Overview

- **Task.** Embedding a Graph: mapping nodes onto a d-dimensional continuous vector space.
- **Why?** Continuous Representations can then be used for task-specific ML models (e.g. Link Prediction or Node Classification).
- **Motivation.** Embedding methods based on Random Walks [2] produce powerful representations. However, they operate in two discrete steps (Random Walks then Representation Learning), and contain hyper-parameters (e.g. walk length) that must be tuned per graph.
- **Our Contribution.** We replace previously-fixed hyper-parameters with trainable parameters that we automatically tune by back-propagation while jointly learning node embeddings.
- **Method.** The hyper-parameters impose a distribution on every node's neighbourhood, which we term context distribution and denote \( Q \). We learn \( Q \) that best preserves the graph structure. We parameterize \( Q \) as an attention model on the power series of the graph transition matrix.
- **Results.** Our method significantly improves performance on Link Prediction by 20%-40% for all graphs. Further, the automatically-learned context distribution agrees with the optimal hyper-parameters we choose, if we manually tune existing methods.

Problem Statement

- Given a graph \( G = (V, E) \), an embedding algorithm produces matrix \( Y \in \mathbb{R}^{V \times d} \) with \( Y_v \) being the \( d \)-dimensional (embedding) representation for node \( v \in V \).
- Embeddings should preserve the structure of the graph: two node embeddings should be close if they are neighbors.
- Quality of embeddings can be measured on link-prediction tasks, as it is desirable to generalize to unseen information.

Classical Approach

Earlier approaches to Node Embeddings include Laplacian Eigenmaps [1]:

\[
\min \sum_{i=1}^{n} ||Y_u - Y_v||^2 \quad (1)
\]

Solved as eigendecomposition of graph Laplacian, which avoids trivial solutions and is equivalent to applying orthonormality constraints: \( Y_u A Y_v = I \).

2D Embedding of Karate Club Network [2]:

**Embedding Algorithm**

![Image of embedding algorithm]

**Input Graph**

![Image of input graph]

**Node Vector Space**

Review: Embedding via Random Walks

Introduced by Perozzi et al [2], this family of algorithms (including AsymPro[3], node2vec[4]):
- Operate in two disjoint steps of (i) Random Walk simulation, (ii) Representation Learning.
- Each of the steps has hyper-parameters
- Step (ii) is done by training a Skipgram model (from word2vec [5]) over the walk sequences.

Skipgram Context in Graphs (as used by DeepWalk, node2vec, etc.):

\[
C = \{ u, v \} \quad (6)
\]

C - 3 context choices

**Walk Graph**

![Image of walk graph]

**Node Vector**

Even though \( C \) can be manually tuned, most of the methods use word2vec implementation and therefore inherit the context sampling: \( c \sim (1/C) \).

**Deriving our Method: Embedding via Matrix Factorization**

- The random walk simulation, context sampling (\( c \sim (1/C) \)) and representation learning, can all be replaced by factorizing node-to-node co-occurrence matrix (similar to [7]).
- Let \( D \in \mathbb{R}^{N \times N} \) be a node-to-node where \( D_{uv} \) counts the event of \( u \) appearing \( c \) steps after \( v \) (with \( c \sim (1/C) \)) across all random walks.

**Objective Function:** We factorize \( D \) using negative-log graph likelihood objective of [3], written in our notation:

\[
\min_{L,R} \left\{ D \cdot \log (\sigma (L \cdot R^\top)) - \left[ A - \sigma \right] \cdot \log \left( 1 - \sigma (L \cdot R^\top) \right) \right\}. \quad (2)
\]

Where nodes are embedded into two (asymmetric) embedding spaces \( L, R \in \mathbb{R}^{N \times \frac{d}{2}} \) (i.e. \( Y = [L; R] \) and the pairwise edge scoring model is their outer-product. Indicator function \( \sigma \) is applied element-wise. \( \| \cdot \| \) norm of the matrix is sum of its entries, which are all positive since element-wise standard logistic \( \sigma : \mathbb{R} \rightarrow (0, 1) \).

\( D \) and Context Distribution \( Q \)

- Context Distribution \( Q \) assigns higher mass to nearby nodes, but the specific form of \( Q \) depends on hyper-parameters (e.g. \( C \) and choice of \( i \)). The value of \( Q \) affects values in the node-to-node matrix \( D \).
- As derived in our Appendix, with DeepWalk [2], \( \mathbb{E}[D] \) can be written as:

\[
\mathbb{E}[D_{G(walk)}] = \sum_{i} (1 - \frac{q_i}{\sum_{j} q_j}) \cdot C \cdot \log \left( \frac{1}{||C||} \right) . \quad (3)
\]

where \( R \) is a diagonal matrix containing the number of walks to be started from each node. We set the diagonal entries to 80.
- If GloVe [7] context sampling was used, we derive:

\[
\mathbb{E}[D_q] = R \cdot \sum_{i} (1 - \frac{q_i}{\sum_{j} q_j}) \cdot C \cdot \log \left( \frac{1}{||C||} \right) . \quad (4)
\]

We want to learn the coefficients of \( (T^\top) \). We propose the parametrized expectation:

\[
\mathbb{E}[D_q] = \sum_{i} (1 - \frac{q_i}{\sum_{j} q_j}) \cdot O_q(T^\top) \cdot (5)
\]

with:

- \( O_q : \mathbb{R} \rightarrow \mathbb{R} \)
- \( \sum_{i} (1 - \frac{q_i}{\sum_{j} q_j}) \cdot O_q(T^\top) \cdot (5)
\]

Our final objective extends Graph Likelihood with attention parameters

\[
\min_{L,R,q} \mathbb{E}[D_{G(walk)}] - \left[ A - \sigma \right] \cdot \log \left( 1 - \sigma (L \cdot R^\top) \right) \quad (6)
\]

is minimized w.r.t. node embeddings \( L,R \) and attention logit vector \( q \) parametrizes \( Q \)
- Attention parameters \( q \) can be thrown-away after training, as they are not part of the model and are not used for inference.

Experiment and Results

**Link Prediction**

- **Datasets:** We use the data splits of [3]. wiki-vote is a voting network of Wikipedia. ego-Facebook is a social network. co-AstroPh and co-HePTh are citation networks. PPI is protein-protein interactions network.
- **Baselines:** Laplacian EigenMaps [1]. Singular Value Decomposition (SVD) on adjacency matrix. DNGR is a deep auto-encoder network, node2vec (node2vec) [4] with two C values (full sweep is on left), and AsymPro is [3].

**Results**

**Dataset**

- wiki-vote
- ego-Facebook
- co-AstroPh
- co-HePTh
- PPI

**Methods**

- EigenMaps
- GloVe
- DNGR
- node2vec
- AsymPro

**Error Reduction**

- wiki-vote
- ego-Facebook
- co-AstroPh
- co-HePTh
- PPI

**Unsupervised Node Classification**

- Cora
- Citeseer
- Pubmed

**References**

[7] Pennington et al, GloVe, EMNLP 2014

Source code available at: http://sami.haija.org/graph/context