

Multi-Scale Attributed Node Embedding

Benedek Rozemberczki¹ Carl Allen¹ Rik Sarkar¹

Abstract

We present network embedding algorithms that capture information about a node from the local distribution over node attributes around it, as observed over random walks following an approach similar to Skip-gram. Observations from neighborhoods of different sizes are either pooled (AE) or encoded distinctly in a multi-scale approach (MUSAE). Capturing attribute-neighborhood relationships over multiple scales is useful for a diverse range of applications, including latent feature identification across disconnected networks with similar attributes. We prove theoretically that matrices of node-feature pointwise mutual information are implicitly factorized by the embeddings. Experiments show that our algorithms are robust, computationally efficient and outperform comparable models on social networks and web graphs.

1. Introduction

Node embedding is a fundamental technique in network analysis that serves as a precursor to numerous downstream machine learning and optimisation tasks, e.g. community detection, network visualization and link prediction (Perozzi et al., 2014; Grover & Leskovec, 2016; Tang et al., 2015). Several recent network embedding methods, such as *Deepwalk* (Perozzi et al., 2014), *Node2Vec* (Grover & Leskovec, 2016) and *Walklets* (Perozzi et al., 2017), achieve impressive performance by learning the network structure following an approach similar to *Word2Vec Skip-gram* (Mikolov et al., 2013b), originally designed for word embedding. In these works, sequences of neighboring nodes are generated from random walks over a network, and representations are distilled from node-node proximity statistics that capture local neighbourhood information.

In real-world networks, nodes often have attributes (or features), e.g. in a social network nodes may represent people

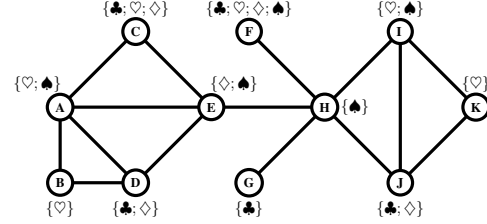


Figure 1. Identifying network roles with attributed embeddings: nodes B and K have the same features as do their nearest neighbors, whereas features at higher order neighborhoods differ.

and node attributes may capture a person’s interests, preferences or habits. Attributes of a node and those of its local neighbourhood may contain information useful in downstream tasks. Such neighborhoods can be considered at different path lengths, or *scales*, e.g. in a social network, near neighbors may be friends, whereas nodes separated by greater scales may have looser friends-of-friends associations. Attributes of neighbors at different scales can be considered separately (*multi-scale*) or *pooled* in some way, e.g. weighted average. Node attributes can identify different network structure, e.g. nodes with similar attributes are often more likely to be connected (known as *homophily*) such that patterns of similar node attributes may identify a community. Alternatively, similar attribute distributions over node neighbourhoods may identify similar network *roles* even at distant locations in a network (Figure 1), or in different networks.

Attributed network embedding methods (e.g. Yang et al., 2015; Huang et al., 2017; Liao et al., 2018) leverage attribute information to supplement local network structure, benefiting many applications, e.g. recommender systems, node classification and link prediction (Yang et al., 2018; Yang & Yang, 2018; Zhang et al., 2018). Methods that consider a node’s attributes generalize those that do not, for which a node’s “feature” can be considered a standard basis vector (i.e. the node-feature matrix is the identity matrix).

Many embedding methods correspond to matrix factorization, indeed some attributed embedding methods (e.g. Yang et al., 2018) explicitly factorize a matrix of link-attribute information. Word embeddings learned using Skip-gram are known to implicitly factorize a matrix of *pointwise mutual information* (PMI) of word co-occurrence statistics (Levy & Goldberg, 2014). Related network embedding methods (Per-

*Equal contribution ¹School of Informatics, The University of Edinburgh, Edinburgh, United Kingdom. Correspondence to: Benedek Rozemberczki <Benedek.Rozemberczki@ed.ac.uk>.

ozzi et al., 2014; Grover & Leskovec, 2016; Tang et al., 2015; Qiu et al., 2018) thus also factorize PMI matrices that relate to the probability of encountering other nodes on a random walk (Qiu et al., 2018).

Our key contributions are:

1. to introduce the first Skip-gram style embedding algorithms that consider attribute distributions over local neighborhoods, both pooled (*AE*) and multi-scale (*MUSAE*), and their counterparts with distinct features attributed to each node (*AE-EGO* and *MUSAE-EGO*);
2. to derive the PMI matrices factorised by all embeddings in terms of adjacency and node-feature matrix and show that popular network embedding methods *DeepWalk* (Perozzi et al., 2014) and *Walklets* (Perozzi et al., 2017) are special cases of *AE* and *MUSAE*; and
3. to show empirically that on real-world networks (e.g. Wikipedia, Facebook) our unsupervised attributed algorithms outperform comparable methods at predicting node attributes, achieve competitive link prediction performance, are computationally scalable and are the first to enable *transfer learning* between networks.

Reference code and introduced evaluation datasets are provided (<https://github.com/benedekrozemberczki/MUSAE>).

2. Related Work

Efficient unsupervised learning of node embeddings for large networks has recently seen unprecedented development. The current paradigm focuses on learning latent node representations such that those sharing neighbors (Perozzi et al., 2014; Tang et al., 2015; Grover & Leskovec, 2016; Perozzi et al., 2017), structural roles (Ribeiro et al., 2017; Ahmed et al., 2018) or attributes are close in the latent space. Our work falls under the first and last of these categories.

Several recent *neighborhood-preserving* node embedding algorithms are inspired by the *Skip-gram* model (Mikolov et al., 2013a;b), which generates word embeddings by implicitly factorizing a matrix of shifted pointwise mutual information (PMI) of word co-occurrence statistics extracted from a text corpus (Levy & Goldberg, 2014). For example, *DeepWalk* (Perozzi et al., 2014) generates a “corpus” of truncated random walks over a graph from which the *Skip-gram* model generates neighborhood preserving node embeddings. In doing so, *DeepWalk* implicitly factorizes a PMI matrix that has been shown to correspond to the mean of a set of normalized adjacency matrix powers (up to a given order) reflecting different path lengths of a first-order Markov process (Qiu et al., 2018). Such averaging, or *pooling* treats neighbors at different path lengths (or *scales*) equally or according to fixed weightings (Mikolov et al., 2013a; Grover & Leskovec, 2016); whereas it has been

found that an optimal weighting may be task or dataset specific (Abu-El-Haija et al., 2018). In contrast, *multi-scale* node embedding methods, such as *LINE* (Tang et al., 2015), *GraRep* (Cao et al., 2015) and *Walklets* (Perozzi et al., 2017), learn separate lower-dimensional embedding *components* for each path length and concatenate them to form the full node representation. Such un-pooled representations, comprising distinct but less information at each scale, are found to give higher performance in a number of downstream settings, without increasing the overall complexity or number of free parameters (Perozzi et al., 2017).

Attributed node embedding methods refine ideas from neighborhood preserving node embeddings to also incorporate node *attributes* (equivalently, features or labels) (Yang et al., 2015; Liao et al., 2018; Huang et al., 2017; Yang et al., 2018; Yang & Yang, 2018). Similarities between both a node’s neighborhood structure and features contribute to determining pairwise proximity in the latent space, although models follow quite different strategies to learn such representations. The (arguably) simplest model, *TADW* (Yang et al., 2015), decomposes a convex combination of normalized adjacency matrix powers into a matrix product that includes the feature matrix. Several other models, such as *SINE* (Zhang et al., 2018) and *ASNE* (Liao et al., 2018), implicitly factorize a matrix formed by concatenating the feature and adjacency matrices. Other approaches such as *TENE* (Yang & Yang, 2018), formulate the attributed node embedding task as a joint non-negative matrix factorization problem in which node representations obtained from sub-tasks are used to regularize one another. *AANE* (Huang et al., 2017) uses a similar network structure based regularization approach, in which a node feature similarity matrix is decomposed using the alternating direction method of multipliers. *BANE* (Yang et al., 2018), the method most similar to our own, learns attributed node embeddings that explicitly factorize the product of a normalized adjacency matrix power and a feature matrix. Many other methods exist, but do not consider the attributes of higher order neighborhoods (Yang et al., 2015; Liao et al., 2018; Huang et al., 2017; Zhang et al., 2018; Yang & Yang, 2018). A key difference between our work and previous methods is that we jointly learn *distinct* representations of nodes and features.

The relationship between our pooled (*AE*) and multi-scale (*MUSAE*) attributed node embedding methods mirrors that between graph convolutional neural networks (GCNNs) and multi-scale GCNNs. The former, e.g. *GCN* (Kipf & Welling, 2017), *GraphSage* (Hamilton et al., 2017), *GAT* (Veličković et al., 2018), *APPNP* (Klicpera et al., 2019), *SGCONV* (Wu et al., 2019) and *ClusterGCN* (Chiang et al., 2019), create latent node representations that pool node attributes from arbitrary order neighborhoods, which are then inseparable and unrecoverable. The latter, e.g. *MixHop* (Abu-El-Haija et al., 2019) learn latent features for each proximity.

3. Attributed Embedding Algorithms

Here we define our algorithms to jointly learn embeddings of nodes and attributes based on the structure and attributes of local neighborhoods. The aim is to learn similar embeddings for nodes that occur in neighborhoods of similar attributes; and similar embeddings for attributes that occur in similar neighborhoods of nodes. Let $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ be an undirected graph, where \mathbb{V} and \mathbb{E} are the sets of vertices and edges (or links), and let \mathbb{F} be the set of all binary node features. For each node $v \in \mathbb{V}$, let $\mathbb{F}_v \subseteq \mathbb{F}$ be the subset of features belonging to v . An embedding of nodes is a mapping $g : \mathbb{V} \rightarrow \mathbb{R}^d$ that assigns a d -dimensional vector $g(v)$, or simply g_v , to each node v and is fully described by a matrix $\mathbf{G} \in \mathbb{R}^{|\mathbb{V}| \times d}$. An embedding of the features (to the same latent space) is a mapping $h : \mathbb{F} \rightarrow \mathbb{R}^d$ with embeddings denoted $h(f) \doteq h_f$, as summarised by a matrix $\mathbf{H} \in \mathbb{R}^{|\mathbb{F}| \times d}$.

3.1. Attributed Embedding

The *Attributed Embedding* (AE) method is described by Algorithm 1. With probability proportional to node degree, s starting nodes v_1 are sampled from \mathbb{V} (line 2). From each starting node, a node sequence of length l is sampled over \mathcal{G} following a first-order random walk (line 3). For a given window size t , iterate over the first $l-t$ nodes of the sequence v_j , termed *source* nodes (line 4). For each source node, the subsequent t nodes are considered *target* nodes (line 5). For each target node v_{j+r} , the tuple (v_j, f) is added to the corpus \mathbb{D} for each feature $f \in \mathbb{F}_{v_{j+r}}$ (lines 6-7). Each tuple (v_{j+r}, f) for features of the source node $f \in \mathbb{F}_{v_j}$ is also added to \mathbb{D} (lines 9-10). Running Skip-gram on \mathbb{D} with b negative samples (line 15) generates the d -dimensional node and feature embeddings (\mathbf{G}, \mathbf{H}) .

3.2. Multi-scale Attributed Embedding

The AE algorithm pools features across neighborhoods of different proximity. Inspired by the performance of (unattributed) multi-scale node embeddings, we adapt AE to learn *multi-scale* attributed embeddings. The embedding component of a node $v \in \mathbb{V}$ at proximity $r \in \{1, \dots, t\}$ is given by a mapping $g^r : \mathbb{V} \rightarrow \mathbb{R}^{d/t}$ (where t divides d). Similarly, the embedding component of feature $f \in \mathbb{F}$ at proximity r is given by a mapping $h^r : \mathbb{F} \rightarrow \mathbb{R}^{d/t}$. Concatenating the t components gives a d -dimensional embedding for each node and feature. The *Multi-Scale Attributed Embedding* (MUSAE) method is described by Algorithm 2. Source and target node pairs are generated from sampled node sequences as for AE (lines 2-5). Each feature of a target node $f \in \mathbb{F}_{v_{j+r}}$ is again considered, but tuples (v_j, f) are added to a *sub-corpus* \mathbb{D}_{\rightarrow} (lines 6-7) and for each source node feature $f \in \mathbb{F}_{v_j}$ tuples (v_{j+r}, f) are added to another sub-corpus \mathbb{D}_{\leftarrow} (lines 9-10). Running Skip-gram with b negative samples on each sub-corpus $\mathbb{D}_r = \mathbb{D}_{\rightarrow} \cup \mathbb{D}_{\leftarrow}$ (line 17) generates the $\frac{d}{t}$ -dimensional components to concatenate.

Algorithm 1 AE sampling and training procedure

Data: $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ – Graph to be embedded.
 $\{\mathbb{F}_v\}_v$ – Set of node feature sets.
 s – Number of sequence samples.
 l – Length of sequences.
 t – Context size.
 d – Embedding dimension.
 b – Number of negative samples.

Result: Node embedding g and feature embedding h .

```

1 for  $i$  in  $1 : s$  do
2   Pick  $v_1 \in \mathbb{V}$  according to  $P(v) \sim \text{deg}(v)/\text{vol}(\mathcal{G})$ .
3    $(v_1, v_2, \dots, v_l) \leftarrow \text{Sample Nodes}(\mathcal{G}, v_1, l)$ 
4   for  $j$  in  $1 : l - t$  do
5     for  $r$  in  $1 : t$  do
6       for  $f$  in  $\mathbb{F}_{v_{j+r}}$  do
7         Add tuple  $(v_j, f)$  to multiset  $\mathbb{D}$ .
8       end
9       for  $f$  in  $\mathbb{F}_{v_j}$  do
10        Add tuple  $(v_{j+r}, f)$  to multiset  $\mathbb{D}$ .
11      end
12    end
13  end
14 end
15 Run SGNS on  $\mathbb{D}$  with  $b$  negative samples and  $d$  dimensions.
16 Output  $g_v, \forall v \in \mathbb{V}$ , and  $h_f, \forall f \in \mathcal{F} = \cup_v \mathbb{F}_v$ .

```

Algorithm 2 MUSAE sampling and training procedure

Data: $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ – Graph to be embedded.
 $\{\mathbb{F}_v\}_v$ – Set of node feature sets.
 s – Number of sequence samples.
 l – Length of sequences.
 t – Context size.
 d – Embedding dimension.
 b – Number of negative samples.

Result: Node embedding components g^r and feature embeddings component h^r , for $r \in \{1, \dots, t\}$.

```

1 for  $i$  in  $1 : s$  do
2   Pick  $v_1 \in \mathbb{V}$  according to  $P(v) \sim \text{deg}(v)/\text{vol}(\mathcal{G})$ .
3    $(v_1, v_2, \dots, v_l) \leftarrow \text{Sample Nodes}(\mathcal{G}, v_1, l)$ 
4   for  $j$  in  $1 : l - t$  do
5     for  $r$  in  $1 : t$  do
6       for  $f$  in  $\mathbb{F}_{v_{j+r}}$  do
7         Add the tuple  $(v_j, f)$  to multiset  $\mathbb{D}_{\rightarrow}$ .
8       end
9       for  $f$  in  $\mathbb{F}_{v_j}$  do
10        Add the tuple  $(v_{j+r}, f)$  to multiset  $\mathbb{D}_{\leftarrow}$ .
11      end
12    end
13  end
14 end
15 for  $r$  in  $1 : t$  do
16   Create  $\mathbb{D}_r$  by unification of  $\mathbb{D}_{\rightarrow}$  and  $\mathbb{D}_{\leftarrow}$ .
17   Run SGNS on  $\mathbb{D}_r$  with  $b$  negative samples and  $\frac{d}{t}$  dimensions.
18   Output  $g_v^r, \forall v \in \mathbb{V}$ , and  $h_f^r, \forall f \in \mathbb{F} = \cup_v \mathbb{F}_v$ .
19 end

```

4. Attributed embedding as implicit matrix factorization

Levy & Goldberg (2014) showed that the loss function of Skip-gram with negative sampling (SGNS) is minimized if its two output embedding matrices \mathbf{W}, \mathbf{C} factorize a matrix of pointwise mutual information (PMI) of word co-occurrence statistics. Specifically, for a corpus \mathbb{D} over a dictionary \mathbb{V} with $|\mathbb{V}| = n$, SGNS (with b negative samples) generates embeddings $\mathbf{w}_w, \mathbf{c}_c \in \mathbb{R}^d$ (columns of $\mathbf{W}, \mathbf{C} \in \mathbb{R}^{d \times n}$) for each target and context word $w, c \in \mathbb{V}$, satisfying:

$$\mathbf{w}_w^\top \mathbf{c}_c \approx \log \left(\frac{\#(w, c | \mathbb{D})}{\#(w) \#(c)} \right) - \log b,$$

where $\#(w)$, $\#(c)$ and $\#(w, c)$ denote counts of w , c and both words appearing within a sliding context window. Considering $\frac{\#(w)}{|\mathbb{D}|}$, $\frac{\#(c)}{|\mathbb{D}|}$, $\frac{\#(w, c)}{|\mathbb{D}|}$ as empirical estimates of $p(w)$, $p(c)$ and $p(w, c)$ respectively gives the approximate low-rank factorization (since typically $d \ll n$):

$$\mathbf{W}^\top \mathbf{C} \approx [\text{PMI}(w, c) - \log b]_{w, c \in \mathbb{V}},$$

Qiu et al. (2018) extended this result to node embedding models that apply SGNS to a “corpus” generated from random walks over the graph. In the case of *DeepWalk* where random walks are first-order Markov, the joint probability distributions over nodes at different steps of a random walk can be expressed in closed form, and a closed form for the factorized PMI matrix follows. Here we derive the matrices implicitly factorized by *AE* and *MUSAE*.

Notation: For a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, $|\mathbb{V}| = n$, let $\mathbf{A} \in \mathbb{R}^{n \times n}$ denote the adjacency matrix and $\mathbf{D} \in \mathbb{R}^{n \times n}$ the diagonal degree matrix, i.e. $D_{v,v} = \deg(v) = \sum_w \mathbf{A}_{v,w}$. Let $c = \sum_{v,w} \mathbf{A}_{v,w}$ denote the volume of \mathcal{G} . We define the binary attribute matrix $\mathbf{F} \in \{0, 1\}^{|\mathbb{V}| \times |\mathbb{F}|}$ by $F_{v,f} = \mathbb{1}_{f \in \mathbb{F}_v}$, $\forall v \in \mathbb{V}, f \in \mathbb{F}$. To ease notation, we set $\mathbf{P} = \mathbf{D}^{-1} \mathbf{A}$ and $\mathbf{E} = \text{diag}(\mathbf{1}^\top \mathbf{D} \mathbf{F})$, where diag indicates a diagonal matrix.

Interpretation: Assuming \mathcal{G} is ergodic, $p(v) = \frac{\deg(v)}{c}$ is the stationary distribution over nodes $v \in \mathbb{V}$, i.e. $c^{-1} \mathbf{D} = \text{diag}(p(v))$; and $c^{-1} \mathbf{A}$ is the stationary joint distribution over consecutive nodes of a random walk $p(v_j, v_{j+1})$. $\mathbf{F}_{v,f}$ can be considered a Bernoulli parameter of the probability $p(f|v)$ of observing feature f at a node v , hence $c^{-1} \mathbf{D} \mathbf{F}$ describes the stationary joint distribution $p(f, v)$ over nodes and features. Accordingly, \mathbf{P} is the transition matrix of conditional probabilities $p(v_{j+1}|v_j)$; and \mathbf{E} is a diagonal matrix proportional to the (binary) probability $p(f)$ of observing feature f at the stationary distribution. We note that $p(f)$ need not sum to 1, whereas $p(v)$ necessarily must.

4.1. Multi-scale case (MUSAE)

We know that the SGNS aspect of *MUSAE* learns embeddings g_v^r, h_f^r that satisfy $g_v^r \top h_f^r \approx \log \left(\frac{\#(v, f)_r | \mathbb{D}_r|}{\#(v)_r \#(f)_r} \right) - \log b$,

$\forall v \in \mathbb{V}, f \in \mathbb{F}$. Our aim is to express this factorization in terms of known properties of the graph \mathcal{G} and its features.

Lemma 1. *The empirical statistics of node-feature pairs obtained from random walks give unbiased estimates of the joint probability of observing feature $f \in \mathbb{F}$ r steps (i) **after**; or (ii) **before** node $v \in \mathbb{V}$, as given by:*

$$\begin{aligned} \text{plim}_{l \rightarrow \infty} \frac{\#(v, f)_{\vec{r}}}{|\mathbb{D}_{\vec{r}}|} &= c^{-1} (\mathbf{D} \mathbf{P}^r \mathbf{F})_{v, f} \\ \text{plim}_{l \rightarrow \infty} \frac{\#(v, f)_{\overleftarrow{r}}}{|\mathbb{D}_{\overleftarrow{r}}|} &= c^{-1} (\mathbf{F}^\top \mathbf{D} \mathbf{P}^r)_{f, v} \end{aligned}$$

Proof. See Appendix A. \square

Lemma 2. *The empirical statistics of node-feature pairs obtained from random walks give unbiased estimates of the joint probability of observing feature $f \in \mathbb{F}$ r steps **either side** of node $v \in \mathbb{V}$, given by:*

$$\text{plim}_{l \rightarrow \infty} \frac{\#(v, f)_r}{|\mathbb{D}_r|} = c^{-1} (\mathbf{D} \mathbf{P}^r \mathbf{F})_{v, f},$$

Proof. See Appendix A. \square

Marginalizing gives unbiased estimates of stationary probability distributions of nodes and features:

$$\begin{aligned} \text{plim}_{l \rightarrow \infty} \frac{\#(v)}{|\mathbb{D}_r|} &= \frac{\deg(v)}{c} = c^{-1} \mathbf{D}_{v, v} \\ \text{plim}_{l \rightarrow \infty} \frac{\#(f)}{|\mathbb{D}_r|} &= \sum_{v|f \in \mathbb{F}_v} \frac{\deg(v)}{c} = c^{-1} \mathbf{E}_{f, f} \end{aligned}$$

Theorem 1. *MUSAE embeddings approximately factorize the node-feature PMI matrix:*

$$\log (c \mathbf{P}^r \mathbf{F} \mathbf{E}^{-1}) - \log b, \quad \text{for } r = 1, \dots, t.$$

Proof.

$$\begin{aligned} \frac{\#(v, f)_r | \mathbb{D}_r|}{\#(f)_r \#(v)_r} &= \left(\frac{\#(v, f)_r}{|\mathbb{D}_r|} \right) / \left(\frac{\#(f)_r}{|\mathbb{D}_r|} \frac{\#(v)_r}{|\mathbb{D}_r|} \right) \\ &\xrightarrow{p} ((c \mathbf{D}^{-1})(c^{-1} \mathbf{D} \mathbf{P}^r \mathbf{F})(c \mathbf{E}^{-1}))_{v, f} \\ &= c (\mathbf{P}^r \mathbf{F} \mathbf{E}^{-1})_{v, f} \end{aligned} \quad \square$$

4.2. Pooled case (AE)

Lemma 3. *The empirical statistics of node-feature pairs learned by the AE algorithm give unbiased estimates of mean joint probabilities over different path lengths:*

$$\text{plim}_{l \rightarrow \infty} \frac{\#(v, f)}{|\mathbb{D}|} = \frac{c}{t} \left(\mathbf{D} \left(\sum_{r=1}^t \mathbf{P}^r \right) \mathbf{F} \right)_{v, f}$$

Proof. By construction, $|\mathbb{D}| = \sum_r |\mathbb{D}_r|$, $\#(v, f) = \sum_r \#(v, f)_r$, $|\mathbb{D}_r| = |\mathbb{D}_s| \forall r, s \in \{1, \dots, t\}$ and so $|\mathbb{D}_s| = t^{-1}|\mathbb{D}|$. Combining with Lemma 2, the result follows. \square

Theorem 2. *AE embeddings approximately factorize the pooled node-feature matrix:*

$$\log \left(\frac{c}{t} \left(\sum_{r=1}^t \mathbf{P}^r \right) \mathbf{F} \mathbf{E}^{-1} \right) - \log b.$$

Proof. Analogous to the proof of Theorem 1. \square

Remark 1. *DeepWalk is a special case of AE with $\mathbf{F} = \mathbf{I}_{|\mathbb{V}|}$.*

That is, *DeepWalk* is equivalent to *AE* if each node has a single unique feature. Thus $\mathbf{E} = \text{diag}(\mathbf{1}^\top \mathbf{D} \mathbf{I}) = \mathbf{D}$ and, by Theorem 2, the embeddings of *DeepWalk* factorize $\log \left(\frac{c}{t} \left(\sum_{r=1}^t \mathbf{P}^r \right) \mathbf{D}^{-1} \right) - \log b$, as derived by Qiu et al. (2018).

Remark 2. *Walklets is a special case of MUSAE with $\mathbf{F} = \mathbf{I}_{|\mathbb{V}|}$.*

Thus, for $r = 1, \dots, t$, the embeddings of *Walklets* factorise $\log \left(c \mathbf{P}^r \mathbf{D}^{-1} \right) - \log b$.

Remark 3. *Appending an identity matrix \mathbf{I} to the feature matrices \mathbf{F} of AE and MUSAE (denoted $[\mathbf{F}; \mathbf{I}]$) adds a unique feature to each node. The resulting algorithms, named AE-EGO and MUSAE-EGO, respectively, learn embeddings that approximately factorize the node-feature PMI matrices:*

$$\log \left(c \mathbf{P}^r [\mathbf{F}; \mathbf{I}] \mathbf{E}^{-1} \right) - \log b, \quad \forall r \in \{1, \dots, t\};$$

$$\text{and} \quad \log \left(\frac{c}{t} \left(\sum_{r=1}^t \mathbf{P}^r \right) [\mathbf{F}; \mathbf{I}] \mathbf{E}^{-1} \right) - \log b.$$

4.3. Complexity analysis

Assuming a constant number of features per source node, the corpus generation has runtime complexity of $\mathcal{O}(s l t \frac{m}{n})$, where $m = \sum_{v \in \mathbb{V}} |\mathbb{F}_v|$ the total number of features across all nodes (with repetition), $q = |\mathbb{F}|$, and $n = |\mathbb{V}|$. With b negative samples, the optimization runtime of a single asynchronous gradient descent epoch on *AE* and the joint optimization runtime of *MUSAE* embeddings is $\mathcal{O}(b d s l t \frac{m}{n})$. With p truncated walks from each source node, the corpus generation complexity is $\mathcal{O}(p n l t m)$ and the model optimization runtime is $\mathcal{O}(b d p n l t m)$. The runtime experiments (Section 5) empirically support this analysis.

Corpus generation has a memory complexity of $\mathcal{O}(s l t \frac{m}{n})$ while the same when generating p truncated walks per node has a memory complexity of $\mathcal{O}(p n l t m)$. Storing the parameters of an *AE* embedding has a memory complexity of $\mathcal{O}((n + q) \cdot d)$ and *MUSAE* embeddings also use $\mathcal{O}((n + q) \cdot d)$ memory.

5. Experimental Evaluation

We evaluate the representations learned by *AE*, *MUSAE* and their *EGO* extensions on several common downstream tasks: node classification, regression, link prediction and also transfer learning across networks. We also report how number of nodes and dimensionality affect runtime. We use standard citation graph benchmark datasets (Cora, Citeseer (Lu & Getoor, 2003) and Pubmed (Namata et al., 2012)) together with datasets of social networks and web graphs collected from Facebook, Github, Wikipedia and Twitch.

5.1. Datasets

Table 1 shows statistics of the evaluation datasets.

Table 1. Statistics of evaluation attributed networks.

Dataset	Nodes	Clustering Coefficient	Density	Unique Features
Cora	2,708	0.094	0.002	1,432
Citeseer	3,327	0.130	0.001	3,703
Pubmed	19,717	0.054	0.001	500
Facebook	22,470	0.232	0.001	4,714
GitHub	37,700	0.013	0.001	4,005
Wiki-Chameleons	2,277	0.314	0.012	3,132
Wiki-Crocodiles	11,631	0.026	0.003	13,183
Wiki-Squirrels	5,201	0.348	0.015	3,148
Twitch-DE	9,498	0.047	0.003	2,545
Twitch-EN	7,126	0.042	0.002	2,545
Twitch-ES	4,648	0.084	0.006	2,545
Twitch-FR	6,549	0.054	0.005	2,545
Twitch-PT	1,912	0.131	0.017	2,545
Twitch-RU	4,385	0.049	0.004	2,545

Facebook: a page-page graph of verified Facebook sites. Nodes correspond to official Facebook pages, links to mutual likes between sites. Node features are extracted from the site descriptions. The task is multi-class classification of the site category of a node.

GitHub: a social network where nodes correspond to developers who have starred at least 10 repositories and edges to mutual follower relationships. Node features are location, starred repositories, employer and e-mail address. The task is to classify nodes as web or machine learning developers.

Wiki-[animals]: Wikipedia page-page networks on three topics: chameleons, crocodiles and squirrels. Nodes represent articles from the English Wikipedia (Dec 2018), edges reflect mutual links between them. Node features indicate the presence of particular nouns in the articles and the average monthly traffic (Oct 2017 - Nov 2018). The regression task is to predict the log average monthly traffic (Dec 2018).

Twitch-[country]: user-user networks where nodes correspond to Twitch users and links to mutual friendships. Node features are games liked, location and streaming habits. All datasets have the same set of node features enabling *transfer learning* across networks. The associated task is binary classification of whether a streamer uses explicit language.

5.2. Hyperparameter Settings

Table 2 (top) shows the hyperparameters used for our algorithms, which are consistent other random walk based approaches (Perozzi et al., 2014; Grover & Leskovec, 2016; Ribeiro et al., 2017; Perozzi et al., 2017). Hyperparameters of other algorithms are in Appendices B and C. Downstream evaluation tasks use logistic and elastic net regression from *Scikit-learn* (Pedregosa et al., 2011) with standard library settings except for parameters reported in Table 2 (bottom).

Table 2. Hyperparameter settings for learning and evaluation.

Parameter	Notation	Value
Dimensions	d	128
Walk length	l	80
Walks per node	p	10
Epochs	e	5
Window size	t	3
Negative samples	b	5
Initial learning rate	α_{\max}	0.05
Final learning rate	α_{\min}	0.025
Regularization coefficient	λ	0.01
Norm mixing parameter	γ	0.5

5.3. Node Attribute Classification

Embedding algorithms take as input a network graph and (training) node attributes. Classification performance is evaluated by training l_2 -regularized logistic/softmax regression to predict a (test) attribute given a node embedding. Table 3 compares our models to leading node embedding methods by micro averaged F_1 score over 100 seeded splits (80% train - 20% test). Whilst relative model performance varies slightly with dataset, the results show that our attributed embeddings tend to outperform other unsupervised methods, with closest performance achieved by *Node2Vec* and *Walklets*. We also observe that (i) multi-scale embeddings tend to outperform their pooled counterparts; (ii) the additional identity features of *EGO* models have no material impact on the task; and (iii) attributed node embeddings that consider only first-order neighbours (e.g. ASNE, TENE) show weak performance. As an upper bound comparison, we report performance of *supervised* methods, which shows a typical performance advantage of $\sim 4\%$ that can be within 1%.

We also test the *few-shot learning* ability of attributed embeddings by repeating the above experiment, but training the logistic regression model with only k randomly selected samples per class. Results for a representative selection of datasets and $k \in \{3, \dots, 30\}$ are shown in Figure 2. Our attributed embeddings show a material performance improvement at few shot learning over other unsupervised methods, particularly for the larger data sets (Facebook and Github). We also observe a modest performance benefit for the *EGO* models, suggesting that the additional network structure they encode is useful when data is limited.

Table 3. Node classification performance: micro F_1 score and standard error over 100 seeded runs (best *unsupervised* result in bold, best *supervised* result in blue, our methods in italics).

	Facebook	GitHub	TwitCh PT	Cora	Citeseer	PubMed
DeepWalk	.863 .001	.858 .001	.672 .007	.833 .004	.603 .007	.802 .001
LINE ₂	.875 .001	.858 .001	.670 .001	.777 .004	.542 .006	.799 .001
Node2Vec	.890 .001	.859 .001	.686 .004	.840 .003	.622 .005	.810 .002
Walklets	.887 .001	.860 .001	.671 .006	.843 .003	.630 .006	.815 .001
TADW	.765 .002	.748 .001	.659 .005	.819 .004	.734 .004	.862 .002
AANE	.796 .001	.856 .001	.661 .006	.793 .006	.733 .004	.867 .001
ASNE	.797 .001	.839 .001	.685 .006	.830 .003	.718 .004	.846 .002
BANE	.868 .001	.762 .001	.664 .006	.807 .005	.713 .003	.823 .002
TENE	.731 .002	.850 .001	.664 .006	.829 .005	.681 .003	.842 .001
<i>AE</i>	.888 .001	.863 .001	.672 .004	.835 .005	.739 .005	.839 .002
<i>AE-EGO</i>	.899 .001	.863 .001	.671 .007	.835 .006	.739 .005	.840 .003
<i>MUSAE</i>	.887 .001	.864 .001	.672 .006	.848 .004	.742 .004	.853 .001
<i>MUSAE-EGO</i>	.894 .001	.864 .001	.671 .002	.849 .004	.741 .004	.851 .002
GCN	.932 .001	.865 .001	.695 .007	.879 .001	.742 .001	.875 .001
GraphSAGE	.814 .002	.854 .001	.631 .004	.881 .001	.747 .001	.864 .001
GAT	.919 .001	.864 .001	.678 .001	.867 .002	.740 .001	.869 .001
MixHop	.941 .002	.850 .001	.630 .004	.859 .001	.780 .001	.891 .001
ClusterGCN	.937 .001	.859 .001	.654 .004	.845 .001	.737 .001	.844 .001
APNP	.938 .001	.868 .001	.755 .001	.888 .001	.754 .001	.884 .001
SGCONV	.836 .002	.829 .001	.663 .003	.878 .002	.763 .001	.807 .001

5.4. Node Attribute Regression

We compare the ability of our attributed embeddings against those of other unsupervised methods at predicting real valued node attributes. For each model, we train embeddings on an unsupervised basis and learn regression parameters of an elastic net to predict the log average web traffic (attribute) of each page (node) of the Wikipedia datasets (created for this task). Table 4 reports average test R^2 (explained variance) and standard error over 100 seeded (80% train - 20% test) splits. This shows that: (i) our attributed embeddings tend to outperform all other methods at the regression task; (ii) multi-scale methods (e.g. *MUSAE*) tend to outperform pooled methods (e.g. *AE*); (iii) the additional network information of the *EGO* models appears beneficial, but only in the multi-scale case. Overall, *MUSAE-EGO* outperforms the best baseline for each dataset by 2-10%.

5.5. Link prediction

A common downstream task for node embeddings is to predict edges in a graph, or *link prediction*. Intuitively, whether attribute-level information helps to predict edges seems dataset/attribute dependent and we do not necessarily expect attributed embeddings to outperform those learned by algorithms focused specifically on network structure (e.g.

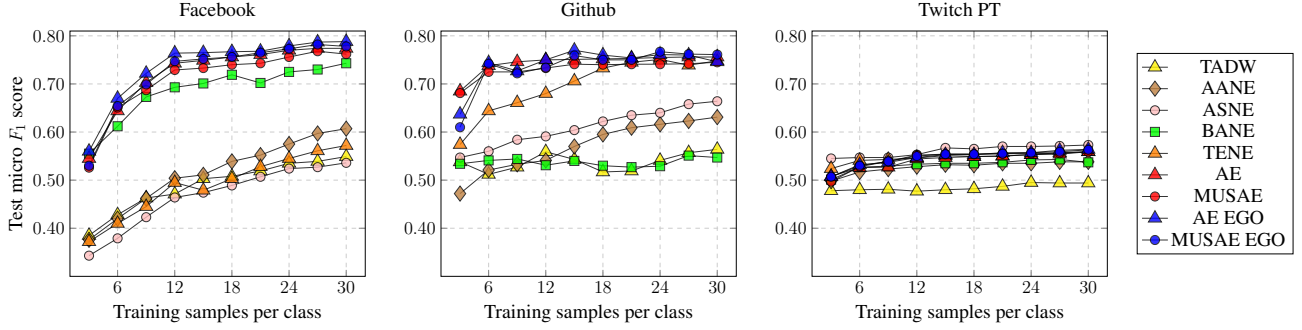


Figure 2. k -shot node classification performance for varying k , evaluated by micro averaged F_1 scores over 100 seeded train-test splits.

Table 4. Node attribute regression: average R^2 statistics and standard error for predicting website traffic (best results in bold).

	Wiki Chameleons	Wiki Crocodiles	Wiki Squirrels
DeepWalk	.375 \pm .004	.553 \pm .002	.170 \pm .001
LINE ₂	.381 \pm .003	.586 \pm .001	.232 \pm .002
Node2Vec	.414 \pm .003	.574 \pm .001	.174 \pm .002
Walklets	.426 \pm .003	.625 \pm .001	.249 \pm .002
TADW	.527 \pm .003	.636 \pm .001	.271 \pm .002
AANE	.598 \pm .007	.732 \pm .002	.287 \pm .002
ASNE	.440 \pm .009	.572 \pm .003	.229 \pm .005
BANE	.464 \pm .003	.617 \pm .001	.168 \pm .002
TENE	.494 \pm .020	.701 \pm .003	.321 \pm .007
AE	.642 \pm .006	.743 \pm .003	.291 \pm .006
AE-EGO	.644 \pm .009	.732 \pm .002	.283 \pm .006
MUSAE	.658 \pm .004	.736 \pm .003	.338 \pm .007
MUSAE-EGO	.653 \pm .011	.747 \pm .003	.354 \pm .009

DeepWalk, *Node2Vec*). We evaluate link prediction performance similarly to Grover & Leskovec (2016), by: (i) randomly removing 50% of edges (maintaining graph connectivity); (ii) learning node embeddings on the attenuated graph; (iii) applying each of four binary operators¹ to generate d -dimensional “edge representations” from pairs of node embeddings for both removed edges (positive samples) and randomly selected nodes (negative samples); and (iv) with an 80% train - 20% test split, training a logistic regression classifier to distinguish positive and negative samples.

Table 5 shows the average AUC for each model over 100 seeded splits with its best performing operator (in all cases Hadamard (H) or l_1) over a range of datasets. Taking the best operator for each model avoids assuming that all embeddings conform to a common metric space. The results show: (i) our attributed embeddings perform comparably to other attributed models, with *MUSAE-EGO* similar to the best performing neighbourhood-preserving algorithm (*Walklets*) and attributed embeddings (*TADW*); (ii) our multi-scale embeddings outperform pooled embeddings; and (iii) the encoded network features of *EGO* embeddings are beneficial.

¹ As in Grover & Leskovec (2016): average, product (Hadamard), absolute difference (l_1), squared difference (l_2).

Table 5. Link prediction: average AUC and standard error for 100 seeded runs and best performing (op)erator (best results in bold).

	Op.	Facebook	GitHub	Twitch DE	Wiki Crocodiles
DeepWalk	H	.981 .001	.799 .001	.750 .003	.966 .001
Node2Vec	H	.982 .001	.822 .001	.780 .003	.973 .001
LINE ₂	l_1	.924 .001	.913 .002	.855 .001	.930 .001
Walklets	l_1	.980 .001	.932 .001	.870 .001	.976 .001
TADW	H	.973 .001	.915 .001	.884 .001	.967 .001
AANE	H	.911 .002	.772 .002	.811 .002	.892 .005
ASNE	H	.973 .001	.912 .001	.866 .002	.940 .001
BANE	H	.653 .002	.664 .003	.816 .002	.738 .002
TENE	l_1	.940 .001	.942 .001	.837 .001	.927 .001
AE	l_1	.968 .001	.889 .001	.870 .002	.952 .002
AE-EGO	l_1	.973 .001	.891 .001	.872 .002	.955 .001
MUSAE	l_1	.973 .001	.908 .001	.879 .001	.967 .001
MUSAE-EGO	l_1	.977 .001	.911 .001	.884 .002	.963 .001

Overall, no algorithm dominates the link prediction task. Importantly, we see that the additional node level information our attributed embeddings learn does not detract from their link prediction performance, which matches that of algorithms that exclusively learn network structure.

5.6. Transfer Learning

Neighbourhood based methods such as *DeepWalk* (Perozzi et al., 2014) learn node embeddings, typically with no mechanism to relate the embeddings of distinct graphs. Thus a regression model learned to predict attributes from node embeddings for one graph (as in any of the previous tasks) will not generally perform well given the node embeddings of another graph. The attributed models *MUSAE* and *AE*, however, learn representations of both nodes and features, in principle enabling information to be shared between graphs with common features. That is, if node and feature embeddings and a regression model are learned for a source graph \mathcal{S} with feature set \mathbb{F} , then node embeddings can be learned for a target graph \mathcal{T} (with the same features \mathbb{F}) that *fit* with the feature embeddings learned for \mathcal{S} and so also its regression model. This amounts to *transfer learning*, or zero-shot

attribute prediction on the target graph. The information shared between graphs allows nodes of \mathcal{T} to be embedded into the same latent space as those of \mathcal{S} . Since other unsupervised embedding methods do not learn explicit feature embeddings, they do not enable such transfer learning.

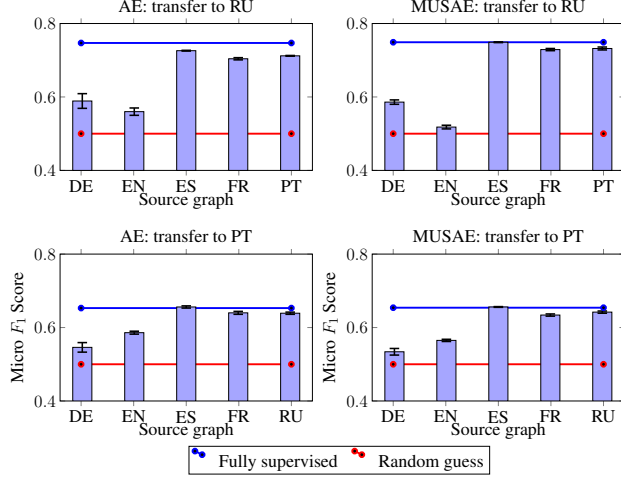


Figure 3. Transfer learning: mean micro F_1 score and standard error over 10 runs with the Twitch Portugal and Russia datasets as target and each other dataset as the source graph. Reference lines show performance for random guessing (red) and fully re-training on the target graph (blue).

The Twitch datasets contain disjoint sets of nodes but a common set of features \mathbb{F} . To perform transfer learning, we: (i) learn node and attribute embeddings for a source graph; (ii) train a regression model to predict a test attribute $f \notin \mathbb{F}$; (iii) re-run the node embedding algorithm on a target graph but with *feature embedding parameters fixed* (as learned in step (i)); and (iv) use the regression model to predict attribute f for target graph nodes. Figure 3 shows micro F_1 scores and standard error (over 10 runs) for the Twitch Portugal and Russia datasets as target and each other dataset as source graphs. The results confirm that *MUSAE* and *AE* learn feature embeddings that transfer between graphs with common features. Specifically, we see that performance always beats random guessing (red line) and that in some cases compares closely to re-training the feature embeddings and regression model on the target graph (blue line).

5.7. Scalability

We demonstrate the efficiency of our algorithms with synthetic data, varying the number of nodes and features per node. Figure 4 shows the mean runtime for sets of 100 experiments on Erdos-Renyi graphs with the number of nodes as indicated, 8 edges per node and the indicated number of unique features per node uniformly selected from a set of 2^{11} . The hyperparameters are in Table 2 except we perform a single epoch with asynchronous gradient descent.

For both *AE* and *MUSAE*, we see that (i) runtime is linear in

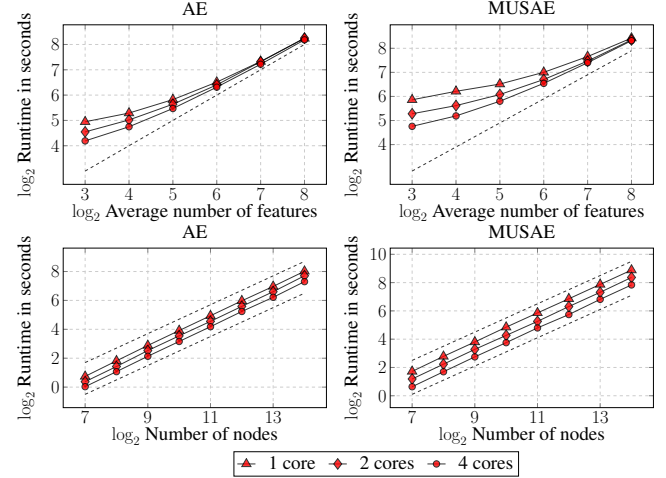


Figure 4. Training time as a function of average feature count and number of nodes. Dashed lines are linear runtime references.

the number of nodes and in the average number of features per node; and (ii) increasing the number of cores does not decrease runtime as the number of unique features per vertex approaches the size of the feature set. Observation (i) agrees with our complexity analysis in Section 4.3.

6. Discussion and Conclusion

We present attributed node embedding algorithms that learn local feature information on a pooled (*AE*) and multi-scale (*MUSAE*) basis. We augment these models to also explicitly learn local network information (*AE-EGO*, *MUSAE-EGO*), blending the benefits of attributed and neighbourhood-preserving algorithms. Results on a range of datasets show that distinguishing neighbourhood information at different scales (*MUSAE* models) is typically beneficial for the downstream tasks of node attribute and link prediction; and that the supplementary network information encoded by *EGO* models typically improves performance further, particularly for link prediction. Combining both, *MUSAE-EGO* outperforms other unsupervised attributed algorithms at predicting attributes and matches performance of neighbourhood-preserving embeddings in link prediction. Furthermore, by learning distinct node and feature embeddings, the *AE* and *MUSAE* algorithms enable transfer learning between graphs with common features, as we demonstrate on real datasets.

We derive explicit pointwise mutual information matrices that each of our algorithms implicitly factorise to enable future interpretability and potential analysis of the information encoded (e.g. as possible for *Word2Vec* (Allen et al., 2019)) and its use in downstream tasks. We see also that two widely used neighbourhood-preserving algorithms (Perozzi et al., 2014; 2017) are special cases of our models. All of our algorithms are efficient and scale linearly with the numbers of nodes and unique features per node.

References

- Abu-El-Haija, S., Perozzi, B., Al-Rfou, R., and Alemi, A. A. Watch Your Step: Learning Node Embeddings via Graph Attention. In *Advances in Neural Information Processing Systems*, 2018.
- Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., and Galstyan, A. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning*, 2019.
- Ahmed, N. K., Rossi, R., Lee, J. B., Kong, X., Willke, T. L., Zhou, R., and Eldardiry, H. Learning role-based graph embeddings. *arXiv preprint arXiv:1802.02896*, 2018.
- Allen, C., Balažević, I., and Hospedales, T. What the Vec? Towards Probabilistically Grounded Embeddings. In *Advances in Neural Information Processing Systems*, 2019.
- Cao, S., Lu, W., and Xu, Q. Grarep: Learning Graph Representations with Global Structural Information. In *International Conference on Information and Knowledge Management*, 2015.
- Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *International Conference on Knowledge Discovery and Data Mining*, 2019.
- Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
- Grover, A. and Leskovec, J. Node2Vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data Mining*, 2016.
- Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 2017.
- Huang, X., Li, J., and Hu, X. Accelerated attributed network embedding. In *SIAM International Conference on Data Mining*, 2017.
- Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.
- Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.
- Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.
- Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019.
- Levy, O. and Goldberg, Y. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, 2014.
- Liao, L., He, X., Zhang, H., and Chua, T.-S. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2257–2270, 2018.
- Lu, Q. and Getoor, L. Link-based classification. In *International Conference on Machine Learning*, 2003.
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations, Workshop Track Proceedings*, 2013a.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 2013b.
- Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, 2010.
- Namata, G., London, B., Getoor, L., Huang, B., and EDU, U. Query-driven active surveying for collective classification. In *International Workshop on Mining and Learning with Graphs*, 2012.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
- Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *International Conference on Knowledge Discovery and Data Mining*, 2014.
- Perozzi, B., Kulkarni, V., Chen, H., and Skiena, S. Don’t Walk, Skip!: Online learning of multi-scale network embeddings. In *International Conference on Advances in Social Networks Analysis and Mining*, 2017.
- Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. Network embedding as matrix factorization: Unifying Deepwalk, LINE, PTE, and Node2Vec. In *International Conference on Web Search and Data Mining*, 2018.
- Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. Struc2vec: Learning node representations from structural identity. In *International Conference on Knowledge Discovery and Data Mining*, 2017.

- Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-Scale Information Network Embedding. In *International Conference on World Wide Web*, 2015.
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. In *International Conference on Learning Representations*, 2018.
- Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*, 2019.
- Yang, C., Liu, Z., Zhao, D., Sun, M., and Chang, E. Y. Network Representation Learning with Rich Text Information. In *International Joint Conference on Artificial Intelligence*, 2015.
- Yang, H., Pan, S., Zhang, P., Chen, L., Lian, D., and Zhang, C. Binarized Attributed Network Embedding. In *IEEE International Conference on Data Mining*, 2018.
- Yang, S. and Yang, B. Enhanced Network Embedding with Text Information. In *International Conference on Pattern Recognition*, 2018.
- Zhang, D., Yin, J., Zhu, X., and Zhang, C. SINE: Scalable Incomplete Network Embedding. In *IEEE International Conference on Data Mining*, 2018.

Appendices

A. Proofs

Lemma 1. *The empirical statistics of node-feature pairs obtained from random walks give unbiased estimates of the joint probability of observing feature $f \in \mathbb{F}$ r steps (i) **after**; or (ii) **before** node $v \in \mathbb{V}$, as given by:*

$$\begin{aligned} \text{plim}_{l \rightarrow \infty} \frac{\#(v, f)_{\vec{r}}}{|\mathbb{D}_{\vec{r}}|} &= c^{-1}(\mathbf{D}\mathbf{P}^r \mathbf{F})_{v, f} \\ \text{plim}_{l \rightarrow \infty} \frac{\#(v, f)_{\leftarrow r}}{|\mathbb{D}_{\leftarrow r}|} &= c^{-1}(\mathbf{F}^\top \mathbf{D}\mathbf{P}^r)_{f, v} \end{aligned}$$

Proof. The proof is analogous to that given for Theorem 2.1 in (Qiu et al., 2018). We show that the computed statistics correspond to sequences of random variables with finite expectation, bounded variance and covariances that tend to zero as the separation between variables within the sequence tends to infinity. The Weak Law of Large Numbers (S.N.Bernstein) then guarantees that the sample mean converges to the expectation of the random variable. We first consider the special case $n = 1$, i.e. we have a single sequence v_1, \dots, v_l generated by a random walk (see Algorithm 1). For a particular node-feature pair (v, f) , we let $Y_i, i \in \{1, \dots, l - t\}$, be the indicator function for the event $v_i = v$ and $f \in \mathbb{F}_{i+r}$. Thus, we have:

$$\frac{\#(v, f)_{\vec{r}}}{|\mathbb{D}_{\vec{r}}|} = \frac{1}{l-t} \sum_{i=1}^{l-t} Y_i, \quad (1)$$

the sample average of the Y_i s. We also have:

$$\mathbb{E}[Y_i] = \frac{\deg(v)}{c} (\mathbf{P}^r \mathbf{F})_{v, f} = \frac{1}{c} (\mathbf{D}\mathbf{P}^r \mathbf{F})_{v, f}$$

$$\begin{aligned} \mathbb{E}[Y_i Y_j] &= \text{Prob}[v_i = v, f \in \mathbb{F}_{i+r}, v_j = v, f \in \mathbb{F}_{j+r}] \\ &= \underbrace{\frac{\deg(v)}{c}}_{p(v_i=v)} \underbrace{\mathbf{P}_{v:}^r}_{p(v_{i+r}|v_i=v)} \underbrace{\text{diag}(\mathbf{F}_{:f})}_{p(f \in \mathbb{F}_{i+r}|v_{i+r})} \underbrace{\mathbf{P}_{:v}^{j-(i+r)}}_{p(v_j=v|v_{i+r})} \underbrace{\mathbf{P}_{v:}^r}_{p(f \in \mathbb{F}_{j+r}|v_j=v)} \\ &\quad \underbrace{p(v_j=v, f \in \mathbb{F}_{i+r}|v_i=v)}_{p(v_j=v, f \in \mathbb{F}_{i+r}|v_i=v)} \end{aligned}$$

for $j > i + r$. This allows us to compute the covariance:

$$\begin{aligned} \text{Cov}(Y_i, Y_j) &= \mathbb{E}[Y_i Y_j] - \mathbb{E}[Y_i] \mathbb{E}[Y_j] \\ &= \frac{\deg(v)}{c} \mathbf{P}_{v:}^r \text{diag}(\mathbf{F}_{:f}) \underbrace{(\mathbf{P}_{:v}^{j-(i+r)} - \frac{\deg(v)}{c} \mathbf{1})}_{\text{tends to 0 as } j-i \rightarrow \infty} \mathbf{P}_{v:}^r \mathbf{F}_{:f}, \end{aligned} \quad (2)$$

where $\mathbf{1}$ is a vector of ones. The difference term (indicated) tends to zero as $j - i \rightarrow \infty$ since then $p(v_j = v|v_{i+r})$ tends to the stationary distribution $p(v) = \frac{\deg(v)}{c}$, regardless of v_{i+r} . Thus, applying the Weak Law of Large Numbers, the sample average converges in probability to the expected

value, i.e.:

$$\frac{\#(v, f)_{\vec{r}}}{|\mathbb{D}_{\vec{r}}|} = \frac{1}{l-t} \sum_{i=1}^{l-t} Y_i \xrightarrow{p} \frac{1}{l-t} \sum_{i=1}^{l-t} \mathbb{E}[Y_i] = \frac{1}{c} (\mathbf{D}\mathbf{P}^r \mathbf{F})_{v, f}$$

A similar argument applies to $\frac{\#(v, f)_{\leftarrow r}}{|\mathbb{D}_{\leftarrow r}|}$, with expectation term $\frac{1}{c} (\mathbf{F}^\top \mathbf{D}\mathbf{P}^r)_{f, v}$. In both cases, the argument readily extends to the general setting where $n > 1$ with suitably defined indicator functions for each of the n random walks (see (Qiu et al., 2018)). \square

Lemma 2. *The empirical statistics of node-feature pairs obtained from random walks give unbiased estimates of the joint probability of observing feature $f \in \mathbb{F}$ r steps **either** side of node $v \in \mathbb{V}$, given by:*

$$\text{plim}_{l \rightarrow \infty} \frac{\#(v, f)_r}{|\mathbb{D}_r|} = c^{-1}(\mathbf{D}\mathbf{P}^r \mathbf{F})_{v, f},$$

Proof.

$$\begin{aligned} \frac{\#(v, f)_r}{|\mathbb{D}_r|} &= \frac{\#(v, f)_{\vec{r}}}{|\mathbb{D}_{\vec{r}}|} + \frac{\#(v, f)_{\leftarrow r}}{|\mathbb{D}_{\leftarrow r}|} \\ &= \frac{1}{2} \left(\frac{\#(v, f)_{\vec{r}}}{|\mathbb{D}_{\vec{r}}|} + \frac{\#(v, f)_{\leftarrow r}}{|\mathbb{D}_{\leftarrow r}|} \right) \\ &\xrightarrow{p} \frac{1}{2} \left(\frac{1}{c} (\mathbf{D}\mathbf{P}^r \mathbf{F})_{v, f} + \frac{1}{c} (\mathbf{F}^\top \mathbf{D}\mathbf{P}^r)_{f, v} \right) \\ &= \frac{1}{2c} (\mathbf{D}\mathbf{P}^r \mathbf{F} + \mathbf{P}^{r\top} \mathbf{D}\mathbf{F})_{v, f} \\ &= \frac{1}{2c} ((\mathbf{D}\mathbf{P}^r + (\mathbf{A}^\top \mathbf{D}^{-1})^r \mathbf{D}) \mathbf{F})_{v, f} \\ &= \frac{1}{2c} ((\mathbf{D}\mathbf{P}^r + \mathbf{D}(\mathbf{D}^{-1} \mathbf{A}^\top)^r) \mathbf{F})_{v, f} \\ &= \frac{1}{c} (\mathbf{D}\mathbf{P}^r \mathbf{F})_{v, f}. \end{aligned}$$

The final step follows by symmetry of \mathbf{A} , indicating how the Lemma can be extended to directed graphs. \square

B. Embedding Model Hyperparameters

Our purpose was a fair evaluation compared to other node embedding procedures. Because of this each we tried to use hyperparameter settings that give similar expressive power to the competing methods with respect to target matrix approximation (Perozzi et al., 2014; Grover & Leskovec, 2016; Perozzi et al., 2017) and number of dimensions.

- **DeepWalk** (Perozzi et al., 2014): We used the hyperparameter settings described in Table 2. While the original *DeepWalk* model uses hierarchical softmax to speed up calculations we used a negative sampling based implementation. This way *DeepWalk* can be seen as a special case of *Node2Vec* (Grover & Leskovec, 2016) when the second-order random walks are equivalent to the first-order walks.

- *LINE₂* (Tang et al., 2015): We created 64 dimensional embeddings based on first and second order proximity and concatenated these together for the downstream tasks. Other hyperparameters are taken from the original work.
- *Node2Vec* (Grover & Leskovec, 2016): Except for the *in-out* and *return* parameters that control the second-order random walk behavior we used the hyperparameter settings described in Table 2. These behavior control parameters were tuned with grid search from the $\{4, 2, 1, 0.5, 0.25\}$ set using a train-validation split of 80% – 20% within the training set itself.
- *Walklets* (Perozzi et al., 2017): We used the hyperparameters described in Table 2 except for window size. We set a window size of 4 with individual embedding sizes of 32. This way the overall number of dimensions of the representation remained the same.
- The attributed node embedding methods *AANE*, *ASNE*, *BANE*, *TADW*, *TENE* all use the hyperparameters described in the respective papers except for the dimension. We parametrized these methods such way that each of the final embeddings used in the downstream tasks is 128 dimensional.
- *Classical GCN* (Kipf & Welling, 2017): We used the standard parameter settings described in this section.
- *GraphSAGE* (Hamilton et al., 2017): We utilized a graph convolutional aggregator on the sampled neighbourhoods, samples of 40 nodes per source, and standard settings.
- *GAT* (Veličković et al., 2018): The negative slope parameter of the leaky ReLU function was 0.2, we applied a single attention head, and used the standard hyperparameter settings.
- *MixHop* (Abu-El-Haija et al., 2019): We took advantage of the 0th, 1st and 2nd powers of the normalized adjacency matrix with 32 dimensional convolutional filters for creating the first hidden representations. This was fed to a feed-forward layer to classify the nodes.
- *ClusterGCN* (Chiang et al., 2019): Just as (Chiang et al., 2019) did, we used the *METIS* procedure (Karypis & Kumar, 1998) to decompose the graph. We clustered the graphs into disjoint clusters, and the number of clusters was the same as the number of node classes (e.g. in case of the Facebook page-page network we created 4 clusters). For training we used the earlier described setup.
- *APPNP* (Klicpera et al., 2019): The top level feed-forward layer had 32 hidden neurons, the teleport probability was set as 0.2 and we used 20 steps for approximate personalized pagerank calculation.
- *SGCONV* (Wu et al., 2019): We used the 2nd power of the normalized adjacency matrix for training the classifier.

C. Supervised Model Hyperparameters

Each model was optimized with the Adam optimizer (Kingma & Ba, 2015) with the standard moving average parameters and the model implementations are sparsity aware modifications based on PyTorch Geometric (Fey & Lenssen, 2019). We needed these modifications in order to accommodate the large number of vertex features – see the last column in Table 1. Except for the *GAT* model (Veličković et al., 2018) we used ReLU intermediate activation functions (Nair & Hinton, 2010) with a softmax unit in the final layer for classification. The hyperparameters used for the training and regularization of the neural models are listed in Table 6.

Table 6. Hyperparameter settings used for training the graph neural network baselines.

Parameter	Value
Epochs	200
Learning rate	0.01
Dropout	0.5
l_2 Weight regularization	0.001
Depth	2
Filters per layer	32

Except for the *APPNP* model each baseline uses information up to 2-hop neighbourhoods. The model specific settings when we needed to deviate from the basic settings which are listed in Table 6 were as follows: