GraphPOPE: Retaining Structural Graph Information Using Position-aware Node Embeddings

Jeroen B. den Boef[1,2], Joran Cornelisse[2] and Paul Groth[1]

[1]University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands
[2]Socialdatabase, Slego 1A, 1046 BM Amsterdam, The Netherlands

Abstract

Exponential computational cost arises when graph convolutions are performed on large graphs such as knowledge graphs. This computational bottleneck, dubbed the 'neighbor explosion' problem, has been overcome through application of graph sampling strategies. Graph Convolutional Network architectures that employ such a strategy, e.g. GraphSAGE, GraphSAINT, circumvent this bottleneck by sampling subgraphs. This approach improves scalability and speed at the cost of information loss of the overall graph topology. To improve topological information retention and utilization in graph sampling frameworks, we introduce Graph Position-aware Preprocessed Embeddings (GraphPOPE), a novel, feature-enhancing preprocessing technique. GraphPOPE samples influential anchor nodes in the graph based on centrality measures and subsequently generates normalized geodesic, Cosine or Euclidean distance embeddings for all nodes with respect to these anchor nodes. Structural graph information is retained during sampling as the position-aware node embeddings act as a skeleton for the graph. Our algorithm outperforms GraphSAGE on a Flickr benchmark dataset. Moreover, we demonstrate the added value of topological information to Graph Neural Networks.

Keywords

Graph Convolutional Networks, Graph Neural Networks, Graph Topology, Feature Embeddings

1. Introduction

Data that emphasizes relationships between data points, e.g. social networks, general knowledge, protein interactions, can be formally represented as graphs. Within machine learning there has been much interest in leveraging these graphs representations leading to the inception of graph learning and Graph Neural Networks (GNN) [1]. Graph neural networks have been successfully applied to a wide variety of tasks utilizing graph-structured data ranging from knowledge graphs to social networks [2, 3, 1, 4].

Many of the initial teething problems of GNNs have been resolved [5, 6, 7, 8, 9, 10]. Graph Convolutional Networks (GCN) combined a convolutional smoothing kernel with a spectral graph representation to achieve state of the art results on transductive node classification tasks [5]. While accurate in a transductive setting, this approach to node classification tasks fails to generalize well to unseen nodes [6]. GraphSAGE opened up the avenue for inductive graph

ISWC2021: Workshop on Deep Learning for Knowledge Graphs (DL4KG), October 25, 2021
jeroen@socialdatabase.com (J. B. d. Boef); joran@socialdatabase.com (J. Cornelisse); p.t.groth@uva.nl (P. Groth)
https://pgroth.com/ (P. Groth)
0000-0001-5649-2778 (J. B. d. Boef); 0000-0003-0183-6910 (P. Groth)

convolution models by learning general embedding generation functions for node features [6]. The GraphSAGE propagation rule utilizing a mean aggregator function is nearly equivalent to the one utilized in transductive GCNs and can be viewed as a linear approximation of localized spectral convolution. Additionally, primitive skip connections are performed by concatenating previous neighborhood representations of nodes with the current neighborhood representation. GraphSAGE also introduced a complementary NeighborSampler dataloader which improved scalability by introducing mini-batch training to Graph Convolutional Networks. The NeighborSampler aids embedding computation by sampling neighboring nodes iteratively and constructing mini-batches of nodes. Bipartite graphs are subsequently constructed to simulate the computation flow of GNNs. While inductive by nature and competitively accurate when introduced, the GraphSAGE model is constrained by its set neighborhood sampling function, as this restricts the convolutional kernel to a fixed size. The introduction of Graph Attention Networks (GAT) resolved this constraint by using an attention mechanism as a dynamic smoothing kernel [7]. When combined with a self-attention mechanism, this approach to graph convolution produces competitive results on both inductive and transductive tasks [7, 11].

1.1. Present work

Graph sampling architectures improve scalability and speed for Graph Convolutional Networks on large graphs at the cost of information loss with respect to overall graph topology. In an effort to improve topological information retention and utilization in graph sampling frameworks, we propose a general preprocessing technique for Neural Networks operating on graph-structured data, called GraphPOPE (Graph POsition-aware Preprocessed Embeddings). In this framework, topological information is embedded into the feature matrix through the generation of relative distance embeddings. By sampling *anchor nodes* from a given graph, identification points are determined. Normalized relative distance embeddings are then generated for all pairings of nodes and anchor nodes. These embeddings serve as a skeleton of the graph and identify which neighborhood a node belongs to. Intuitively, GraphPOPE embeddings can be interpreted as node2vec neighborhood embeddings for the whole graph, whereas node2vec generates second-order random walks neighborhood embeddings for individual nodes [12]. This makes GraphPOPE applicable to Multi Layer Perceptrons and local pooling models such as Graph Convolutional Networks alike, as topological information beyond the scope of a convolutional kernel is provided.

2. Related Literature

Conceptually, GraphPOPE is closely related to recent advances in GNNs that seek to improve topological information usage through position-aware convolutional layers. We consider GraphPOPE closely intertwined with graph sampling techniques as it mitigates topological information loss.

[0]Code implementations are publicly available on Github: https://github.com/JeroendenBoef/GraphPOPE

2.1. Graph Sampling Approaches

to GNNs favor scalability and speed over embedding or self-attention strategies by sampling subgraphs for training. Notable instances of this approach are Cluster-GCN and GraphSAINT, which both address the main computational bottleneck of GCNs [8, 9]. This computational bottleneck has been dubbed the *neighbor explosion* problem and is twofold: First, outputs of a GCN for a single node require data from neighbouring nodes in the previous layer of the network [8, 9]. Every layer within a GCN requires another n-hop neighbors where n depends on the convolutional kernel size, increasing computational cost exponentially for every layer. Second, back-propagation of a GCN requires all of the embeddings in the computation graph to be stored in GPU memory. Cluster-GCN proposes a solution to this bottleneck by preceding training with a clustering phase. Clusters of nodes belonging to dense subgraphs are identified during this clustering phase. These subgraphs are then used to restrict neighborhood search, acting as boundaries for the convolutional kernel. This relatively simple strategy introduces scalability to graph convolutional networks while reducing computational costs by a large margin. However, the employed clustering algorithm introduces additional heavy computational costs.

GraphSAINT adopts a similar approach to Cluster-GCN, marginally improving accuracy but substantially decreasing computation time [9]. The improved computational speed is mainly achieved through employment of inexpensive sampling algorithms, contrasting the expensive clustering algorithm utilized by Cluster-GCN.

2.2. Information Loss:

While sampling graphs during training mitigates the neighbor explosion problem and reintroduces scalability to GCNs, it nevertheless results in the loss of information. Restricting the GCN to specific clusters or completely disconnected subgraphs during training withholds or even removes edges and thus information from the graph. Chiang and colleagues identify this shortcoming for Cluster-GCN and introduce stochastic multiple clustering in an effort to reduce clustering bias and restore lost information simultaneously [8]. However, stochastic multiple clustering exclusively addresses the issue of cut edges during stochastic gradient descent batch updates, disregarding information loss preceding this phase.

A residual weakness within GNNs is their inability to distinguish between node positions with regards to the broader context of graph structure [13]. Not taking node features into account, two nodes can reside within opposite sides of a graph while having a topologically identical neighbourhood structure. Attempted heuristics range from attempts at deeper GNNs to node feature augmentation using position-aware convolutional layers [10].

3. Method: GraphPOPE

The inclusion of position-aware node embeddings as a general preprocessing technique for graph-based Neural Networks is motivated by the perceived topological information loss in graph sampling GCNs. Embedding this information into the node features before subgraph sampling could improve model performance. We first describe the geodesic GraphPOPE algorithm, which samples anchor nodes stochastically and subsequently generates position-aware

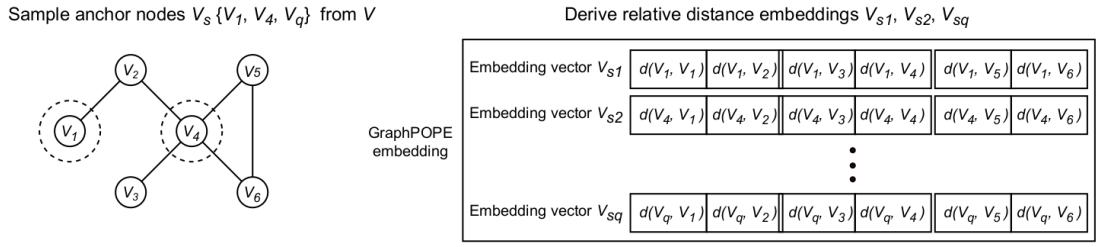

Figure 1: Schematic overview of the GraphPOPE embedding generation

node embeddings for all nodes in a given graph (Section 3.1). Embedding enhancement through biased anchor node sampling and algorithmic time complexity are detailed in Section 3.2. Finally, we introduce a faster, embedding space approximation of the geodesic GraphPOPE in Section 3.3. A schematic overview of the GraphPOPE algorithm is depicted in figure 1.

3.1. Geodesic Distance Embeddings

This section details the GraphPOPE anchor node sampler and geodesic distance embedding generator algorithm (Algorithm 1 - GraphPOPE-geodesic), which assumes the output matrix is concatenated with the feature matrix so that all nodes are enriched with their respective distance embeddings. Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ denote graph $\mathcal{G}$ with nodes $\mathcal{V}$ and edges $\mathcal{E}$, n the amount of anchor nodes $\mathcal{V}_s$ to sample, d the geodesic distance function used to derive relative node distances and $D^{N\times n}$ the geodesic distance matrix generated by GraphPOPE. The intuition behind this algorithm is that for each node v_i in the graph, normalized geodesic distances between this node and all sampled anchor nodes $\mathcal{V}_s$ are computed and added to feature vector $\mathbf{v}_v$, which is subsequently added to the relative distance matrix D at index i. Anchor nodes are sampled stochastically to reduce algorithm complexity and prevent bias in the data. The distance function employed for this computation is either a single-source or all-pairs shortest path algorithm, returning 0 if the target node is unreachable and $\frac{1}{d(v_i,u_j)}$ otherwise. As this distance function serves as an approximation of how many hops a node is from an anchor node, it can be replaced by similar but faster distance functions.

3.2. Biased Anchor Node Sampling

The vanilla GraphPOPE (Algorithm 1) avoids bias through stochastic sampling of anchor nodes. This approach to sampling has a potential drawback of sampling less influential nodes. We introduce biased anchor node sampling based on node centrality which replaces the stochastic sampler in Algorithm 1 and alleviates this aforementioned phenomenon.

In this algorithm (see Appendix A, Algorithm 2: Biased Sampler), centrality scores are derived for all nodes and the highest ranking nodes are selected as anchor nodes. This extension upon the vanilla GraphPOPE algorithm aims to increase and stabilize the amount of topological

[1] j is utilized as an enumeration of $u \in \mathcal{V}_s$ in line 3 to insert d_{ij} into vector v_v at index j

Algorithm 1 GraphPOPE-geodesic

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Sampling amount n; Distance function d
Output: Geodesic distance matrix $D^{N \times n}$

Stochastically sample n anchor nodes $\mathcal{V}_s$ from $\mathcal{V}$
for $v_i \in \mathcal{V}$ **do**
  for $u_j \in \mathcal{V}_s$ **do**
    $d_{ij} \leftarrow \frac{1}{d(v_i, u_j)}$
    Embedding vector $\mathbf{v}_v[j] \leftarrow d_{ij}$ [1]
  end for
  $D[i] \leftarrow \mathbf{v}_v$
end for

information encoded in the distance embeddings by selecting nodes with higher centrality scores.

The geodesic distance matrix generation of Algorithm 1 employs a single-source shortest path algorithm for distance function d with time complexity $O(V + E)$ [14]. With this function being utilized for every combination of sampled anchor node V_s with every node $v_i \in V$, this results in an overall complexity of $O(V(V_s(V + E)))$. This can be simplified to a notation of $O(V_s(V^2 + VE))$, which is close to a worst-case complexity of $O(V^4)$ for a densely connected, directional graph ($E = V^2$) with $V_s = V$. This is nevertheless an extreme scenario, divergent from most average cases. When treated as an all-pairs shortest path problem on a directed graph and computed in parallel, this complexity can be improved to $O(V(V + E))$. An example of such an approach is the Floyd-Warshall algorithm, which generates a complete mapping of all shortest paths in a given graph. While this all-pairs approach to the shortest path derivation reduces the complexity from approximately $O(V^3)$ to $O(V^2)$, it is substantially more costly with regards to memory usage.

Biased anchor node sampling through node centrality introduces additional time complexity. Betweenness centrality would likely identify well connected, influential nodes most accurately as it denotes the fraction of shortest paths that pass through a given node. Nodes with a high betweenness centrality score would logically be more connected than those with a lower score, and thus less likely to return 0 when fed into distance function d. As this centrality measure requires shortest path computation, biased sampling has similar scalability issues to the shortest path algorithms employed in Algorithm 1. As faster approximations for betweenness centrality, other measures such as eigenvector-, clustering coefficient-, degree-, farness- and closeness centrality are utilized for centrality function c [15].

3.3. Embedding Space Approximation

In order to resolve the exponential scaling complexity of GraphPOPE-geodesic, we propose an embedding space alternative. This algorithm (see Appendix A, Algorithm 3: GraphPOPE-node2vec) utilizes Node2vec to generate local neighborhood embeddings $\mathcal{V}_E$ for every node v_i in $\mathcal{V}$ [12]. Anchor nodes can then be sampled stochastically or with a bias by K-means clustering $\mathcal{V}_E$ into n clusters and utilizing the cluster centroids as *pseudo anchor nodes* $\mathcal{V}_{Es}$. Classical

K-means has a time complexity of $O(n^2)$ which can be reduced to $O(n)$ through cluster shifting [16]. Moreover, the necessity of biased sampling is reduced for this algorithm, as edge traversal is not utilized in the distance computation. As a result, information loss does not occur in a similar fashion to the geodesic distance calculation. Distance between $\mathscr{V}_E$ and $\mathscr{V}_{Es}$ is then calculated through parallelized matrix multiplications, which has a linear complexity of $O(n)$. Algorithms used for the distance function are Cosine similarity, Cosine distance and Euclidean distance.

4. Experiments

We evaluate position-aware node embeddings by performing node property predictions on two benchmark datasets: Flickr and PubMed [9, 17]. In all experiments, predictions are performed on nodes that are unseen during training but included in the preliminary GraphPOPE embedding generation. This is thus considered a supervised, transductive learning setting. We used Weights & Biases for experiment tracking and hyperparameter optimization [18]. Sections 4.1 and 4.2 detail the experimental setup and data, respectively.

4.1. Experimental Setup

Experiments were conducted with an Ubuntu OS, GTX 2060 and RTX 3060 NVIDIA GPUs and Intel i7-5820K and Intel gold 6130 CPUs. Geodesic distances are derived with Networkx and all models are implemented through a combination of Pytorch, Pytorch Geometric and an abstraction layer of Pytorch Lightning [19, 20, 21, 22]. Our baseline model is a vanilla GraphSAGE architecture, consisting of 2-3 SAGE convolutional layers with intermediate batch normalization layers, the GraphSAGE mini-batch NeighborSampler, a hidden dimension size of 256, a dropout rate of 0.5 and a cross-entropy loss function [23, 6, 21]. GraphPOPE models used for experimentation are divided into two categories, geodesic and node2vec approaches. Geodesic iterations consist of the vanilla GraphPOPE-stochastic version and its biased alternatives. Specifically betweenness-, closeness-, degree-, clustering coefficient-, eigenvector centrality and PageRank-based versions. Node2vec implementations are normalized Cosine and Euclidean distances supported by biased anchor node sampling through K-means clustering. All experiments were conducted utilizing identical architecture and hyperparameters with the exception of optimized batch sizes, amount of convolutional layers (2 or 3) and GraphPOPE settings.

Hyperparameter optimization was conducted separately per dataset for GraphPOPE-geodesic, GraphPOPE-node2vec and the vanilla GraphSAGE baseline to ensure unbiased comparison. These optimizations are implemented through hyperparameter sweeps with a Bayesian search method to optimize validation accuracy [18]. Sweeps are performed over n anchor nodes, batch size B and the amount of convolutional layers ℓ.

Experiments are run for a max of 300 epochs using early stopping mechanics and a learning rate monitor. A starting learning rate of 0.001 is employed which is reduced upon stagnation of validation loss. Finally, an Adam optimizer is used for all models and early stopping is provoked if validation accuracy does not increase for 20 epochs [24, 25].

The motivation behind this experimental setup is twofold. It allows us to assess whether graph sampling GCNs can be improved through introduction of additional topological information. Moreover, it reduces the possibility of experimental bias as the convolutional kernel introduces unfavourable conditions. Convolutional kernels already have access to topological information of local node neighborhoods, rendering the information gain of GraphPOPE less potent.

4.2. Data

Two graph datasets of contrasting size and density are employed. Test partition nodes are unseen during training with a separate dataloader that is exclusively instantiated upon conclusion of training.

Flickr dataset We use the Flickr dataset introduced with GraphSAINT [9]. In the dataset, edges denote shared metadata among images such as locations or users. Labels are manually merged tags and represent 7 entities such as animals, nature, humans, etc. Features are 500-D bag of words extractions of SIFT image descriptions. The graph contains roughly 89,250 nodes with 899,756 edges connecting them. Similar to Zeng and colleagues, edge weights are normalized in-degrees and a fixed-partition split is applied to the data, resulting in 50/25/25 train/val/test split [9].

Citation dataset The PubMed medical citation graph is used to assess performance on a smaller, less challenging node property prediction dataset [17]. In this directed graph, nodes represent scientific publications regarding diabetes research from the PubMed database, edges indicate outgoing citations and labels represent publication categories. The dataset consists of 19,717 nodes divided over 3 classes with 44,338 edges. We employ the FastGCN partition split, resulting in 500 validation and 1000 test nodes, leaving the remaining 18,217 nodes for training [26].

5. Results

We provide experimental results detailing accuracy metrics on the tasks, feature importance of n anchor nodes and hyperparameter sweeps. Reported accuracy scores are averages of 20 runs in a range of fixed global seeds to ensure reproducibility. Tables 1 detail the results. Accuracy scores are accompanied by their respective standard error and highest accuracy values are denoted in bold. on benchmarking tasks. Optimized hyperparameters are given in Appendix A, Table 2.

Table 1 shows an accuracy increment for all geodesic GraphPOPE models with respect to the baseline on the Flickr dataset. Moreover, GraphPOPE-geodesic implementations that employ biased sampling of anchor nodes generally experience more substantial accuracy gains. Contrastingly, node2vec-based approximations yield no improved performance. Results on PubMed indicate a homogeneous performance of 89% accuracy.

Results on Flickr indicate that configurations with more anchor nodes generally experience a more substantial increase in performance except for node2vec approximations. Increasing the amount of anchor nodes from 32 to 256 for an unoptimized, geodesic GraphPOPE with closeness centrality sampling raises accuracy from 51.98% to 53.05%. Moreover, validation accuracy yields an 88% positive correlation with the amount of anchor nodes over 90 hyperparameter sweeps

on the aforementioned configuration. PubMed experiments display a contrasting trend where the amount of anchor nodes provide diminishing returns. On this smaller dataset, the amount of anchor nodes have a linear correlation of -25%, resulting in an optimal configuration with 64 anchor nodes to maximize validation accuracy.

Table 1
Averaged prediction results over 20 runs on optimized hyperparameters. Biased K-means anchor node sampling is utilized for N2V implementations.

	Flickr	PubMed
Name	Acc	Acc
Betweenness centrality	52.93 ± 0.23	89.14 ± 0.67
Closeness centrality	52.55 ± 0.24	89.28 ± 0.65
Degree centrality	52.92 ± 0.23	89.32 ± 0.63
Clustering coefficient	52.63 ± 0.23	89.40 ± 0.59
Eigenvector centrality	52.48 ± 0.17	89.29 ± 0.60
PageRank	**52.94** ± 0.25	89.05 ± 0.55
Stochastic	52.75 ± 0.24	**89.55** ± 0.56
N2V-cdist	51.70 ± 0.29	89.52 ± 0.39
N2V-Euclidean	51.68 ± 0.30	89.52 ± 0.39
Baseline	51.78 ± 0.17	89.51 ± 0.35

6. Discussion

Our experimental results on the Flickr dataset display trends of accuracy gain for GraphPOPE-enhanced architectures with a more substantial improvement on larger datasets. Additionally, accuracy gains improve upon biased anchor node sampling. The amount of anchor nodes correlates positively with an increase in validation accuracy. This suggests that additional topological information can be beneficial to sampling-assisted Graph Convolutional Networks. Specifically for larger graphs, where the fraction of topological information beyond the convolutional kernel is higher.

GraphPOPE-node2vec poses an ineffective embedding-space alternative. Whereas computation complexity is improved, model accuracy is not. This phenomenon might be explained by the fact that convolutional kernels employed by GCNs already have access to the local neighborhood information encoded by node2vec.

Scalability is a recurring bottleneck for GNNs. Our time complexity stems from the algorithm's geodesic distance calculation. Overcoming these scalability issues could prove beneficial for graph learning on large datasets, given the substantial accuracy increments on such graphs. Embedding-space approximations of the distance calculation could provide a solution for the memory and computation time bottlenecks, removing the need for costly edge traversal operations. Alternatively, models could be trained to approximate the distance function. Finally, deep learning alternatives for the identification of influential nodes in the graph could accelerate performance of biased anchor node sampling, potentially improving another time complexity component.

7. Conclusion

We introduced GraphPOPE, a novel prepossessing technique designed to improve topological information retention and utilization in Graph Neural Networks. Our algorithm is applicable to any Neural Network that has access to graph-structured data and operates through position-aware node embedding generation. We demonstrate an accuracy gain on graph benchmarking datasets Flickr and PubMed with the application of our position-aware node embeddings, which can be improved additionally at the cost of additional time complexity. Our experimental results indicate that larger graphs benefit more from increased amounts of topological information retention. Finally, we propose approximations for future research to reduce time complexity, thus increasing applicability to real-world scenarios.

References

[1] W. L. Hamilton, Graph representation learning, Synthesis Lectures on Artifical Intelligence and Machine Learning 14 (2020) 1–159.

[2] W. Fan, Y. He, Y. Ma, E. Zhao, D. Yin, Q. Li, J. Tang, Graph neural networks for social recommendation, arXiv (2019) 417–426.

[3] S. Arora, A survey on graph neural networks for knowledge graph completion (2020). arXiv:2007.12374.

[4] T. Thanapalasingam, L. van Berkel, P. Bloem, P. Groth, Relational graph convolutional networks: A closer look, CoRR abs/2107.10015 (2021). arXiv:2107.10015.

[5] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).

[6] W. L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, Advances in Neural Information Processing Systems 2017-Decem (2017) 1025–1035. arXiv:1706.02216.

[7] P. Velicković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, arXiv (2017) 1–12. arXiv:1710.10903.

[8] W. L. Chiang, Y. Li, X. Liu, S. Bengio, S. Si, C. J. Hsieh, Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks, Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019) 257–266. doi:10.1145/3292500.3330925. arXiv:1905.07953.

[9] H. Zeng, H. Zhou, A. Srivastava, V. Prasanna, R. Kannan, GraphSAINT: Graph sampling based inductive learning method, arXiv (2019). arXiv:1907.04931.

[10] J. You, R. Ying, J. Leskovec, Position-aware graph neural networks, 36th International Conference on Machine Learning, ICML 2019 2019-June (2019) 12372–12381. arXiv:1906.04817.

[11] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, Open graph benchmark: Datasets for machine learning on graphs, arXiv preprint arXiv:2005.00687 (2020).

[12] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[13] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, arXiv preprint arXiv:1810.00826 (2018).
[14] A. Bhargava, Grokking Algorithms: An illustrated guide for programmers and other curious people, Simon and Schuster, 2016.
[15] X. He, N. Meghanathan, Alternatives to betweenness centrality: A measure of correlation coefficient, 2016. doi:10.5121/csit.2016.61301.
[16] M. Pakhira, A linear time-complexity k-means algorithm using cluster shifting, 2014. doi:10.1109/CICN.2014.220.
[17] Z. Yang, W. W. Cohen, R. Salakhutdinov, Revisiting semi-supervised learning with graph embeddings, 2016. arXiv:1603.08861.
[18] L. Biewald, Experiment tracking with weights and biases, 2020. URL: https://www.wandb.com/, software available from wandb.com.
[19] A. Hagberg, P. Swart, D. S Chult, Exploring network structure, dynamics, and function using NetworkX, Technical Report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library (2019) 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
[21] M. Fey, J. E. Lenssen, Fast graph representation learning with pytorch geometric, arXiv preprint arXiv:1903.02428 (2019).
[22] W. Falcon, et al., Pytorch lightning, GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning 3 (2019).
[23] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, CoRR abs/1502.03167 (2015). URL: http://arxiv.org/abs/1502.03167. arXiv:1502.03167.
[24] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
[25] I. Loshchilov, F. Hutter, Fixing weight decay regularization in adam, CoRR abs/1711.05101 (2017). URL: http://arxiv.org/abs/1711.05101. arXiv:1711.05101.
[26] J. Chen, T. Ma, C. Xiao, Fastgcn: Fast learning with graph convolutional networks via importance sampling, CoRR abs/1801.10247 (2018). URL: http://arxiv.org/abs/1801.10247. arXiv:1801.10247.

A. Appendix

Algorithm 2 Biased sampler

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Sampling amount n; Centrality function c
Output: Anchor nodes $\mathcal{V}_s$

$C \leftarrow c(\mathcal{G})$
Sample n highest C_i anchor nodes $\mathcal{V}_s$ from $\mathcal{V}$

Algorithm 3 GraphPOPE-node2vec

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; Sampling amount n; Distance function d; Node2vec algorithm z; K-means clustering algorithm k; Bias setting $b \in \{True, False\}$
Output: Normalized embedding distance matrix $\tilde{D}^{N \times n}$

Node2vec embedding $\mathcal{V}_E \leftarrow Z(\mathcal{G})$
if $b == True$ **then**
    clustering centroids $C \leftarrow K(\mathcal{V}_E)$
    Sample n pseudo anchor nodes $\mathcal{V}_{Es}$ from C
else
    if $b == False$ **then**
        Stochastically sample n anchor nodes $\mathcal{V}_{Es}$ from $\mathcal{V}_E$
    end if
end if
$D \leftarrow d(\mathcal{V}_E, \mathcal{V}_{Es})$
Normalize D

Table 2
Optimized hyperparameter settings for GraphSAGE and GraphPOPE-enhanced GraphSAGE. Hyperparameters depicted are n anchor nodes $\mathcal{V}_s$, batch size B and convolutional layers ℓ.

	Flickr			PubMed		
Name	$n\mathcal{V}_s$	B	ℓ	$n\mathcal{V}_s$	B	ℓ
GraphPOPE-geodesic	256	1550	3	64	800	2
GraphPOPE-node2vec	64	625	3	256	750	3
Baseline	-	3550	2	-	2200	2