A COMPARATIVE EVALUATION ON NETWORK REPRESENTATION LEARNING TECHNIQUES

Konstantinos Ameranis **Isidora Kiourti** **Konstantinos Sotiropoulos** **Erasmo Tani**

Isidora Tourni

ABSTRACT

The abstract paragraph should be indented 1/2 inch (3 picas) on both left and right-hand margins. Use 10 point type, with a vertical spacing of 11 points. The word ABSTRACT must be centered, in small caps, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 INTRODUCTION

2 NETWORK EMBEDDINGS

2.1 MATRIX FACTORIZATION

MULTIDIMENSIONAL SCALING (MDS)

Multidimensional Scaling (MDS) is a method for creating a Euclidean embedding of data for which one has distance / dissimilarity information. For instance, given an $N \times N$ matrix of distances between N points, one can embed the points into $\mathbb{R}^k$ so as to preserve distance information. In particular, one can use MDS as a way to create useful features for graphs by considering the shortest path distance between vertices. MDS is similar to PCA, except instead of using correlation information, we make use of pointwise distances.

Classical Multidimensional Scaling works as follows: let D be the dissimilarity matrix. Then:

1. Let $D^{(2)}$ be the point-wise square of the distance matrix,
2. Let $J = I - \frac{1}{n}\vec{1}\vec{1}^T$,
3. Let $B = -\frac{1}{2}JD^{(2)}J$,
4. Find the top m eigenvalues of B $\lambda_1, ...\lambda_m$, and the corresponding eigenvalues $e_1, ..., e_m$,
5. Let $X = E_m\Lambda^{1/2}$.

(note to self:doesn't work if distance not euclidean)

Classical Multidimensional Scaling minimizes a loss function called *strain*:

$$Strain_D(x_1, ..., x_N) = \left(\frac{(\sum_{i,j} b_{i,j} - \langle x_i, x_j\rangle)^2}{\sum_{i,j} b_{i,j}^2}\right).$$

SPECTRAL EMBEDDING

Another way to embed a graph in Euclidean space is given by the spectral embedding. This method computes the k eigenvectors of the normalized Laplacian matrix $\mathcal{L}$ corresponding to the k smallest eigenvalues, and uses each of them as an embedding of the vertices into $\mathbb{R}$, resulting in an embedding into $\mathbb{R}^k$. The normalized Laplacian matrix is given by:

$$\mathcal{L} = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$$

One can prove that the quadratic form of the Laplacian is a relaxation to the minimum conductance cut problem defined as follows:

$$\underset{S \subseteq V}{\text{minimize}} \frac{|E(S, \overline{S})|}{\min\{vol(S), vol(\overline{S})\}}.$$

The eigenvectors of the Laplacian therefore act as optimizers of the relaxation, and tend to align points in space so as to keep connected points close to each other.

ISOMAP

Isomap is a method for manifold learning / non-linear dimensionality reduction. In a general setting, given data points living in a (non-linear) manifold in $\mathbb{R}^n$ we would like to embed them into lower dimensional space while preserving geodesic distances.

2.2 RANDOM WALKS

LINE

LINE is a network embedding model, able to handle very large, arbitrary types of graph networks $G = (V, E)$, (undirected, directed and/or weighted), by optimizing an objective which preserves both local and global network structures. Local structures are represented by the observed links in the networks, as for each pair of vertices linked by edge (u, v), the weight w_{uv} on that edge indicates the first-order proximity between u and v. Global structure is the second-order proximity between the vertices, determined through the shared neighborhood structures, as nodes with shared neighbors are likely to be similar. If $p_u = (w_{u,1}; \ldots; w_{u,|V|})$ denotes the first-order proximity of u with all the other vertices, then the second-order proximity between u and v is the similarity between p_u and p_v.

In both cases, an objective function is defined and optimized, and the difference in the variants of the model that are described lies on whether first- or second-order proximity (or both) is used, and on the method selected for optimizing this objective in each case. Using first-order proximity, the KL-divergence between the joint and the empirical distribution of vertices v_i and v_j for each undirected edge (i, j) is minimized. On the other hand, using second-order proximity, u'_i is defined as the representation of v_i, when it is treated as a specific "context", and u_i as the representation of v_i when it is treated as a vertex. In this case, the KL-divergence between the conditional and the empirical distribution over the contexts is minimized.

A negative sampling approach tackles the problem of trivial infinity solutions in the case of first-order proximity and that of the computationally expensive minimization of the objective in the case of second-order proximity. Multiple negative edges are sampled according to some noisy distribution for each edge (i, j), and an asynchronous Stochastic Gradient Descent algorithm is used, which, in each step, samples a mini-batch of edges and updates the model parameters. When edge weights have a high variance, scales of the gradients in SGD diverge, making it harder to find a good learning rate. Optimization via edge-sampling is then applied, by sampling from the original edges, with sampling probabilities proportional to the original edge weights. Sampled edges are treated as binary edges.

To combine first- and second-order proximity, a simple way is to concatenate the vector representations learned by both into a longer vector, and reweigh the dimensions, to balance the two representations. It was found that in practice, optimization takes $O(|E|)$ time, and the overall complexity of LINE is $O(d(K + 1))$, given that K is the number of negative samples and $d << |V|$ the dimension of the lower-dimensional space.

HARP

HARP is a meta strategy which solves the graph representation learning problem using a hierarchical approach. All methods described before could easily get stuck at a bad local minima as the result of poor initialization of the non-convex optimization. Moreover,these methods mostly aim to preserve local proximities in a graph but neglect its global structure.

HARP recursively coalesces the nodes and edges in the original graph to get a series of successively smaller but structurally similar graphs . These coalesced graphs, each with a different granularity, provide a view of the original graphs global structure. Starting from the most simplified form, each graph is used to learn a set of initial representations which serve as good initializations for embedding the next, more detailed graph. This process is repeated until we get an embedding for each node in the original graph.

HARP's method for multi-level graph representation learning consists of three parts:

- **Graph Coarsening:** starting from the original graph, $G = G_0$, a hierarchy of successively smaller graphs $G_0, G_1, \ldots, G_L$ is created. A hybrid graph coarsening scheme is developed, which is repeatedly applied to obtain a small graph. It combines two algorithms, edge collapsing and star collapsing, preserving first- and second-order proximity respectively. Edge collapsing is an edge selection and node merging algorithm, which arbitrarily merges nodes of edges into single nodes, providing a graph with at least half the edges of the original one. Star collapsing algorithm, on the other hand, considers star-like structured graphs, and merges nodes with the same neighbors into supernodes.
- **Graph Embedding:** Using a provided Graph Embedding algorithm, Graph Embedding is obtained on the Coarsest Graph G_L, a small sized graph providing a high quality representation.
- **Representation Prolongation and Refinement:** For each graph G_i, the graph representation of G_{i+1} is prolonged and taken as its initial embedding, Φ'_{G_i}, followed by applying a provided embedding algorithm to (G_i, Φ'_{G_i}) to further refine Φ'_{G_i} and obtain refined embedding Φ_{G_i}.

Processes of Graph Embedding, Prolongation and Refinement are then applied recursively to the larger graphs and their embeddings, until we obtain the graph embedding of the original graph, Φ'_{G_0}.

HARP is combined with a few state-of-the-art graph embedding methods (DeepWalk, LINE, Node2vec) to produce higher quality embeddings for all of them. Time Complexity of HARP(DW) is the same as that of the original DeepWalk, equal to $O(\gamma|V|tw(d + dlog|V|))$, where γ is the number of random walks, t is the walk length, w is the window size, d is the representation size, and $|V|$ is the number of nodes. Similarly, time complexity of HARP(LINE) is the same as that of LINE, $O(r|E|)$, linear to the number of edges in the graph, $|E|$, and the number of iterations over edges, r.

3 Graph Convolutional Networks

The recent popularity of convolutional neural networks **?**, and their success in classifying images and other tasks with a striking accuracy, has intrigued the scientific community with the question of whether -at least a variant of them- can be leveraged in learning tasks of graph networked data. The use of graph structures in computer vision**?**, as well in the representation of social networks **?**, has made this task look appealing and worth of investigation-research. However, using convolutional neural networks for networked data directly, is not straightforward and it poses significant challenges.

Challenges in Graph Convolutional Networks First and foremost, as CNNs were mainly used for image classification, they assume that data lie on a regular grid in a geometric space (most commonly Euclidean). On the contrary, graph data are a typical form of unordered data that lie in an irregular domain. Also, the heavy-tailed distribution of node degrees in real networks **?** makes the filtering-convolution part difficult, as it is not easy to define a constant-sized neighborhood and apply a localized filter, as in the traditional CNN case. Also, the pooling stage has to be defined as well.

Convolution in graphs First efforts to address the above mentioned issues by Bruna et al. **?** and introduce the architecture of CNNs to networked data, mainly draw from the field of Graph Signal Processing **?**. Graphs are generally considered undirected and their representation is given through their **Laplacian** $L = D - A$ or more commonly the normalized **Laplacian** $L = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. As this matrix is a real symmetric and positive semidefinite, it admits an eigenvector decomposition $L = U\Lambda U^T$, where U is a square orthonormal matrix with the eigenvectors as its columns, and Λ

is the diagonal matrix of eigenvalues. Letting $x \in \mathbb{R}^n$ be a feature vector of the nodes of a graph, the *graph fourier transform* is then defined as $\hat{x} = U^T x \in \mathbb{R}^n$ and its inverse as $x = U\hat{x}$. The **convolution operator** on a graph $\mathcal{G}$ is defined on the Fourier domain, as:

$$x *_{\mathcal{G}} y = U((U^T x) \odot (U^T y))$$

where $\odot$ denotes the element-wise Hadamard product.
Therefore, a signal x is filtered by g_θ, as:

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = U g_\theta(\Lambda)U^T x \tag{1}$$

where $g_\theta(\Lambda) = diag(\theta)$ is a non-parametric filter.
This approach has, however, the following limitations:

1. Filters are not localized
2. Their learning complexity scales with the dimensionality of the data $O(n)$
3. The computational cost of filtering is high- $O(n^2)$, due to the multiplication with the Fourier basis U.

CHEBNET**?**

4 COMPARATIVE EVALUATION

4.1 DATASETS

4.2 EVALUATION TASKS

4.3 COMPARATIVE EVALUATION

5 CONCLUSION