, Liuyi Pei, James H Fowler, Peter J Mucha, and Mason Alexander Porter. 2009. Party polarization in congress: A network science approach. (2009).

[54] Robert West, Hristo S Paskov, Jure Leskovec, and Christopher Potts. 2014. Exploiting social network structure for person-to-person sentiment analysis. *arXiv preprint arXiv:1409.2450* (2014).

[55] Matthew D Zeiler. 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).