A Comparative Evaluation on Network Representation Learning Techniques

Konstantinos Ameranis **Panagiota 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

Graph data is pervasive in modern data analysis. In Computer Science, graphs have been an object of study for decades, representing a fundamental tool in the area of data structures and algorithms, and a model for systems ranging from computer networks and social networks to the World Wide Web. The impact of graphs has however vastly exceeded the boundaries of Computer Science, as these models have been used in fields as diverse as Biology, where they are used, for instance, to represent protein interaction networks, and Operation Research, in which they are used to model infrastructure networks, such as systems of roads or pipes.

In order to analyze network data it is often useful to obtain a representation of the vertices of the graph in Euclidean Space (an **embedding**). This allows us to leverage the power of machine learning and data mining techniques that are tailored to Euclidean setting and use them to make sense of the properties of the graph. In order for this to work, the embedding has to somehow encode some of the information and patterns that were contained in the original network. Some desirable properties the embedding should satisfy are the following Chen et al. (2018):

Adaptability Since data might be changing over time, the embedding should be efficiently updatable,
Scalability The embedding should be computable on large graphs,
Community aware The Euclidean distance in the embedding should preserve the community structure of the underlying graph,
Low dimensionality Having a low dimensional embedding often allows for better generalization.

The problem of finding the best such embedding has proved to be far from trivial, and while many solutions have been proposed, there appears to be no universal answer.

In this project we will look at some of the embedding techniques that have been developed and test their effectiveness in the context of a simple machine learning task.

Definitions

We now now introduce out objects of study.

A **(undirected) graph** is a pair (V, E), where V is a finite set of **vertices** (or nodes) and E is a collection of **edges** (/links). Each element of E is a two-element subset $\{v, u\}$ of V, encoding the property that the vertex $u \in V$ is connected to the vertex $v \in V$.

A graph **embedding** for a given graph $G = (V, E)$ is a function $\Phi : V \to \mathbb{R}^{|V| \times d}$.

2 Network Embeddings

2.1 Matrix Factorization

Multidimensional Scaling (MDS)

Multidimensional Scaling (MDS) (see Cox & Cox (2000) and Borg & Groenen (2003)) 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 $X = E_m\Lambda^{1/2}$.

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

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

Classical MDS only works for Euclidean spaces, so in the context of graphs, where the underlying metric is given by shortest path metric, we use a slightly different version of MDS. This variant, known as metric MDS, finds en embedding $x_1, ..., x_n$ minimizing the following objective function (stress):

$$\text{Stress}_D(x_1, ..., x_n) = \left(\sum_{i \neq j = 1, ..., N} (d_{ij} - ||x_i - x_j||)^2\right)^{1/2}$$

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:

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

Where D is the $n \times n$ diagonal matrix of degrees of vertices and A is the graph adjacency matrix. One can prove that the quadratic form of the Laplacian is a relaxation to the minimum conductance cut problem (see for instance Chung & Graham (1997)) 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. Note that eigenvectors of the normalized Laplacian are solutions to the generalized eigenvalue problem:

$$L\mathbf{x} = \lambda D\mathbf{x}$$

where L is the (non-normalized) Laplacian matrix: $L = D - A$ and D is defined as above. Spectral embeddings based on the Laplacian are common primitives used in other methods, such as manifold learning, (see for instance Belkin & Niyogi (2003)). These techniques are also at the heart of a common graph clustering algorithm known as spectral clustering **?**.

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 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.

As a way to better combine first- and second-order proximities, the authors propose to jointly train the objective function in the future. Also, they suggest that higher-order proximity approaches could be applied to provide a better result. Another objective could be to find new embeddings of heterogeneous information networks, meaning graphs with vertices of multiple types. Furthermore, the case of no observed connections between new and existing vertices could be explored in the future by resorting to other network information, such as the textual information of the vertices.

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 $G_0, G_1, \dots, G_L$ 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 a straightforward task 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:

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

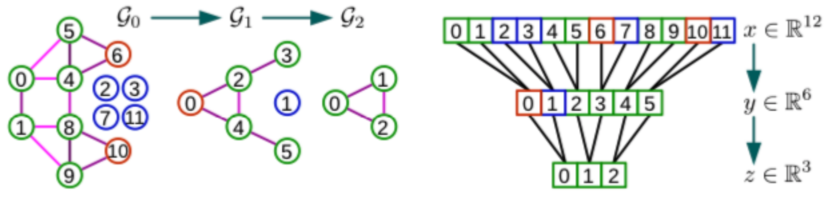

Figure 1: Pooling operation, using Greclus clustering and arranging nodes in a balanced binary tree

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.

Spectral CNN using Chebyshev Polynomials

Deferrard et al. **?** were the first to propose a way to deal with the above issues.
Localized Filter First of all, they propose the use of a polynomial filter:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \tag{2}$$

where the parameter $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients. As $(L^K)_{i,j} = 0$ whenever $d_{\mathcal{G}}(i,j) > K$, these spectral filters are K-localized.
Learning complexity Their learning complexity is equal to the support size of the filter, hence $O(K)$, as in conventional CNNs.
Computational cost of filtering As mentioned earlier, because of the multiplication of the filter with the Fourier basis U, the computational cost is relatively high, $O(n^2)$. For this reason, the express the filter as a Chebyshev polynomial of order k, that can be computed using the recurrence $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_o = 1$ and $T_1 = x$. Thus, the whole filtering operation can be written as $y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\Lambda})x$, where $T_k(\Lambda) \in \mathbb{R}^{nxn}$ is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $\widetilde{\Lambda} = 2L/\lambda_{max} - I_n$, using the recurrence $\bar{x}_k = 2\widetilde{\Lambda}\bar{x}_{k-1} - \bar{x}_{k-2}$, with $\bar{x}_0 = x$ and $\bar{x}_1 = \widetilde{\Lambda}x$. The cost of the filtering operation becomes then $O(K|E|)$, as it is a multiplication of a sparse matrix with a column vector of size K.
Graph Coarsening and Pooling Pooling on traditional convolutional neural networks, is performed usually on a patch of neighboring pixels on an image. However, in graphs, we can not explicitly define this notion of neighborhood as in a Euclidean grid. Therefore, the authors of this paper propose a multilevel clustering approach, that not only clusters similar vertices - in a topological fashion- together, but also produces coarser versions of the graph on each level, resembling closely the pooling operation of traditional CNNs. The multilevel clustering algorithm chosen is Graclus **?**. In order to perform pooling in a fast and memory efficient manner, nodes are organized in a balanced binary tree. Specifically, after coarsening, each node has either two childer, if it was matched at the finer level, or one, if it was not. Those singleton nodes are paired with fake nodes, having a neutral value -with respect to the activation function- as its input. A visualization of this operation, can be seen at Fig. 1.

1^{st} order approximation and semi-supervised classification

Kipf and Welling **?** build upon the previous work of Defferrard presented above, introducing simplifications that allow for significantly faster training times and higher predictive accuracy. Also, in their framework they introduce a way to perform semi-supervised classification tasks, hence their

model has the ability to classify nodes, even if most samples of the dataset are unlabeled.
A very simple form of a later -wise propagation rule would be of the form:

$$f(H^{(l+1)}, A) = \sigma(AH^{(l)}W^{(l)})$$

where $W^{(l)}$ is the weight matrix of the l-layer of the neural network, $H^{(l)}$ is the matrix of activations at the previous layer, and $\sigma(\cdot)$ could be a non-linear activation function, like ReLU.
However, this propagation rule, has some limitations. FIrst of all, a node only sums features of neighboring nodes, but not itself. Hence, we need to augment the adjacency matrix with self-loops, $\widetilde{A} = A + I_N$. Secondly, as long as A is not normalized, this summation will change the scale of features, hence we need a symmetric normalization, i.e. $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ to get rid of this problem. Thus, the propagation rule now becomes:

$$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{3}$$

As most real-world networks (e.g. social networks, citation networks, knowledge graphs) have distributions close to power-law, the authors propose to restrict the Chebyshev filters proposed by **?** to $K = 1$, expecting that this simplification will allow them to build deeper models and avoid any possible overfitting on local neighborhood structures. Setting, further, $\theta_0 = -\theta_1 = \theta$, leaves their model with only a single parameter to learn. Thus, the convolution operation is now:
$g_\theta * x \approx \theta(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}})x$

GRAPHSAGE

4 COMPARATIVE EVALUATION

4.1 DATASETS

4.2 EVALUATION TASKS

4.3 COMPARATIVE EVALUATION

5 CONCLUSIONS

OTHER DIRECTIONS

Most of the aforementioned context searching strategies used in network embedding models rely on a definition of context nodes that is identical for all networks and does not adapt to the properties or the application domain of each graph. Therefore, much work has been done on unifying different network embeddings under a general framework. The most significant ones are the following:

- **GraphAttention:** In Veličković et al. (2017), the proposed method automatically learns different attention parameters for different networks, by parameterizing the attention over the power series of a transition matrix.
- **GEM-D:** In Chen et al. (2017), the authors' approach decomposes graph embedding algorithms, such as Laplacian Eigenvectors, DeepWalk, LINE, and node2vec, into three building blocks: node proximity function, warping function and loss function. It shows that these algorithms can all be unified under this framework, and tests different design choices for each building block on real-world graphs, to pick the triple which works the best empirically.
- **NetMF:** In Qiu et al. (2018), the authors show that models such as DeepWalk, LINE, PTE, and node2vec, which use a negative sampling method, can be unified into a matrix factorization framework with closed forms, and provide the theoretical connections between skip-gram based network embedding algorithms and graph Laplacian. Based on these observations, they present the NetMF method for DeepWalk and LINE, as well as its approximation algorithm for computing network embedding, and show it offers significant improvements over the aforementioned models for conventional network mining tasks.

Similarly, many methods have been proposed to reduce the dependence of the existing embedding methods upon general loss functions and optimization models, which are not tuned for any particular task, and hence tend to have a suboptimal performance in comparison to end-to-end task-specific embedding methods. For example, in Abu-El-Haija et al. (2017), the authors propose a new definition of the graph likelihood function, designed specifically for link prediction.

REFERENCES

Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. Learning edge representations via low-rank asymmetric projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1787–1796. ACM, 2017.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Ingwer Borg and Patrick Groenen. Modern multidimensional scaling: Theory and applications. *Journal of Educational Measurement*, 40(3):277–280, 2003.

Haochen Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. A tutorial on network embeddings. *arXiv preprint arXiv:1808.02590*, 2018.

Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovačević, and Christos Faloutsos. Fast, warped graph embedding: Unifying framework and one-click algorithm. *arXiv preprint arXiv:1702.05764*, 2017.

Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. Chapman and hall/CRC, 2000.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, pp. 459–467, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5581-0. doi: 10.1145/3159652.3159706. URL `http://doi.acm.org/10.1145/3159652.3159706`.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.