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:

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 a single set of objects, and 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
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 \( \mathcal{E}_- = \{(u, v, s) \in \mathcal{E} \text{ and } s = -1\} \) be the set of positive edges. We let \( |\cdot| \) be the standard cardinality operator, and as such as we let \( N = |\mathcal{V}| \) 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 \( f : \mathcal{V} \rightarrow \mathbb{R}^d \), 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 \( W \), \( W_{i,:} \) (the \( i^{th} \) row of the matrix \( W \)) is the representation learned for item with index \( i \). 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:

\[
\max_{f, \theta} \prod_{(u, v, s) \in \mathcal{E}} Pr(y = I_{s=1} \mid u, v, \theta, f) \tag{1}
\]

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

\[
\min_{f, \theta} \sum_{(u, v, s) \in \mathcal{E}} -\log Pr(y = I_{s=1} \mid u, v, \theta, f) \tag{2}
\]

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:

\[
Pr(x \mid u, v, \theta, f) = \begin{cases} 
\sigma(\xi(u, v; \theta, f)) & x = 1 \\
1 - \sigma(\xi(u, v; \theta, f)) & x = 0
\end{cases}
\]

(3)

where \( \sigma \) is the logit function. Note that we introduce a function \( \xi : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R} \) that acts as a "measure" of relative similarity of two nodes. We let \( \xi \) be a function of the following form:

\[
\xi(u, v; \theta, f) = \varphi(f(u); \beta)M_2\varphi(f(v); \beta)^T + b_2
\]

\[+ b_2
\]

(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, \( \varphi : \mathbb{R}^d \rightarrow \mathbb{R}^d \) is a parameterizable activation function, and \( \beta \) is a learnable parameter of \( \varphi \). Note that \( b_2, M_2, and \beta \) are \( \theta \) 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, \( \xi(u, v; \theta, f) \) 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, \( \xi(u, v; \theta, f) \) should be negative, thereby leading to \( Pr(x = 1 \mid u, v, \theta, f) < Pr(x = 0 \mid u, v, \theta, f) \). We let \( \xi \) be a single-channeled parameterized leaky rectified linear unit as defined by He et al. [22]. Its definition is shown below:

\[
\varphi(x[i]; \beta) = \max(0, x[i]) + \beta \min(0, x[i])
\]

(5)

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

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 \( f : \mathcal{V} \rightarrow \mathbb{R}^d \) as follows:

\[
\begin{align*}
f(u) &= (M_1g(u)^T + b_1)^T \\
\end{align*}
\]

\[+ b_1
\]

(6)

\[+ b_1
\]

(7)

where \( u \in \mathcal{V}, M_1 \in \mathbb{R}^{d \times 2d}, W \in \mathbb{R}^{N \times d}, and b_1 \in \mathbb{R} \) 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 follows: 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 \( \Theta \).

\[
\begin{align*}
    r_u &= W_{u,:}^T; \\
    x_u &= (M_1 r_u^T + b_1)^T; \\
    q_u &= \text{PReLU}(x_u, \beta); \\
    z &= q_u M_2 q_u^T + b_2; \\
    p &= \sigma(z)
\end{align*}
\]

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:

\[
\begin{align*}
    r_u &= W_{u,:}; \\
    x_u &= (M_1 r_u^T + b_1)^T
\end{align*}
\]

### 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 parameterizing 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 \( \Theta \) 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.

\[
\mathcal{L}(E, p, \Theta) = \frac{1}{|E|} \sum_{(u,v,s) \in E} \left( 1 - I_{\{s=1\}} \right) \log(p(u, v; \Theta)) + \left( I_{\{s=1\}} \right) \log(1-p(u, v; \Theta))
\]

To prevent overfitting, we apply standard L2 regularizations on the parameters contained in \( \Theta \), to obtain:

\[
\mathcal{L}(E, p, \Theta) = \frac{1}{|E|} \sum_{(u,v,s) \in E} \left( 1 - I_{\{s=1\}} \right) \log(p(u, v; \Theta)) + \left( I_{\{s=1\}} \right) \log(1-p(u, v; \Theta)) + \lambda \sum_{\Theta} ||\Theta||_2
\]

where \( 0 < \lambda \ll 1 \) 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 \( \beta \) 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 \( E = \min(|E_+|, |E_-|) \). Hence, a single epoch has complexity \( O(d^3 E) \). If we iterate for \( t \) epochs, then the computational complexity of training our model is \( O(t E d^3) \).

### 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 hyperparameters 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/P₀ 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. 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 Bitcoin-Otc and BitcoinAlpha [28], Epinions [30], Wiki-Rfa [54] and Slashdot [30] datasets were downloaded from the SNAP [31] repository. The US Senate and House roll-call votes [53] dataset were downloaded through the ICON [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 \( u \) and user \( v \) if...
visualization, we can develop insight into the underlying patterns contained in the subgraph of the original graph. In such cases, the triples extracted for SiNE were extracted from the edges formed this matrix into a complete signed network by constructing an undirected complete graph. The sign for an edge between nodes is computed as follows:

\[
E_{ij} = \begin{cases} 
1 & \text{if } x_i[\alpha] \neq 0 \\
0 & \text{otherwise}
\end{cases}
\]

where \(x_i[\alpha]\) is the vote of senator \(y\) on bill \(x\) (either 1, 0, or -1), and \(H\) is the hamming distance 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

| Dataset         | \(N\) | \(|E_+|\)   | \(|E_-|\)   |
|-----------------|-------|-------------|-------------|
| 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 admiship and creating a negative edge if \(u\) opposed \(v\)'s request for admiship. 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:

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

where \(u_i[\alpha]\) is the vote of senator \(y\) on bill \(x\) (either 1, 0, or -1), and \(H\) is the hamming distance 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, SiNE 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 \(E\). 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.

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 and tSNE 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 SiNE and 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. 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.

5 Conclusion

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 SiEM and SiNE/\(P_0\) are seen in Figures 3 and 4 respectively. While both SiEM and SiNE/\(P_0\)'s representations achieve a noticeable separation between Republicans and Democrats, the separation achieved by SiEM’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 SiEM, and the plots on the right are the representations learned by SiNE. Notice that with 60% of the data, SiEM 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 SiEM 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.
with a portion of the class labels. We used 5-fold cross validation on
all of our datasets usable for node classification were complete
graphs, we used SiNE/P₀ instead of SiNE.

As seen in both Tables 2 and 3, both the representations learned
by StEM and SiNE/P₀ tend to perform well on node classification.

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

<table>
<thead>
<tr>
<th>Dataset/Method</th>
<th>StEM</th>
<th>SiNE/P₀</th>
<th>% improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>House-102</td>
<td>0.977 (0.013)</td>
<td>0.868 (0.059)</td>
<td>12.6</td>
</tr>
<tr>
<td>House-103</td>
<td>0.984 (0.014)</td>
<td>0.909 (0.031)</td>
<td>7.5</td>
</tr>
<tr>
<td>House-104</td>
<td>0.982 (0.015)</td>
<td>0.883 (0.054)</td>
<td>11.2</td>
</tr>
<tr>
<td>House-105</td>
<td>0.980 (0.085)</td>
<td>0.849 (0.069)</td>
<td>15.4</td>
</tr>
<tr>
<td>House-106</td>
<td>0.964 (0.013)</td>
<td>0.528 (0.084)</td>
<td>82.6</td>
</tr>
<tr>
<td>House-107</td>
<td>0.977 (0.021)</td>
<td>0.731 (0.083)</td>
<td>24.6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Dataset/Method</th>
<th>StEM</th>
<th>SiNE/P₀</th>
<th>% improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>House-102</td>
<td>0.976 (0.013)</td>
<td>0.842 (0.077)</td>
<td>15.9</td>
</tr>
<tr>
<td>House-103</td>
<td>0.983 (0.014)</td>
<td>0.909 (0.032)</td>
<td>8.1</td>
</tr>
<tr>
<td>House-104</td>
<td>0.981 (0.016)</td>
<td>0.880 (0.054)</td>
<td>11.5</td>
</tr>
<tr>
<td>House-105</td>
<td>0.979 (0.008)</td>
<td>0.845 (0.070)</td>
<td>15.9</td>
</tr>
<tr>
<td>House-106</td>
<td>0.962 (0.014)</td>
<td>0.512 (0.090)</td>
<td>87.9</td>
</tr>
<tr>
<td>House-107</td>
<td>0.977 (0.070)</td>
<td>0.723 (0.086)</td>
<td>35.1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadamard</td>
<td>(y[i] = x_a[i] \times x_b[i])</td>
</tr>
<tr>
<td>L1</td>
<td>(y[i] =</td>
</tr>
<tr>
<td>L2</td>
<td>(y[i] = (x_a[i] - x_b[i])^2)</td>
</tr>
<tr>
<td>Average</td>
<td>(y[i] = 0.5 \times (x_a[i] + x_b[i]))</td>
</tr>
<tr>
<td>Concatenation</td>
<td>(y = x_a \otimes x_b)</td>
</tr>
</tbody>
</table>

### 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 representa-
tions 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 topology
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
digraphs, we used SiNE/P₀ instead of SiNE.

As seen in both Tables 2 and 3, both the representations learned
by StEM and SiNE/P₀ tend to perform well on node classification.

### 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 com-
puted by composing their incident nodes using binary operations.
These edge representations are then used to train a logistic regres-
sion 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, \(E\), into a training set and a testing set. The training set was
used to learn representations using StEM and SiNE. These repre-
sentations were then composed using a binary function from Table
4 to generate representations for the edges in \(E\). Using the rep-
resentations for the edges in the training set, we trained logistic
regression classifiers to predict edge signs from edge representa-
tions. Due to the imbalance in signs in \(E\), 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
Aside from accuracy, runtime is also an important consideration when evaluating representation learning methods. We ran StEM and SiNE, and the L1 operator was the second best for SiNE. We believe that Hadamard composed representations performing worse than the L1 composed representations for SiNE to be a result Equation (4) effects on the representations learned by StEM. We believe 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 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-imagineings 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.

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

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>StEM</th>
<th>SiNE</th>
<th>% improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitcoinOtc</td>
<td>1.57</td>
<td>37.43</td>
<td>95.8</td>
<td></td>
</tr>
<tr>
<td>BitcoinAlpha</td>
<td>1.12</td>
<td>17.41</td>
<td>93.6</td>
<td></td>
</tr>
<tr>
<td>Slashdot</td>
<td>41.47</td>
<td>97.92</td>
<td>57.7</td>
<td></td>
</tr>
<tr>
<td>Epinions</td>
<td>33.27</td>
<td>899.59</td>
<td>96.3</td>
<td></td>
</tr>
<tr>
<td>Wiki-Rfa</td>
<td>11.11</td>
<td>181.08</td>
<td>93.6</td>
<td></td>
</tr>
</tbody>
</table>

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.