

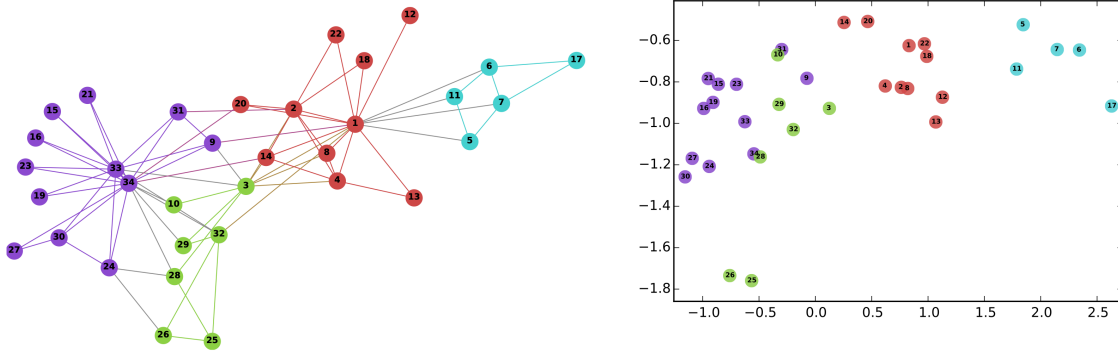
Embeddings by random walks

Most modern "shallow" node embedding methods build on random walks with restarts. Random walks offer a flexible, stochastic notion of node similarity that captures both local structure and higher-order, multi-hop relationships in a graph. In addition to their expressiveness, they are also highly efficient computationally.

Recall that our goal is to learn an embedding z_i for each node i such that

$$\text{similarity}(i, j) \approx z_i^T z_j.$$

A natural choice for $\text{similarity}(i, j)$ is the probability that nodes i and j co-occur along a random walk on the graph. Intuitively, if a random walk starting at node i reaches node j with high probability, then nodes i and j can be regarded as "similar" (e.g., close and strongly connected in the graph).



Learning objective & task

Let N_i denote a multiset of nodes visited by fixed-length random walks starting at node i . Then, our objective becomes to learn an embedding z_i for node i that is predictive of the nodes in its random walk neighborhood N_i .

$$\arg \max_Z \prod_i \Pr(N_i \mid z_i)$$

For numeric stability, the objective can be expressed as a log-likelihood

$$\arg \min_Z \sum_i -\log \Pr(N_i | z_i),$$

which is equivalent to

$$\arg \min_Z \sum_i \sum_{j \in N_i} -\log \Pr(j | z_i).$$

We parametrize the probability $\Pr(j | z_i)$ with a differentiable softmax function as

$$\Pr(j | z_i) = \frac{\exp z_i^T z_j}{\sum_j \exp z_i^T z_j}.$$

So, the final log-likelihood objective or loss function can be written as

$$\mathcal{L} = \sum_i \mathcal{L}^{(i)} = \sum_i \sum_{j \in N_i} -\log \frac{\exp z_i^T z_j}{\sum_j \exp z_i^T z_j}$$

and our learning task becomes

$$\arg \min_Z \mathcal{L}.$$

Notice that computing the loss function $\mathcal{L}$ directly requires $\mathcal{O}(n^2)$ time, but it can be efficiently approximated using negative sampling (details omitted here).

Stochastic gradient descent

The loss function $\mathcal{L}$ can be minimized by stochastic gradient descent as follows.

1. Randomly initialize the embedding z_i for each node i .
2. Iterate until the loss function $\mathcal{L}$ is decreasing (i.e. convergence):
 - randomly sample a node i and compute the gradient $\frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$ for each node j ;
 - update the embeddings as $z_j \leftarrow z_j - \eta \frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$, where η is the learning rate (e.g., $\eta = 0.01$).

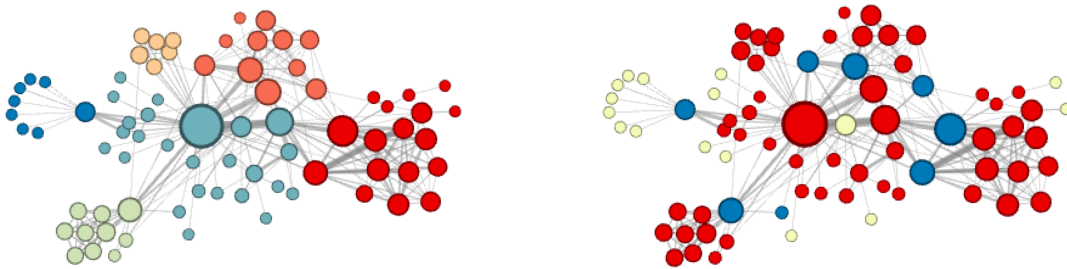
node2vec algorithm

The complete *node2vec* optimization procedure can be summarized as follows ([Grover & Leskovec, 2016](#)).

1. For each node i , simulate r random walks of length l (e.g., $r = 10$ and $l = 80$).
2. Optimize the loss function $\mathcal{L}$ using stochastic gradient descent until convergence.

The *node2vec* paper further introduces different biased random walk strategies. At each

step, the walker returns to the previous node with probability proportional to $\frac{1}{p}$, moves to a neighbor of the previous node with probability proportional to 1, and steps further away from the previous node with probability proportional to $\frac{1}{q}$. Here, $p \in \mathbb{R}^+$ is called the return parameter, and $q \in \mathbb{R}^+$ is referred to as the in-out or "walk-away" parameter, which balances between depth-first and breadth-first exploration of the graph. Typically, we set $p, q \in \{0.25, 0.5, 1, 2, 4\}$.



Other algorithms

The *DeepWalk* algorithm ([Perozzi et al., 2014](#)), a predecessor of *node2vec*, employs unbiased random walks and uses a slightly different optimization scheme. The *LINE* algorithm directly optimizes 1-hop and 2-hop node proximity probabilities ([Tang et al., 2015](#)). The *struc2vec* algorithm performs random walks on a modified graph, allowing it to capture structural similarities between nodes ([Ribeiro et al., 2017](#)). For a review of different embedding techniques, their applications, and comparative performance, see ([Goyal & Ferrara, 2018](#)).

