Scalable algorithms for graph-based semi-supervised learning with embedding

Mikhail Kamalov

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Mise a l'échelle des algorithmes pour l'apprentissage semi-supervisé basé sur des graphes avec le plongement

Mikhail Kamalov

Inria Sophia Antipolis - Méditerranée

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur

Dirigée par : Konstantin Avrachenkov
Soutenue le : 1 Décembre 2022

Devant le jury, composé de :
Marianne Clausel, Université de Lorraine, Rapporteur
Khalid Benabdeslem, Université Claude Bernard Lyon 1, Rapporteur
Auréelie Boisbunon, Ericsson, Examinateur
Paulo Gonçalves, Inria Rhone-Alpes, Examinateur
Christophe Crespelle, I3S, Université Côte d'Azur, Examinateur
Jonathan Daeden, MyDataModels, Invité
Konstantin Avrachenkov, Inria Sophia-Antipolis, Directeur de thèse

Inria

Scalable algorithms for graph-based semi-supervised learning with embedding

Jury:

President		
Christophe Crespelle	Professor	I3S, Université Côte d'Azur
Reviewers		
Marianne Clausel	Professeur	Université de Lorraine
Khalid Benabdeslem	Associate Professeur	Université Claude Bernard Lyon 1
Examiners		
Aurélie Boisbunon	Docteure	Ericsson
Paulo Gonçalves	Directeur de Recherche	Inria Rhone-Alpes
Christophe Crespelle	Professeur	I3S, Université Côte d'Azur
Invited		
Jonathan Daeden	Docteur	MyDataModels company
Director		
Konstantin Avrachenkov	Directeur de Recherche	Inria Sophia-Antipolis

Résumé. De nos jours, l'apprentissage semi-supervisé basé sur les graphes (GB-SSL) est un domaine en plein essor pour classer les nœuds d'un graphe avec un nombre extrêmement faible de nœuds labélisés. Cependant, les algorithmes GB-SSL ont deux limites générales : la première est la complexité mémoire/temps qui se présente dans tous les algorithmes GB-SSL de pointe sur de larges graphes. En particulier, la forte consommation de mémoire se produit dans les réseaux de convolution de graphes et conduit à des problèmes d'OOM (Out of Memory) sur GPU ou RAM; la seconde apparaît dans tous les algorithmes GB-SSL basés sur la perte de régularisation Laplacienne. La contribution majeure de cette thèse est divisée en deux parties afin de proposer des stratégies qui garantiraient d'éviter les restrictions mentionnées ci-dessus. Dans la première partie de cette thèse, nous proposons un nouvel algorithme linéaire appelé Markov-Batch Stochastic Approximation (MBSA) pour résoudre le PageRank Personnalisé. MBSA met à jour des lots de nœuds et propose un compromis significativement meilleur que les autres modèles linéaires entre la consommation de mémoire et le taux de convergence pour un résultat de classification optimal. Ensuite, nous proposons un nouveau réseau de convolution de graphes à échelle, appelé MBSA-NN, qui intègre notre MBSA linéaire. Le MBSA-NN évite les problèmes d'OOM et réduit considérablement la consommation de temps et de mémoire sur GPU et RAM. Nous avons appliqué le MBSA-NN à plusieurs grands ensembles de données, et nous avons montré qu'il peut traiter des graphes avec plus de 10M nœuds et 2M de caractéristiques en une minute sur une machine standard, y compris le temps de prétraitement, d'apprentissage et d'inférence. De plus, nous montrons qu'il a une consommation mémoire/temps significativement améliorée et une précision compétitive par rapport aux meilleurs algorithmes de mise à l'échelle GB-SSL les plus récents. La deuxième partie de cette thèse se concentre sur les solutions aux problèmes de perte de régularisation du Laplacien. Pour cette raison, nous proposons un nouveau cadre appelé Graph Diffusion & PCA (GDPCA). Ce cadre combine une analyse en composantes principales modifiée avec la perte supervisée classique et la perte de régularisation laplacienne. GDPCA permet de traiter le cas où la matrice d'adjacence présente des *Arêtes binaires* et évite la *Malédiction de la dimensionnalité*. De plus, GDPCA peut être appliqué à des ensembles de données non graphiques, tels que des images, en construisant un graphe de similarité. En outre, nous proposons un cadre qui intègre PageRank SSL dans un modèle génératif (GenPR). GenPR joint l'entraînement de la représentation de l'espace latent des nœuds et la propagation des labels à travers la matrice d'adjacence repondérée par les similarités des nœuds dans l'espace latent. Nous démontrons qu'un modèle génératif peut améliorer la précision et réduire le nombre d'étapes d'itération pour PageRank SSL. En outre, nous montrons comment intégrer

MBSA dans le cadre de GenPR pour fournir le régime de formation par lots de GenPR. Enfin, nous proposons un cadre SSL flexible basé sur l'empilement des algorithmes GDPCA et de Zoetrope Genetic Programming dans un nouveau cadre : PaZoe. Ce cadre d'auto-labélisation montre que les algorithmes basés sur les graphes et les algorithmes non basés sur les graphes améliorent conjointement la qualité des prédictions et sont plus performants que chaque composant pris séparément. Nous montrons également que PaZoe surpasse les algorithmes SSL de pointe sur des jeux de données réels. Notez que l'un des ensembles de données a été généré par nos soins, en prenant les données d'un équipement industriel classé pour imiter les moteurs à courant continu pendant leur fonctionnement.

Mots clés. Apprentissage Semi-Supervisé, Réseaux de Neurones, Approximation Stochastique, Personalized PageRank

Abstract. Nowadays, graph-based semi-supervised learning (GB-SSL) is a fast-growing area of classifying nodes in a graph with an extremely low number of labelled nodes. However, the GB-SSL algorithms have two general limitations: the first is the memory/time complexity that arises in all state-of-the-art GB-SSL algorithms on extremely large graphs. In particular, the high memory consumption occurs in graph convolution networks and leads to Out of Memory (OOM) issues on GPU or RAM; (ii) the second one appears in all GB-SSL algorithms based on Laplacian regularization loss. This thesis' major contribution is divided into two parts in order to suggest strategies that would guarantee to avoid the restrictions mentioned above. In the first part of this thesis, we propose a novel linear algorithm called Markov-Batch Stochastic Approximation (MBSA) for solving Personalized PageRank. MBSA updates nodes batches and proposes a significantly better tradeoff between memory consumption and convergence rate for an optimal classification result than other linear models. Then, we propose a novel scaling graph convolution network, denoted as MBSA-NN, which embeds our linear MBSA. MBSA-NN avoids OOM issues and significantly reduces time and memory consumption on GPU and RAM. We applied MBSA-NN on several very large datasets, and we showed that it can handle graphs with more than 10M nodes and 2M of features under one minute on one standard machine, including preprocessing, training and inference time. Furthermore, we show that it has significantly improved memory/time consumption and competitive accuracy concerning the latest best GB-SSL scaling algorithms. The second part of this thesis focuses on solutions to Laplacian regularization loss issues. For that reason, we propose a novel framework called Graph Diffusion & PCA (GDPCA). This framework combines a modified Principal Component Analysis with the classical supervised loss and Laplacian regularization loss. GDPCA allows handling the case where the adjacency matrix presents through *Binary edges* and avoids the *Curse of dimensionality*. Also, GDPCA can be applied to non-graph datasets, such as images, by constructing a similarity graph. Furthermore, we propose a framework that embeds PageRank SSL in a generative model (GenPR). GenPR joint training of nodes latent space representation and label spreading through the reweighted adjacency matrix by node similarities in the latent space. We demonstrate that a generative model can improve accuracy and reduce the number of iteration steps for PageRank SSL. Moreover, we show how to embed MBSA into the GenPR framework for providing the batch training regime of GenPR. Finally, we propose a flexible SSL framework based on stacking GDPCA and Zoetrope Genetic Programming algorithms into a novel framework: PaZoe. This self-labelling framework shows that graph-based and non-graph based algorithms jointly improve the quality of predictions and outperform each component taken alone. We also

show that PaZoe outperforms state-of-the-art SSL algorithms on real datasets. Note that the one of the datasets was generated in house, taking data from industrial graded equipment to mimic DC motors during operation.

Key words. Semi-Supervised Learning, Neural Networks, Stochastic Approximation, Personalized PageRank

Acknowledgements

I want to start by expressing my gratitude to Konstantin, my adviser, for his kindness and all of the helpful guidance and knowledge. Also, I want to say great thanks to the MyDataModels company for supporting my PhD. I am very happy that I had a chance to work with so many brilliant, kind and smart peoples such as Ingrid, Boris, Magomed from MyDataModels and, Abhishek, Tarek, Othman, Guilherme, Max, Chuan, Kishor, from Inria. Also, I want to thank our team assistant Jane for her kind help in many important matters. Special thanks to Aurélie, Jonathan and Andrei for insightful criticism of my works and for support with adaptation in France. Also, to Professor Borkar for so fruitful collaboration during my internship at IIT Bombay.

I am very grateful to the people who supported me despite the distance. I would like to thank my grandparents, Ludmila, Tatiana, Yurii, my mother Natalia, mother-in-law Olga, and father-in-law Andrei. Especially thanks to my grandfather Yakov who cared about me regardless of whatever and continues to care about me even from heaven.

I am also fortunate enough to have friendships that helped me a lot in reviewing my works, especially Yulia, Sergey, Vladimir Sasha, Tatiana, Ilya and Vladimir. Additional thanks to the advisor of my master thesis Vladimir Dobrynin for his goodness and all of the valuable knowledge. Moreover, I am really grateful to my close friends for never giving up on me, no matter what, specially Rodion, Yurii and Jenya. Thank you all.

Finally and most significantly, I would like to dedicate this thesis to my wife and son, Alena and Mikhail, my closest people in the world and my staunchest defenders and supporters. I can never express my gratitude to them enough for all that they have done and given up for me to be here. This thesis is *to them, for them, and because of them.*

Contents

List of Figures

List of Tables

List of Notations

Symbol	Meaning
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	an undirected and unweighted graph
$n = \lvert\mathcal{V}\rvert$	the number of nodes
$e = \lvert\mathcal{E}\rvert$	the number of edges
$\mathcal{V}_l$	the set of labeled nodes
$\mathcal{V}_u$	the sets of unlabeled nodes
$n_l = \lvert\mathcal{V}_l\rvert$	the number of labeled nodes
$n_u = \lvert\mathcal{V}_u\rvert$	the number of unlabeled nodes
$A = [A_{i,j}]_{i,j=1}^{n,n} \in \mathbb{R}^{n \times n}$	the adjacency matrix
$D = \text{diag}(D_{i,i})_{i=1}^{n}$	the degree matrix
$D_{i,i} = \sum_{j=1}^{n} A_{i,j}$	an element of the degree matrix
$\widetilde{A} = D^{-\delta} A D^{\delta-1} = [\widetilde{A}_{i,j}]_{i,j=1}^{n,n} \in \mathbb{R}^{n \times n}$	a regularized adjacency matrix
$\delta \in \{0, 1, 0.5\}$	a regularization parameter
$X = [X_i]_{i=1}^{n} \in \mathbb{R}^{n \times d}$	a matrix of node features
d	the number of input features
$Y = [Y_i]_{i=1}^{n} \in \mathbb{R}^{n \times c}$	a matrix of node classes
c	the number of classes
$W_{(l)}^{t} \in \mathbb{R}^{d \times m}$	a trainable weight matrix
m	the size of latent space representation of node features
l	the number of the layer in neural network;
t	the number of iterations

Chapter 1

Introduction

This introductory chapter provides an overview of the graph-based semi-supervised learning (GB-SSL) domain that motivates and forms various problems studied in this thesis. This chapter is structured as follows: i) at the beginning, we show the general idea, which lies under the hood of semi-supervised learning and explain the highly demanded applications of the GB-SSL; ii) we present an overview of existing fast-growing directions in the GB-SSL with a detailed explanation of the difference in their sub-directions; iii) then, we define the existing general critical/non-critical and specific limitations in the GB-SSL domain and list the set of goals we wish to achieve; iv) finally, we state the thesis contributions and present how they are organized in the following chapters.

1.1 General overview of Semi-supervised learning

Semi-supervised learning (SSL) is widely used to solve classification tasks with an extremely low amount of labelled data points. Nowadays, the area of SSL for classification tasks consists of two main research areas: the graph-based SSL (GB-SSL), where besides the object features, we can utilise the graph structure of data (e.g., citation and social networks where features are words from paper/post and edges are citations between papers/posts etc.) and non-graph based SSL (e.g., an image where we have just the image features, classification of time series etc.). The main ideas of these two areas are presented below:

- The graph-based SSL (GB-SSL) algorithms rely on a classical diffusion-based idea that uses the graph structure to spread the node class information. The principal diffusion-based algorithms are Label Propagation (LP) [3], PageRank SSL (PRSSL) [4], manifold regularization (ManiReg) [5]. More complex modifications of classical diffusion-based idea lean on graph convolution networks' application. The graph convolution networks convolve the graph structure with node features for the classification. The principal graph convolution algorithms are Graph Convolution Network (GCN) [6], Graph attention network (GAT) [7], the jumping knowledge network with concatenation (JK) [8], and Graphite [7]. Figure 1.1 shows the difference between the diffusion-based idea and the idea of graph convolution networks;

- The non-graph based SSL (non-GB-SSL) algorithms are based on the idea of extending default classification loss (e.g. hinge loss, cross-entropy) by customized semi-supervised regularization, such as in the transductive SVM (TSVM) [9], SSL logistic regression [10] or on the idea of the similarity learning as in K-nearest neighbours (KNN) [11]. For the non-graph based SSL, the complex modifications of the above ideas are based on applying the neural networks to blend the unsupervised and supervised losses. In particular, the unsupervised loss can be defined as variational autoencoder (VAE) loss, and the supervised loss can consider the classification loss as in the semi-supervised VAE (VAESSL) [12], the AtlasRBF [10] and contractive autoencoder (CAE) [13].

We would like to point out that the current work focuses on resolving issues in the graph-based semi-supervised learning (GB-SSL) since, nowadays, GB-SSL is a fast-growing area of research. In particular, GB-SSL algorithms are widely used for various tasks in real life: Scientific paper classification in citation networks [6, 14] where articles are nodes and

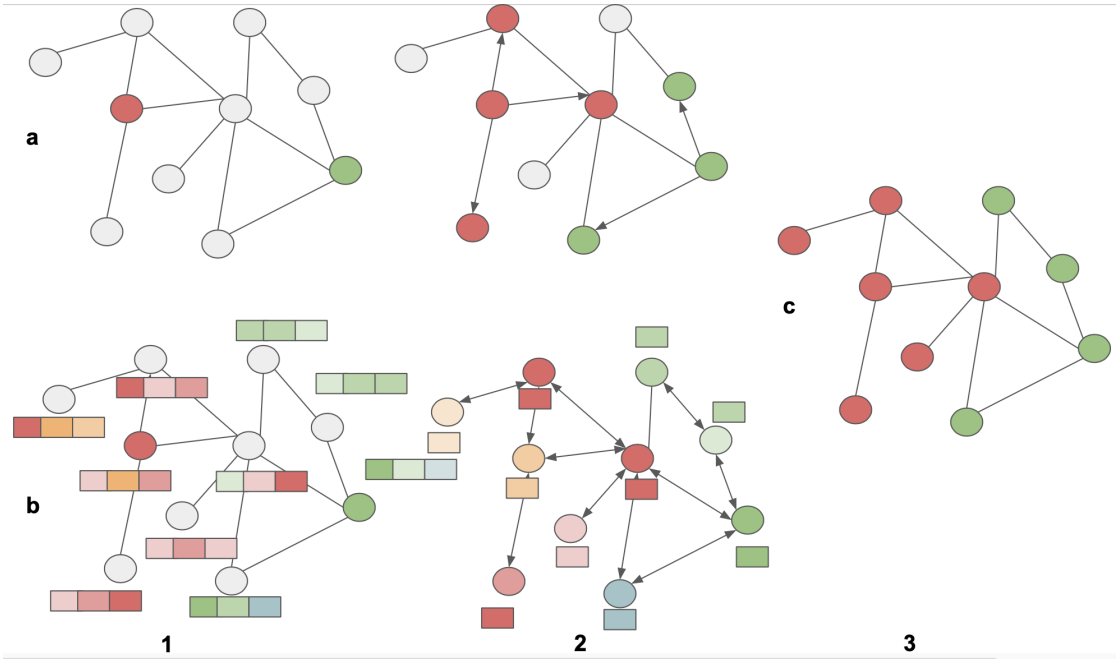

Figure 1.1: Comparison of **(a)** diffusion-based idea of label spreading through the graph versus **(b)** convolution of the graph structure with node features (boxes denote the features); **1** is an initial step; **(a) 2** is a spreading of label to neighbour nodes, **(b) 2** is a hidden dimension representation of features convoluted with graph structure; **(c) 3** is a final prediction.

citations are edges between articles with labels being the topics of articles; Classification of medical studies, where collecting labelled nodes is an expensive procedure [15]; Predicting the damaged equipment in the factory [16]. This case is highly demanded since the damaged equipment is a rare incident and collecting many of them is dangerous and expensive for production manufactory; Moreover, GB-SSL is helpful in post labelling in social networks [14, 17, 18] since it allows to make an automatic post labelling, relying on a small number of labelled posts. In other words, GB-SSL avoids expenses on crowdsourcing [19] and collecting a high number of labelled posts for supervised learning; Furthermore, GB-SSL is useful in detecting protein functions in different biological protein-protein interactions [20].

1.2 Overview of GB-SSL algorithms

An overview of the current, rapidly expanding GB-SSL research directions is provided in this section. We provide these directions as a composite of various research sub-directions,

each of which has unique research and practical characteristics. We specifically highlight the key concept for each sub-direction and list the most recent, top algorithms.

The structure of this section is as follows: the first part explains the idea of **Classical diffusion-based algorithms** as an original fundamental direction of GB-SSL. Note that classical diffusion-based algorithms are scalable for large graphs; the next part defines **Graph convolution networks** as a direction of GB-SSL, which combines the linear Classical diffusion-based algorithms with convolution neural networks for increasing accuracy; the last part describes the recent high demand direction of GB-SSL named **Scaling algorithms**.

Also, note that the aforementioned GB-SSL directions have the following general limitations: (i) **Classical diffusion-based algorithms** consume much less computational and memory complexity during training than Scaling algorithms. However, they lose to both Scaling algorithms and Graph convolution networks in classification accuracy in practice; (ii) **Graph convolution networks** are limited to small, sparse graph applications, because Out-Of-Memory (OOM) issues arise on large graphs; (iii) **Scaling algorithms** have high computational complexity, and in worst[1] cases, some of them face an OOM issue. Moreover, scaling algorithms do not guarantee an high accuracy close to graph convolution networks.

Due to the aforementioned limitations, the primary goal of this thesis is to develop scaling algorithms that will prevent critical OOM issues and significantly lessen memory and computational cost in comparison to current scaling techniques. We pay special attention to reducing memory and computational complexity from the point of allowing training on the low computational power of extremely large graphs. In addition, we develop scaling algorithms in this thesis that, when compared to existing scaling algorithms and graph convolution networks, might provide competitive classification accuracy in real-world applications. Additionally, we propose a novel diffusion-based and graph convolution frameworks that can improve classification accuracy in real-world problems. It should be noted that the goals of this thesis and the specific descriptions of the limitations mentioned above are presented in the following Section 1.3.

1.2.1 Classical diffusion-based algorithms

The main idea of classical diffusion-based algorithms is to recover the classes of node by spreading the information about labelled nodes through the graph to label unlabeled nodes. Classical diffusion-based algorithms have the following two sub-directions:

Linear transformation. This sub-direction leads to spreading the node classes from labelled nodes to unlabelled ones by minimizing Standard Laplacian loss as in [21], the

[1]The critical case described is in Section 1.3

Normalized Laplacian loss as in Label Propagation (LP) [3], the generalization Personalized PageRank as in PageRank SSL (PRSSL) [4] and distributed stochastic approximation for semi-supervised learning (DSA-SSL) [2], the novel graph regularization algorithm based on random matrix theory as in Semi-Supervised Learning for Large Dimensional Data (SSL-LDD) [22]. Note that the SSL-LDD [22] solves the specific *Curse of dimensionality* issue in Laplacian regularization loss by replacing of the standard Laplacian regularization by centered similarity matrix. However, SSL-LDD shows perceptible results mostly in the case of binary clustering and classification of non-graph data.

Nonlinear transformation. This sub-direction considers nonlinear transformation of node features to spread them through the sample of their neighbour nodes further and minimize Laplacian regularization loss for this transformation. This sub-direction contains the following principal algorithms, such as manifold regularization (ManiReg) [5], Planetoid [23], DeepWalk [24] or EmbedNN [25].

1.2.2 Graph convolution networks

Graph convolution networks apply the convolution property of the graph structure with node features for the classification of unlabeled nodes. More precisely, graph convolution networks compute the dot product of the adjacency matrix with the nonlinear transformation of node features for the classification. The principal graph convolution networks are divided into the classical nonlinear, generative and deep graph convolution networks, which are presented below:

Classical nonlinear graph convolution networks. This sub-direction convolves the graph adjacency matrix with nonlinear transformed node features for node classification. The main idea was proposed in the Graph Convolution Network (GCN) [6]. The latest works are: Graph attention network (GAT) [7], Gated attention network (GAAN) [26], GraphStar [27] propose combining GCN with attention mechanism [28]. The attention mechanism in graphs data expresses the important indicator of neighbour node features for a labelled node during training. Another novel algorithm, denoted as approximated Personalized graph neural network (APPNP) [14]. APPNP generalizes and improves the performance of the GCN by growing its complexity, including repeating PowerIteration [29] steps from PRSSL[2] algorithm during training. Moreover, note that there exist different nonlinear modifications of APPNP in the literature, among which are [6, 30–32].

Generative graph convolution networks. This sub-direction combines the nonlinear graph convolution networks with the generative model such as Variational Autoencoder (VAE) [33]. In particular, the latent representation of nodes features/edges from the

[2]Classical diffusion-based algorithm

generative VAE model is applied for enrichment of the graph convolution network. Examples of these models are: Generative PageRank (GenPR) [18], Graphite [31], Graph stochastic neural network (GSNN) [34], Bayesian Graph Neural network (BGCN) [35], the latent space model or stochastic block for GCN [36].

Deep graph convolution networks. This sub-direction proposes various architectures of neural networks for enhancing the GCN performance based on increasing the model depth. The principal deep graph convolution networks are: AS-GCN [37], DeepGCN [38], JK-net [39], MixHop [40], Snow-ball [41], DAGNN [42], GCNII [43], DropEdge [44] and Bayesian-GDC [45]. Note that the above algorithms propose adaptive residual learning framework [46, 47] from computer vision to graph convolution networks. Indeed, residual learning simplifies the network computation across a high number of layers. Furthermore, some of the aforementioned algorithms customize the dropout regularization technique [48] concerning graphs data.

1.2.3 Scaling algorithms

Scaling algorithms is a recent fast-growing high demand research direction for GB-SSL designed to overcome Out-Of-Memory (OOM) issues in the mentioned **Graph convolution networks** and provide in practice the high classification accuracy against **Classical diffusion-based algorithms**. In particular, most scaling algorithms focus on scaling **Graph convolution networks** on extremely large graphs such as Amazon ($2M$ nodes, classifying products by category) and Mag-coarse ($10M$ nodes, classifying papers by field of study). However, note that some scaling algorithms [49, 50] avoid the OOM issues on GPU by high resource power on RAM, making them vulnerable to OOM issues on RAM during training on small computers. The principal sub-directions of scaling algorithms are present below:

Nodes-neighbours selection. There are two main ideas for selecting the nodes-neighbours: i) determine the best neighbours to represent the labelled nodes during the training. PPRGO [51] and PinSage [52] apply an importance score to each neighbour node of the labelled node; ii) uniformly sample a fixed number of neighbours for the labelled nodes in a batch per training iteration as in Graph-S [53] and VR-GCN [54].

Subgraph-sampling. This sub-direction focuses on developing various subgraph sampling algorithms to guarantee the connectivity of nodes in subgraphs for further enhancement of neural network batch training. The principal algorithms in Subgraph-sampling are: Shadow-GNN [55], Graph-Saint [56], Cluster-GCN [57]. In particular, the latest best Shadow-GNN [55] outperforms the Graph-Saint and Cluster-GCN regarding classification accuracy and computational complexity. The Shadow-GNN shows profitable

results since it relies on shallow sampling subgraphs with 2-3 neighbour nodes for labelled nodes. However, Shadow-GNN requires more neural network layers for better node representation than the number of nodes in the subgraph, which lengthens training time. Additionally, Shadow-GNN could lose classification accuracy in contrast to graph convolution networks like APPNP and GCN since it approximates the GCN on a whole graph with an error. It should be noted that all of the approaches mentioned above create batch ensembling methods to broaden the representation for labelled nodes.

Model simplification. Model simplification algorithms make the node features propagate through the graph at the first step and then apply the multi-layer perceptron for training on the batch of updated node features. The latest best algorithms for model simplification are: Simple graph convolution (SGC) [58], Approximate Graph Propagation (AGP) [49], Graph neural network via Bidirectional Propagation (GBP) [50] and Graph Diffusion Convolution (GDC) [59]. More precisely, the aforementioned algorithms solve the Personalized PageRank problem for computing a feature propagation matrix and further use it for training a multi-layer perceptron in a batch regime.

Layer sampling. The neural network's layer sampling algorithms employ various sampling techniques of neighbour nodes from the previous layer. To avoid doing additional weight matrix computations for each node in the training batch, this concept takes advantage of historical activations of the previous layer. In particular, different Layer sampling algorithms modify this idea in various ways: for example, GraphSAGE [53] makes uniform node sampling from the previous layer neighbours; another work proposed S-GCN [54] algorithm, which limits the number of neighbouring nodes by demanding only two boost nodes from the last layer; the studies FastGCN [60] LADIES [61] propose to make node sampling independently for each layer; the latest GNNAutoScale (GAS) [62] algorithm combines node sampling from the previous layer on GPU with nodes from the RAM from the last training step. In other words, GAS utilizes all neighbours for labelled nodes during training.

1.3 Problems and Goals

This section explains the common critical and non-critical problems of GB-SSL algorithms and is structured as follows: the first sub-section presents the substantial critical computational and memory limitations that arise in each GB-SSL direction; the next sub-section outlines the specific problems that arise in the classical diffusion-based algorithms and the common non-critical problems that exist in most of the state-of-the-art GB-SSL algorithms; the last sub-section summarizes the main goals of this work.

1.3.1 Computational and memory critical limitations

Most GB-SSL algorithms, such as [2, 4, 23, 25], focus on solving the following Personalized PageRank (PPR) linear system:

$$(I - \alpha\widetilde{A})Z = (1-\alpha)Y, \tag{1.1}$$

where $\widetilde{A} = D^{-\delta}AD^{\delta-1} = [\widetilde{A}_{i,j}]_{i,j=1}^{n,n} \in \mathbb{R}^{n\times n}$ is a regularized adjacency matrix, $D = \text{diag}(D_{i,i})_{i=1}^{n}$ is a degree matrix with $D_{i,i} = \sum_{j=1}^{n} A_{i,j}$, $A = [A_{i,j}]_{i,j=1}^{n,n}$ is an adjacency matrix, $Y = [Y_i]_{i=1}^{n}$ is a matrix that represents labels, α is a regularization parameter and $Z = [Z_i]_{i=1}^{n} \in \mathbb{R}^{n\times c}$ is a classification result.

Classical diffusion-based algorithms. While the direct computation of the solution of the System (1.1) results in a $\mathcal{O}(n^3)$ computational complexity in [4], proposes the classical diffusion-based algorithm named as PageRank Semi-Supervised learning (PRSSL). PRSSL solves System (1.1) by application of the PowerIteration method which reduced complexity to $\mathcal{O}(n^2)$ or even to a smaller complexity in sparse graphs:

$$Z^{t+1} = \alpha\widetilde{A}Z^t + (1-\alpha)Y \tag{1.2}$$

where $Z^t \in \mathbb{R}^{n\times c}$ is a classification result at iteration t, $t \in [0, \ldots, \tau-2]$ with τ, the number of iterations is typically much less than n. Also, in [4], it is proved that PRSSL (1.3.1) converges to the explicit solution Z^* of (1.1). However, PRSSL has a high computational and memory complexity per iteration on large dense graphs. Furthermore, since PRSSL does not utilize node features during training, it loses to graph convolution networks in terms of classification accuracy.

The other known approaches try to reduce even further this computational complexity of System (1.1), e.g. by means of sampling schemes or updating nodes by batch (see [63] for a comprehensive survey), among which we can cite: Jacobian over relaxation (JOR) [64], Randomized Kaczmarz (RK) [1], Doubly Stochastic Block Gauss-Seidel (DSBGS) [65], Randomized Block Gauss-Seidel (RBGS) [66] algorithms. At each iteration, these algorithms update either the current node or a batch of nodes by their neighbors. Alternatively, the distributed stochastic approximation approach for semi-supervised learning (DSA-SSL) proposed in [2] consists in updating the current node from another single neighbor node at each iteration. This allows one to solve the memory and computational complexity issues of PowerIteration method (1.3.1), but results in a slow convergence rate to the optimal classification result.

In general, the linear classical diffusion-based algorithms can be very efficient from the point of memory consumption on large sparse graphs. However, their classification

accuracy is limited in practice mainly because System (1.1) does not account for the node features, only the adjacency matrix. Moreover, some of them require high computational complexity (e.g. PRSSL $\mathcal{O}(n^2)$, Randomized Kaczmarz $\mathcal{O}(n)$ per iteration), which can be critical at dense large graphs. At the same time, the nonlinear classical diffusion-based algorithms have a higher classification accuracy than linear ones since they embed nonlinear node features while minimizing Laplacian regularization loss. However, they consume much more memory and training time for computing Laplacian regularization loss over all neighbours of the labelled node.

Graph convolution networks. In order to improve the nodes' classification accuracy in practice and take into account the input features without minimization of Laplacian regularization loss for reducing training time, graph convolution networks were developed, such as APPNP [14]. They consist in nesting the recurrent equation (1.3.1) inside a neural networks as follows:

$$\begin{aligned} Z^0 &= ReLU(XW^t_{(0)})W^t_{(1)} \\ Z^{\tau+1} &= \alpha\widetilde{A}Z^{\tau} + (1-\alpha)Z^0 \\ Z^{\tau} &= softmax(\alpha\widetilde{A}Z^{\tau-1} + (1-\alpha)Z^0) \end{aligned} \tag{1.3}$$

where $ReLU(\cdot)$ is an activation function and $W^t_{(0)} \in \mathbb{R}^{d\times m}$ and $W^t_{(1)} \in \mathbb{R}^{m\times c}$ are trainable dense weight matrices at time t for converting input node features from d-space into hidden m-space and from hidden m-space into classes c-space, τ is the number of PowerIteration steps. The PowerIteration method in (2.8) for updating the nodes by combining hidden representations of their neighbours leads to increased computational complexity and critical OOM issues on GPUs on large graphs. In particular, computing the hidden representation for a current node features demands combining the hidden representation of its neighbours, and the neighbours, for its part, have to consider the hidden information of their neighbours, and so forth (see Figure 1.2). The above process makes costly neighbourhood growth, which grows exponentially with each extra layer. For that reason, many proposed graph convolution networks, such as [6, 18, 30, 31] and DeepGCN [38], JK-net [39], MixHop [40], DAGNN [42], GCNII [43], DropEdge [44], focus on small, sparse adjacency matrices with restrictions on the exponential growth of neighbour nodes. Note that this behaviour also leads to OOM issues on GPU, as in GCN, APPNP and other latest algorithms (AS-GCN [37], DeepGCN [38], JK-net [39], MixHop [40], DAGNN [42], GCNII [43], DropEdge [44] and Bayesian-GDC [45]).

Scaling algorithms. Moreover, we emphasize that the different types of scaling algorithms developed to scale graph convolution networks have their memory and computational bottlenecks:

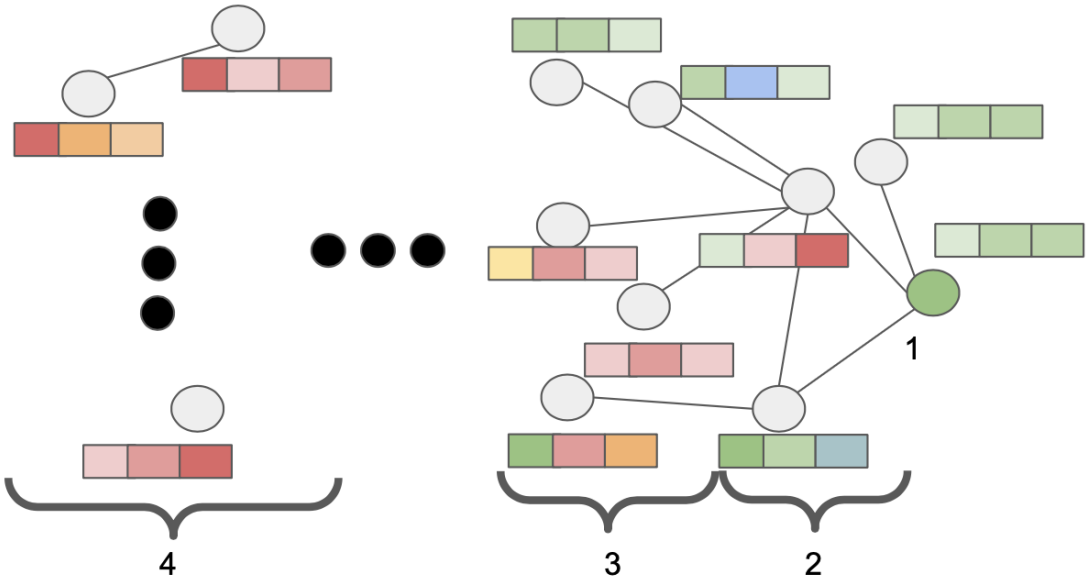

Figure 1.2: Computation step of Graph Convolution Network (boxes denote the hidden representation of the node features): **1** is a current node; **2** Three neighbour nodes to the current node; **3** Five neighbour nodes to the previous set of neighbour nodes; **4** The last n neighbour nodes to the previous set of neighbour nodes.

For example, the latest best **model simplification** algorithms, such as AGP [49], and GBP [50], resolve OOM issues at GPU for graph convolution networks and outperform the SGC [67] in memory and computational complexity by avoiding the use of PI during training. However, they do not guarantee the convergence to the exact solution of System (1.1). Moreover, they update one-node features by their neighbours, increasing the memory and computational complexity on large dense graphs with large feature d-space. The second bottleneck leads to OOM issues at RAM when d and n are extremely large. In particular, the OOM issues arise in the case when the number of neighbour nodes is close to n, then for update requires keeping the dense feature matrix X in RAM memory. However, these algorithms avoid this issue by high computational power (e.g. for AGP [49], use 512 GB RAM).

In case of **layer sampling** FastGCN, LADIES have a high computation complexity during training. In particular, the training complexity depends linearly on the number of node samples for each layer. In practice, during the training, the above algorithms mostly require much more nodes than the average node degree in the graph. The latest GNNAutoScale (GAS) algorithm utilizes a small number of neighbour nodes for labelled nodes on GPU (without sampling). The rest of the neighbours it extracts from RAM

and keeps the embeddings from the previous training step. This strategy of separating the memory consumption resolves the issue with computation complexity, however it leads to the issues with increasing memory on RAM. The worst-case scenario of this issue, which might cause OOM on RAM, is when the network has a deep architecture with many hidden dimensions and the number of neighbour nodes is near to the number of nodes in the graph.

Also, note that the **subgraph-sampling** algorithms such as: Cluster-GCN has an OOM issue on GPU in the worst case when the number of clusters is equal to one, and the behaviour of this algorithm starts to be equivalent to the behaviour of GCN. Cluster-GCN defines the cluster as a group of connected nodes in a graph that can be used like a batch during the training process; The Graph-Saint requires keeping in memory all neighbours for labelled nodes during the sampling (per iteration of training), which can increase the computational complexity and face the OOM on GPU in the same worst-case as for GAS.

Furthermore, note that **graph convolution networks** as well as **scaling algorithms** such as **model simplification** and **nodes-neighbour selection** have computational complexity issues because most of them [2, 4, 6, 14, 17, 31, 49–51, 67] rely on the computation of Personalized PageRank (PPR) [68] by PowerIteration [69] method or Jacobian over relaxation (JOR) [64] like methods. In particular, PowerIteration updates one node by the information from its neighbors and these neighbors are updated by the information from their neighbors, etc. This leads to an implicit increase of the computational complexity at: **the preprocessing/training step** when a node is updated by combining features/latent representations of its neighbors as in approximate personalized propagation of neural predictions (APPNP) [14], graph convolutional network (GCN) [6], GBP [50], SGC [67], AGP [49]; **the inference step** when the node is updated by classification results from its neighbors as in PPRGO [51] or Graphite [31].

Critical example. We want to draw attention to the critical computational problems that might prevent different GB-SSL algorithms from successfully training on a sizable dataset on a single standard computer. For that reason we consider MAG-coarse dataset [51] which contains $n = 12$M nodes and $d = 7$M of feature. Moreover, we used GPU GeForce 1070 (8 GB) with 32 GB RAM for computation. Note that we do not take into account the linear and nonlinear classical diffusion-based algorithms since, in fact, they lose in practice to graph convolution networks and scaling techniques in terms of classification accuracy. Below we explain issues that arise in graph convolution networks and scaling algorithms on the MAG-coarse dataset:

- **Graph Convolution Networks** (GCN, APPNP, GAT, etc.) have OOM on GPU since they need to keep in GPU memory the dense matrix $W^t_{(0)} \in \mathcal{R}^{7M \times m}$ with

$m = 32$ [3] and compute the dot products of $\widetilde{A}(XW^t_{(0)})$ where $X \in \widetilde{R}^{12M \times 7M}$ and $\widetilde{A} \in \widetilde{R}^{12M \times 12M}$ are sparse. This dot product is crucial because it must combine the hidden representations of a current node's neighbors in order to compute the hidden representation for that node's characteristics. The neighbors, in turn, must take into account the hidden information of their neighbors, and so on;

- **Scaling algorithms**: *Model simplification (AGP, SGC, GBP)* have a OOM issue on RAM since these algorithms make transformation of node features matrix by PowerIteration method as in $\widetilde{X}^{t+1} = \alpha\widetilde{A}\widetilde{X}^t + (1-\alpha)X$, which means that their require to keep in memory the dense result $\widetilde{X}^t \in \mathcal{R}^{12M \times 7M}$; *Subgraph-sampling* (Graph-Saint, Shadow-GCN) have a high computational complexity ($3-8$ hours per epoch) due to the fact that they are required to make subgraph sampling at each iteration. Also, Graph-Saint has an OOM on GPU. In particular, during training, it stuck on the batch with labelled nodes, which requires a subgraph with their neighbour nodes with the number close to 12 M; *Layer-sampling* (GAS) has an OOM on RAM because during training it requires to keep in memory and to update across epochs the dense matrices with nodes embedding (e.g. dense embedding matrices $\widetilde{X}^{n \times m}_{(0)}$, $\widetilde{X}^{n \times c}_{(1)}$ or dense trainable weight matrices $W^{d \times m}_{(0)}$, $W^{m \times c}_{(1)}$ for 0 and 1 layers respectively). In particular, this dense matrices required to keep in memory in case if we want to update labelled nodes by their neighbours and the number of neighbour nodes is close to n; *Nodes-neighbour sampling* (PPRGO) has an $\mathcal{O}(n^2)$ computational complexity at inference since it uses the PowerIteration method.

Finally, based on the aforementioned issues and bottlenecks in the existing critical example, we can emphasise the following general critical problems, which remain unresolved for all GB-SSL directions:

1. **Out-of-memory (OOM) RAM/GPU issues**:OOM issues occur because graph convolution networks spread the information through the whole graph, which requires keeping in memory a complete graph and information about embeddings [6, 14] from neural network layers for all nodes. Even more, it could arises in the worst cases of scaling algorithms (e.g. AGP, Graph-Saint, GAS);

2. **High computational complexity**: Most of scaling algorithms resolve OOM issues on GPU but require a high computational complexity [49, 56, 57] and do not provide accuracy close to graph convolution networks [51]. The high computational complexity in most of the scaling algorithms is connected with the complicated nodes sampling

[3] is an optimal value defined in [51]

strategies for batches during the training (e.g. Graph-Saint, GAS) or application of JOR-like methods for computation PageRank at preprocessing/inference step (e.g. AGP, PPRGO). Moreover, linear classical diffusion-based algorithms consume much less memory during the training than scaling algorithms. However, most of them require high computational complexity (e.g. PRSSL $\mathcal{O}(n^2)$, Randomized Kaczmarz $\mathcal{O}(n)$ per iteration).

1.3.2 Specific Laplacian regularization and general non-critical GB-SSL problems

This section highlights the specific problems associated with adjustments of Laplacian regularization loss in classical diffusion-based algorithms and general non-critical issues present for all GB-SSL algorithms. Note that solutions to these problems are valuable from the practical point of view since they mainly impact classification accuracy, explainability of classification results and versatility of applications on real datasets. Moreover, these problems do not block the training process, opposite to computational/memory limitations issues.

Most of the works in classical diffusion-based algorithms focus on **the linear transformation** of Laplacian regularization loss [4, 23, 25] and consider the following minimization loss function:

$$\min_{Z\in\mathbb{R}^{n\times k}}\Big\{\sum_{i=1}^{n}\sum_{j=1}^{n}A_{i,j}||Z_i-Z_j||_2^2+\mu\sum_{i=1}^{n}||Z_i-Y_i||_2^2\Big\}, \tag{1.4}$$

where μ is a Lagrangian multiplier, n is the number of nodes, $A=[A_{i,j}]_{i,j=1}^{n,n}$ is an adjacency matrix, $Z=[Z_i]_{i=1}^{n}$ is a classification result and $Y=[Y_i]_{i=1}^{n}$ is a matrix that represents labels. The first part of the objective in (1.4) is a Laplacian regularization, which penalizes nodes connected from different classes, while the second part is a supervised classification loss.

Other [5, 25, 70, 71] works focus on **the nonlinear transformation** of Laplacian regularization loss and propose to take into account the node features and consider different modifications of the nonlinear Laplacian regularization loss:

$$\begin{aligned}\mathcal{L}_{lap}(x,y)&=\mathcal{L}(f(x),y)+\mu\mathcal{L}_{reg}(X,A)\\&=\mathcal{L}(f(x),y)+\mu\sum_{i,j}A_{i,j}||\psi(x_i)-\psi(x_j)||^2,\end{aligned} \tag{1.5}$$

where the first part of the above equation $\mathcal{L}(f(x),y)$ is a supervised loss for the labelled part of the graph and the second part $\mathcal{L}_{reg}(X,A)$ is the extension of Laplacian regularisation

loss. $\psi(\cdot)$ is some nonlinear function from the neural network layer. This type of loss works in batch mode and during training can take into account the information about the graph as well as about the nodes' features.

We should emphasize that for the presented above modifications of Laplacian regularization loss (1.4), (1.5) exist the following specific problems:

1. **Binary edges** ($A_{i,j} = 0$ or $A_{i,j} = 1$) are a poor reflection of node similarity which can lead to a weak estimation of the Laplacian regularization. For instance, $A_{i,j} = 1$ does not provide the information about impact of cited paper j on the citing one i; Even more, $A_{i,j} = 0$ may show that author i did not cite the paper j, but he could have used some information from it. The special case of this issue arises in the case of utilization of labelled dangling nodes for training. In particular, the labelled dangling nodes will make a feeble impact during the training, which leads to losing performance in terms of accuracy. Also, we underline that this issue vanishes in graphs where nodes are connected only inside their classes and do not have cross edges between different classes. Even more, we would like to note that binary edges could negatively impact classification accuracy in practice even on **graph convolution networks** and **scaling algorithms**, which do not use the latent representation of edges;

2. **Curse of dimensionality**- arises when A is replaced by a similarity matrix $W = [h(X_i, X_j)]_{i,j=1}^n \in \mathbb{R}^{n\times n}$ with a positive definite kernel $h(\cdot)$ and $d \to \infty$ where $X = [X_i]_{i=1}^n$ is a matrix of node features and d is the node features dimension. This replacement presents in [4, 22] and it is made to avoid the sparsity of A. This issue is especially noticeable in the case of paper classification, for example, based on the Heaps law [72], the d-space of features (bag-of-words [73]) is increasing with respect to the number and length of papers. This issue is critical for the linear transformation (1.4).

Finally, we want to highlight the general non-critical problems that arise in most state-of-the-art GB-SSL algorithms:

1. **Versatility limitations -** the loss functions (1.4), (1.5) as well as graph convolution networks (e.g. APPNP (2.8)) and scaling algorithms do not support the semi-supervised learning on the universal type of datasets since they require the information about graph structure [6, 12, 23, 74]. This general limitation underlines that the GB-SSL algorithms do not maintain competitive performance in not graph-based areas. On the opposite side, the non-graph based SSL algorithms

such as TSVM and VAESSL [12] do not guarantee the competitive performance on graph-based data;

2. **Explainability -** is the last and more general problem that is critical nowadays in the machine learning domain. We underline this problem in our work so far as the GB-SSL algorithms are in demand in such natural fields as medicine and factories [4], where the transparency of predictions is also one of the essential requirements.

1.3.3 Goals

Based on the descriptions of problems mentioned in the above subsections, we want to emphasize the necessity of looking for a solution for Computational and Memory limitations in GB-SSL. This limitation is more critical than the other problems because it can block the application of algorithms on large datasets at the beginning, which leads to skipping the solutions for the rest of the problems. Moreover, Computational and Memory limitations are critical to the equipment required for training. In particular, the computation on extremely large graphs increases energy training costs (and thus the carbon footprint) due to a long time continuous usage of local computing resources with high powerful GPU and RAM and the additional cost of transmitting data over long-distance links in case of training the models in web-cloud or on a distributed cluster. Energy considerations for GB-SSL have been investigated in a few recent papers in different setups [75–77].

In this regard, our work aims to resolve the common critical GB-SSL algorithms issues with computational and memory limitations and also propose solutions for the rest of the common non-critical problems in GB-SSL. This order of goals allows us to propose: at first, the algorithm for significantly reducing computational and memory complexity in comparison with the latest scaling algorithms, and then, next, the frameworks for resolving specific and common non-critical GB-SSL problems, which can be scaled by algorithm from the first step.

1.4 Contributions

This section explains our work's main contributions and focuses on the goals defined in the previous section. The structure of this section is as follows: (i) The first sub-section defines the principal contribution of this work **Markov-Batch Stochastic Approximation (MBSA) algorithm** which focuses on the solution of **computational and memory critical**

[4] GB-SSL in evidence-based medicine and prediction of broken equipment on the factory.

limitations; (ii) The following two sub-sections define the solutions to the specific **Laplacian regularization problem** and general non-critical problems from the viewpoints of classical diffusion-based algorithms and graph convolution networks, respectively; (iii) The last sub-section shows how the novel diffusion-based algorithm suggested in this work may be used to handle practical issues like identifying the unbalanced states of engines in a manufacturing facility.

1.4.1 Markov-Batch Stochastic Approximation algorithm

We propose a novel **Markov-Batch Stochastic Approximation** (MBSA) algorithm for graph-based semi-supervised learning based on the stochastic approximation theory. We provide a theoretical analysis of MBSA with proof of stability and convergence to the desired Personalized PageRank (PPR) solution. We also show that MBSA can be used in the asynchronous parallel regime. In addition, we provide a multi-threading implementation on C++ for MBSA. We show on various datasets that MBSA outperforms the linear Jacobian over relaxation (JOR) [64], Randomized Kaczmarz (RK) [1], Doubly Stochastic Block Gauss-Seidel (DSBGS) [65], Randomized Block Gauss-Seidel (RBGS) [66] algorithms, reaching higher performance in a shorter computational time. Moreover, we show an insight that the fast recovery of the PPR order goes to the best accuracy faster than searching for an exact PPR solution. We adapted MBSA for batch training of graph convolution networks (e.g. APPNP, GCN) and named it MBSA-NN. Furthermore, we show the theoretical results for MBSA-NN on its convergence to a local minimum of graph convolution network on a complete graph. We applied MBSA-NN on several extremely large datasets and show that it can handle graphs with more than $n \approx 10M$[5] nodes and $d \approx 2M$ of node features under **one** minute on one standard machine, including preprocessing, training and inference time. Furthermore, we demonstrate that it greatly reduces memory and time consumption and offers competitive accuracy performance compared to the most recent top scaling algorithms. In particular, it consumes **10** times fewer nodes per batch at training than PPRGO and reduces memory consumption by **50**% at the inference against PPRGO. Finally, we provide an open-access implementation of MBSA-NN on Python3.8 with Tensorflow v2, one thread MBSA on Python3.8 and multi-thread MBSA on C++.

[5] M stands for million.

1.4.2 Graph diffusion & PCA

We propose a novel diffusion-based algorithm named as **Graph diffusion** & **PCA** (GDPCA) framework aiming at solving the *Curse of dimensionality*, *Binary edges* and *Versatility limitations* issues. The main idea of GDPCA relies on joint minimization of a reorganized principal component analysis (PCA) loss and linear transformation of Laplacian regularization loss (1.4). Also, we provide a theoretical analysis of GDPCA performance with proof that GDPCA provides an explicit solution to the proposed minimization loss problem. We apply it to real datasets and show that GDPCA is the best among classical diffusion-based state-of-the-art algorithms and has comparable performance with graph convolution networks with significantly lower computational complexity. Moreover, we show that GDPCA can also be applied to datasets with no explicit graph structure, such as images, and that it outperforms classical diffusion-based algorithms and graph convolution network algorithms on this dataset. Furthermore, we show that GDPCA can be scalable by MBSA for providing low computational complexity. Finally, we provide the implementation of GDPCA on Python3.8 in open access.

1.4.3 Generative PageRank

We propose a novel graph convolution inductive/ transductive framework, created by embedding PageRank-SSL (PRSSL) [4] in generative model named as **Generative PageRank** (GenPR). We show that the generative model can be used to reweight edges in the adjacency matrix to improve the nodes classification and solve *Curse of dimensionality*, *Binary edges* and *Versatility limitations* problems. Also, we explain that training in the transductive or inductive regimes helps to manage the computational complexity by separating training of nodes' low dimensional representation against nodes classification. Moreover, we show that **GenPR** improves the interpretability of neural network classification results based on the information about nodes' similarity in the latent space. In particular, it helps to resolve *Explainability* issue. GenPR provides results that outperform the recently proposed graph convolution networks and reduces the number of steps of PageRank [68] to obtain more accurate classification accuracy than APPNP.

1.4.4 GDPCA and Zoetrope Genetic Programming for detecting imbalanced states of engines

We propose a flexible SSL framework based on the stacking of PageRank & PCA (GDPCA) (enabling self-labelling [78]) and Zoetrope Genetic Programming (ZGP) [79] named as PaZoe. Note that we adapt PaZoe framework to sensor data. This self-labelling framework

shows that graph-based (e.g. GDPCA) and non-graph based (e.g ZGP) algorithms jointly improve the quality of predictions and outperform each component taken alone. We also show that PaZoe outperforms state-of-the-art GB-SSL and non-GB-SSL algorithms on three time-series datasets where two of which are public domain gesture datasets and the third one we generated from scratch based on a DC motor[6] for the classification of the type of motor imbalance at different rotation speeds. The third set was generated in house, taking data from industrial graded equipment to mimic DC motors during operation. Two other datasets, including gesture recordings, were taken from the public domain. The GDPCA part of PaZoe implemented on Python3.8 and generated dataset for DC motor are available in open access.

[6]GBM2804H with Nucleo G431RB ST L6230

Chapter 2

Related work

This chapter defines the latest state-of-the-art (SOTA) algorithms close to this work and covers GB-SSL and non-GB-SSL branches. We highlight the findings that served as our inspiration for this investigation. The sections below are organized in the following way: i) at the beginning, we present the latest SOTA algorithm for solving the Personalized PageRank (PPR) problem in a batchwise regime; ii) the next section shows the latest best SOTA scaling algorithms. In particular, this section presents the closest algorithms to our work with the latest result from node-neighbour selection and model simplification sub-directions of GB-SSL scaling algorithms; iii) finally, we show the main idea, which lies under the linear transformations in most classical diffusion-based algorithms. Moreover, we explain the classical graph convolution networks, which embed the graph structure directly to neural network architecture to avoid the computation of the Laplacian regularization loss. Also, we show the SOTA algorithm in the non-GB-SSL, which will be extended in our work for application in batchwise graph-based cases.

2.1 Batch algorithms for PPR

Since in the previous Chapter 1, we defined that the solution of Personalized PageRank (PPR) problem is an essential problem nowadays [2, 4, 23, 25]. In this section, we show the newest SOTA batchwise solutions for the following PPR linear system:

$$(I - \alpha\widetilde{A})Z = (1 - \alpha)Y, \tag{2.1}$$

where $\widetilde{A} = D^{-\delta}AD^{\delta-1} = [\widetilde{A}_{i,j}]_{i,j=1}^{n,n} \in \mathbb{R}^{n\times n}$ is a regularized adjacency matrix and $D = \text{diag}(D_{i,i})_{i=1}^{n}$ is a degree matrix with $D_{i,i} = \sum_{j=1}^{n} A_{i,j}$. In particular, in this section, we consider two latest batchwise Gauss-Seidel algorithms much similar to this work.

2.1.1 Doubly Stochastic Block Gauss-Seidel

We present the novel batchwise Gauss-Seidel algorithm with exponential learning rate named as Doubly Stochastic Block Gauss-Seidel (DSBGS). Let $\{I_1, I_2, ..., I_s\}$ and $\{J_1, J_2, ..., J_s\}$ denote the partition of rows and columns of adjacency matrix where s is the number of partitions and $\cup_{i=1}^{s} I_i = \mathcal{V}$, $I_i \cap I_j = \emptyset$. Let $P = \{I_1, I_2, ..., I_s\} \times \{J_1, J_2, ..., J_s\}$.

Algorithm 1: Doubly Stochastic Block Gauss-Seidel (DSBGS) [65]

INPUT : A, Y, P, α, δ, β

1 **INITIALIZE**: $\widetilde{A}$, $Z^0 = Y$;
2 **for** $t \leftarrow 0$ **to** τ **do**
3 Pick $(I, J) \sim P$ with probability $\frac{||\widetilde{A}_{I,J}||_F^2}{||\widetilde{A}||_F^2}$; where $\widetilde{A}_{I,J} \in \widetilde{A}$;
4 Update $Z^t = Z^{t-1} - \beta\frac{I_{:,J}\widetilde{A}_{I,J}^T I_{:,I}^T}{||\widetilde{A}_{I,J}||_F^2}(\alpha\widetilde{A}Z^{t-1} - (1 - \alpha)Y)$
5 **end**

This algorithm guarantees the exponential convergence rate to the exact solution of System (2.1). Moreover, DSBGS proposes batchwise updating strategie per iteration. However, it has the following main bottleneck: the batch I, which updates the batch J, utilizes the information from all of its neighbour nodes (see the line 4 in Algorithm 1). In the worst case, it can lead to increased time and memory consumption on extremely large graphs where the degree of nodes is close to the number of nodes.

2.1.2 Randomized Block Gauss-Seidel

Another latest batchwise Gauss-Seidel algorithm named Randomized Block Gauss-Seidel (RBGS) also has an exponential convergence rate. However, RBGS is much simpler

than DSBGS from the point of the partition selection since RBGS performs the uniform partition selection and computes partitioning only over columns. In other words, RGBS use only $\{J_1, J_2, ..., J_s\}$ partitions over columns of the adjacency matrix.

Algorithm 2: Randomized Block Gauss-Seidel (RBGS) [66]

INPUT : A, Y, α

1 **INITIALIZE**: $\widetilde{A}$, $Z^0 = Y$, $r^0 = (1-\alpha)Y - (I - \alpha\widetilde{A})Z^0$;
2 **for** $t \leftarrow 0$ **to** τ **do**
3 Pick J uniformly from $\{J_1, J_2, ..., J_s\}$;
4 Create block of $(I - \alpha\widetilde{A})_{[.,J]} \in (I - \alpha\widetilde{A})$;
5 Generate $E \in \mathcal{R}^{n \times bs}$ where bs is a size of partition and $\forall i \in \{1, ..., bs\}$ the i_{th} column of E is $E_{[.,i]}$, has all zeros with a 1 in the c_ith position, where c_i is the ith entry in the selected J;
6 $Z^t = Z^{t-1} + E(I - \alpha\widetilde{A})^{\dagger}_{[.,J]} r^{t-1}$, where $(I - \alpha\widetilde{A})^{\dagger}_{[.,J]}$ is the Moore-Penrose pseudoinverse of matrix $(I - \alpha\widetilde{A})_{[.,J]}$
7 $r^t = (1-\alpha)Y - (I - \alpha\widetilde{A})Z^t$
8 **end**

This algorithm defines the most straightforward uniform strategy of partition selection contrary to DSBGS. Also, it guarantees the exponential convergence rate to the exact solution of the System (2.1). Note that RBGS avoids keeping in memory the probability of partition intersections as made in DSBGS (see P in Algorithm 1). This option is helpful from the point of partitioning large graphs, where we can have a relatively big dense matrix of the probability of partition intersections. However, RBGS has the following bottlenecks similar to DSBGS: 1) it computes the Moore-Penrose pseudoinverse of the bloc matrix, increasing the time and memory complexity, especially on large graphs. (see the line 6 in Algorithm 2); 2) the updates in the line 6 and 7 in Algorithm 2 use the information from all of J partition neighbour nodes. These issues are as well as in DSBGS (Algorithm 1) can increasing time and memory consumption during computations on large graphs.

2.2 Scaling algorithms

This section defines the latest best state-of-the-art scaling algorithms close to our work. In detail, we describe the novel high performance sub-directions of scaling algorithms, such as model simplification and nodes-neighbour selection.

2.2.1 Model simplification

The model simplification focuses on separating the step of feature propagation against training multi-layer perceptron in the batch regime. In particular, the latest algorithms such as Approximate Graph Propagation (AGP) [49] and Graph neural network via Bidirectional Propagation (GBP) [50] solve the system (2.1) for updating the matrix of nodes features X for its further use in multi-layer perceptron. It means that in (2.1) instead of Y the above-mentioned algorithms use X where the updated matrix of nodes features is $Z \in \mathcal{R}^{n \times d}$. These algorithms have higher classification accuracy than mentioned above linear algorithms. Also, they resolve the OOM issues on GPU for graph convolution networks and outperform the SGC [67] algorithm in memory and computational complexity due to avoiding of PowerIterations method for feature spreading. However, they have the following bottlenecks: 1) do not guarantee the exact solution of System (2.1). It means that these algorithms do not provide optimal feature embeddings contrary to embeddings which the exact solution of System (2.1) can have. Notably, compared to the remainder sub-direction of scaling algorithms, this might demonstrate the weakest accuracy; 2) update one node by their neighbours which leads to increasing of the memory and computational complexity on large graphs with large d. The second bottleneck in the worst case leads to OOM issues at RAM in the case of extremely large graphs where there are nodes with the number of neighbours close to n and with a large number of features d. It should be noted that by keeping track of the decline in the number of residual neighbour nodes, Randomized AGP lowers the number of neighbours per node update. The large memory consumption peaks at the beginning of iterations are still present despite this, though. Since all nodes have high residuals at the beginning, the high peaks appear during the initial iterations, which increases computing complexity because of neighbour sampling.

2.2.2 Nodes-neighbours selection

The latest best representative of nodes-neighbours selection algorithms is PPRGO [51]. PPRGO focuses on applying an approximated Personalized PageRank (PPR) [80] for selecting the top k PPR neighbors for each labeled node and further use them in the neural network training process. PPRGO avoids the OOM issues on RAM because it does not apply updating node features as in GBP and AGP. However, PPRGO has the following bottlenecks: 1) in each batch, requires $bs \times k$ nodes, bs being the batch size, leading in the worst case (viz. dense adjacency matrix) to a total of **16384** nodes for the optimal values $bs = 512$ and $k = 32$ defined in their work [51]. Indeed, they show that the

performance is improved by using as many neighbors as possible. furthermore, since the top k PPR neighbors represent each labeled node, it reduces the variety of labeled node representation; 2) finally, at the training step, PPRGO is slower than AGP and GBP and loses to them in terms of accuracy. This issue arises because AGP and GBP support the training process on a fixed number of nodes in a batch, contrary to PPRGO, where the number of nodes in a batch depends on k; 3) finally, PPRGO applies the PowerIteration method at the inference step. This behaviour at inference increases the memory, and computational complexity on large graphs since node classification results are spread through a complete graph.

2.3 Classical graph-based & Non-Graph based SSL algorithms

We need to acknowledge that the frameworks proposed in Sections 4, 5 were inspired by the results of the Personalized PageRank application for SSL [4] and Variational Autoencoder (VAE) [33] extension for SSL [12]. Indeed, we considered the following algorithms as a good ground for solving the limitations with Laplacian regularization problems.

2.3.1 Classical diffusion-based algorithm

PageRank Semi-Supervised learning

One of the main components of the frameworks GDPCA (see Chapter 4) and GenPR (see Chapter 5) proposed in this work is the PageRank-based method for semi-supervised learning [4]. The work [4] minimizes the following function

$$\min_{Z}\Big\{\sum_{i=1}^{N}\sum_{j=1}^{N}A_{i,j}||D_{i,i}^{\sigma-1}Z_{i.}-D_{j,j}^{\sigma-1}F_{j.}||^2+\mu\sum_{i=1}^{N}D_{i,i}^{2\sigma-1}||Z_{i.}-Y_{i.}||^2\Big\} \tag{2.2}$$

with respect to the matrix of classification results Z. The first part of the minimization function in (2.2) is a Laplacian regularization, which penalizes nodes connected from different classes, while the second part is a supervised classification loss. The above optimization problem has an explicit solution proposed in [4]:

$$Z_{.k}=\frac{\mu}{2+\mu}(I-\frac{2}{2+\mu}\widetilde{A})^{-1}Y_{.k} \tag{2.3}$$

where $\widetilde{A}=D^{-\delta}AD^{\delta-1}=[\widetilde{A}_{i,j}]_{i,j=1}^{n,n}\in\mathbb{R}^{n\times n}$ is a regularized adjacency matrix and $D=\text{diag}(D_{i,i})_{i=1}^{n}$ is a degree matrix with $D_{i,i}=\sum_{j=1}^{n}A_{i,j}$. In particular, from equation (2.3)

we derive: the Standard Laplacian method if $\sigma = 1$; the PageRank method if $\sigma = 0$ and the Normalized Laplacian method if $\sigma = 1/2$. Let us denote $\alpha = \frac{\mu}{2+\mu}$ and rearrange the terms in (2.3) to obtain the power iteration [29] like algorithm for iterative calculation of the classification function:

$$Z^t_{.k} = \alpha \widetilde{A} Z^{t-1}_{.k} + (1-\alpha) Y_{.k}, \quad k = 1, 2, ..., \tag{2.4}$$

where $Z^t_{.k}$ is a result of the t-th iteration, smoothly changing the node labels during $t \geq 0$ iterations. This algorithm avoids the $\mathcal{O}(n^3)$ computational complexity of exact computation PageRank. Also, it can be applied to data with or without graph structure. Particularly, if a graph structure is not given to us, we can calculate the similarity matrix W from the feature space. For instance, W can be constructed using the Radial Basis Function (RBF):

$$W_{i,j} = \exp(-||X_i - X_j||^2/d). \tag{2.5}$$

However, PRSSL has the following issues: 1) since PRSSL is based on the PowerIteration method, then it takes $\mathcal{O}(n^2)$ computational complexity per iteration, which leads to increasing the computation time on the large dense graphs; 2) the case of binary edges in a graph leads to a weak estimation of the Laplacian regularization loss in PRSSL. For instance, this problem is significant when most of the available labelled nodes are dangling, making a feeble estimation of the Laplacian regularization loss; 3) the computation of node similarity in W (2.5) becomes indiscernible in the high-dimensional cases because the difference between a maximum (max_d) and a minimum (min_d) Euclidean distances goes to zero when the dimension increases [81], i.e.,

$$P\Big[\Big|\frac{max_d}{min_d} - 1\Big| < \varepsilon\Big] \to 1, \quad d \to \infty.$$

2.3.2 Graph convolution networks

Graph convolution network

The main idea proposed in Graph Convolution Network (GCN) [6] rely on the dot product of adjacency matrix A and the nonlinear transformation of features X for the classification. In particular, this dot product makes a representation of nodes through the sum of the nonlinearly transformed features of their neighbour nodes for further transformation of these representations into classification results. GCN proposes directly encoding the graph structure in neural network architecture and computing only supervised classification loss without the Laplacian regularization loss. The architecture of GCN presented below:

$$Z = softmax(ReLU(\widetilde{A}XW^t_{(0)})W^t_{(1)}) \tag{2.6}$$

where $ReLU(\cdot) = max(0, \cdot)$ and $softmax(Z_{i,j}) = \frac{\exp^{Z_{i,j}}}{\sum_{j=1}^{c} \exp^{Z_{i,j}}}$ are an activation functions and $W^t_{(0)} \in \mathbb{R}^{d\times m}$ and $W^t_{(1)} \in \mathbb{R}^{m\times c}$ are trainable weight matrices at time t for converting input node features form d-space into hidden m-space and from hidden m-space into classes c-space. This architecture is trained through minimization of the following function over available labelled nodes:

$$\min_{W_{(0)},W_{(1)}} \Big\{ \sum_{i=1}^{n_l} ||Z_i - Y_i||^2 \Big\} \tag{2.7}$$

where n_l is the number of labelled nodes. Note that the quadratic loss function in (2.7) can be replaced by any other classification loss function.

Approximated Personalized graph neural network

The generalized latest version of GCN is proposed in a novel algorithm named approximated Personalized graph neural network (APPNP) [14]. In particular, APPNP generalizes and improves the performance of the GCN by growing its complexity, including the repeating PowerIteration steps from PRSSL[1] algorithm during training. The architecture of APPNP consists in nesting the recurrent equation (2.4) inside a neural networks as follows:

$$\begin{aligned} Z^0 &= ReLU(XW^t_{(0)})W^t_{(1)} \\ Z^{\tau+1} &= \alpha\widetilde{A}Z^\tau + (1-\alpha)Z^0 \\ Z^\tau &= softmax(\alpha\widetilde{A}Z^{\tau-1} + (1-\alpha)Z^0) \end{aligned} \tag{2.8}$$

where τ is the number of PowerIteration steps. Note that training of APPNP rely on minimization the same loss function as in GCN (2.7). Since during the training, APPNP and GCN exclude the minimization of the Laplacian regularization, making their training process faster than in classical diffusion-based algorithms with nonlinear transformations such as Planetoid, DeepWalk or EmbedNN. However, APPNP and GCN compute the hidden representation for current node features by combining the hidden representation of its neighbours, and the neighbours, for their part, have to consider the hidden information of their neighbours, and so forth. In particular, this behaviour of combining hidden

[1]Classical diffusion-based algorithm

neighbour representations requires keeping in memory the dense matrix with the hidden representation of nodes $XW^t_{(0)}$ if the number of neighbours is close to n, which particularly leads to increased computational complexity and critical OOM issues on GPUs on large graphs.

2.3.3 Non-Graph based algorithm

VAE for semi-supervised learning

Another principal component of GenPR framework (see Chapter 5) with respect to PRSSL is variational autoencoder (VAE) for semi-supervised learning [12]. For the definition of VAE, we are using the following assumptions:

Assume that the set of points X are i.i.d. samples of variable x. It is also assumed that x is generated with respect to a latent continuous random variable z in two steps (e.g., see [33]):

1. a value for z_i is generated from some prior distribution $p_\theta(z)$;
2. a value for x_i is generated from conditional distribution $p_\theta(x|z)$.

Hence, we have the following generative model with parameters θ:

$$p_\theta(x, z) = p_\theta(z)p_\theta(x|z) \tag{2.9}$$

where the posterior density $q_\theta(z|x) = \frac{p_\theta(z)p_\theta(x|z)}{p_\theta(x)}$ is typically intractable.

However, under above assumption, we can apply the main idea of VAE [33] using a variational approximation posterior $q_\phi(z|x_i)$ to the true posterior $p_\theta(z|x_i)$ with parameters ϕ. This calculation is based on the minimization of the variational lower bound (ELBO), which consists of two parts:

1. Kullback-Leibler divergence (D_{KL}) between $q_\phi(z|x_i)$ and $p_\theta(z)$;
2. conditional expectation $\mathbb{E}_{q_\phi(z|x_i)}$ of $\log\, p_\theta(x|z)$ under condition of the approximation posterior $q_\phi(z|x_i)$.

$$\begin{aligned}\mathcal{U}(\theta, \phi, x_i) = &-D_{KL}(q_\phi(z|x_i)||p_\theta(z)) \\ &+ \mathbb{E}_{q_\phi(z|x_i)}[\log\, p_\theta(x_i|z)]\end{aligned}$$

Let us assume that the prior $p_\theta(z)$ is the isotropic multidimensional Gaussian distribution $p_\theta(z) = \mathcal{N}(z; 0, I)$ with the expectation equal to 0 and with the covariance matrix equal to I.

Moreover, let us apply the reparameterisation trick for the calculation of the approximation posterior $q_\phi(z|x_i)$. Then, the variational approximation $q_\phi(z|x_i)$ to the true posterior $p_\theta(z|x_i)$ becomes:

$$\begin{aligned} q_\phi(z|x_i) \sim z_{i,k}, \quad z_{i,k} = \mu(x_i) + \sigma(x_i) \odot \epsilon_k, \\ \epsilon_k \sim \mathcal{N}(0, \tau I) \\ \sigma(x_i) \odot \epsilon_k - \sigma(x_j) \odot \epsilon_k \underset{\tau \to 0}{\to} 0 \quad \forall\ x_i, x_j \in X \end{aligned}$$

where $\odot$ is an element-wise product and $\mu(x_i)$ and $\sigma(x_i)$ are the encoding results from MLP.

Then under the above assumption we obtain:

$$\begin{aligned} \mathcal{U}(\theta, \phi, x_i) = \frac{1}{2} \sum_{j=1}^{s} (1 + log((\sigma_j^2(x_i))) \\ - (\mu_j(x_i))^2 - (\sigma_j^2(x_i))) \\ + \frac{1}{K} \sum_{k=1}^{K} \log\ p_\theta(x_i|z_{i,k}) \end{aligned}$$

where s is the dimension of the latent variable z and K is the number of samples of the new values from posterior (by default $K = 1$). The work [12] proposes to extend the generative model (5.3) by including the information about the labelled nodes in the following way:

$$\begin{aligned} q_\phi(z|y_i, x_i) = \mathcal{N}(z|\mu(y_i, x_i), \sigma^2(x_i)) \\ q_\phi(y|x) = Cat(y|\pi_\phi(x)) \end{aligned} \tag{2.10}$$

using the semi-supervised loss:

$$\begin{aligned} \mathcal{L}_{ssl}(x, y) = \sum_{(x,y)} q_\phi(y|x) \mathbb{E}_{q_\phi(y,z|x)} [\log\ p_\theta(x|y, z) \\ + \log\ p_\theta(y) + \log\ p(z) - \log\ q_\phi(z|x, y)] \end{aligned} \tag{2.11}$$

where $Cat(y|\pi_\phi(x))$ is the multinomial distribution of class labels, $\pi_\phi(x)$ is the output from MLP. Finally, based on the combination of (2.10), (2.11) loss functions, we can train VAE in a semi-supervised way. Note that this algorithm supports batchwise training strategies, which positively impacts the memory complexity during the training. However, the loss (2.11) has the following main limitations: This method works better for classifying the data without a predetermined graph structure. This specifically indicates that this approach does not ensure competitive performance on datasets built on graphs.

Chapter 3

Markov-Batch Stochastic Approximation algorithm

Since computational and memory limitations remain present issues in all directions of GB-SSL, we would like to make the following contribution. In this section, we first propose a novel linear algorithm called Markov-Batch Stochastic Approximation (MBSA) for solving Personalized PageRank (PPR), which updates nodes by batch and proposes a significantly better tradeoff between memory consumption and convergence rate to an optimal classification result compared with other linear models. Then, we suggest a novel, non-linear scaling graph convolution neural network called MBSA-NN that embeds linear MBSA to avoid memory concerns and greatly cut down on processing time and memory use. We applied MBSA-NN on several very large datasets and show that it can handle graphs with more than $\mathbf{10^7}$ nodes and $\mathbf{2 \times 10^6}$ of features in under **one** minute on one standard machine, including preprocessing, training and inference time. Furthermore, we show that it has significant improvements in terms of memory and time consumption and comparable performance in terms of accuracy with respect to the latest best scaling algorithms. In particular, it consumes **10** times less nodes per batch at training than the only algorithm without out-of-memory issues (PPRGO) on our experiment, and reduces the memory consumption by **50**% at inference.

This chapter is divided into four parts, each of which has its own sections. These parts are organized in the following way: i) we begin by defining the novel Markov-Batch Stochastic Approximation (MBSA) algorithm with theoretical evidence of its convergence to the exact solution of PPR. Additionally, we demonstrate how MBSA could work in a parallel asynchronous regime (pMBSA) and describe the specifics of its C++ development. The experimental results from the point of convergence rate to the exact classification

results and the memory/time consumption during training are also included in this part. Finally, we demonstrate the MBSA ablation analysis; ii) the next part defines the adaptation of MBSA into graph convolution networks and proposes the MBSA-NN algorithm. We state that the locally optimal solution of the graph convolution network on the whole graph is almost surely reached by MBSA-NN. In addition, we compare MBSA-NN with the most recent top graph scaling and graph convolution networks in this part for accuracy, memory use, and time on GPU and RAM; iii) the modification of MBSA and MBSA-NN based on uniform batch sampling is suggested in the following section. Moreover, we theoretically verify in this section that the convergence of MBSA and MBSA-NN described in earlier sections holds with this alteration. Also, we compare the accuracy, memory, and time consumption on the GPU and RAM for this modification; iv) finally, the last part contains the proof of the theorems defined in the previous parts. Also, this part shows the details of experiments with information about parameters of algorithms, links on GitHub repositories, datasets and the technical environment.

3.1 Markov-Batch Stochastic Approximation (MBSA)

The problem of solving System (3.1) in a pure batch regime with a guarantee of an exact solution and lowering memory and computation complexity while maintaining a high accuracy on large graphs is still up for debate with regard to the various GB-SSL neural network and linear algorithm architectures that are currently in use.

$$(I - \alpha\widetilde{A})Z = (1 - \alpha)Y, \tag{3.1}$$

where $\widetilde{A} = D^{-\delta}AD^{\delta-1} = [\widetilde{A}_{i,j}]_{i,j=1}^{n,n} \in \mathbb{R}^{n\times n}$ is a regularized adjacency matrix and $D = \text{diag}(D_{i,i})_{i=1}^{n}$ is a degree matrix with $D_{i,i} = \sum_{j=1}^{n} A_{i,j}$, $Y = [Y_i]_{i=1}^{n}$ is a matrix that represents labels, I is the identity matrix, α is a regularization parameter and $Z = [Z_i]_{i=1}^{n} \in \mathbb{R}^{n\times c}$ is a classification result.

Therefore, this section presents a novel linear algorithm called Markov-Batch Stochastic Approximation (MBSA) that avoids OOM issues. The primary goal of the MBSA algorithm is to solve System (3.1) by updating the batch of nodes with another currently available batch of nodes. This strategy is opposed to the existing nodes updating strategies: as mentioned in previous section, in DSA-SSL, the nodes are updated one by one; in JOR and approximated PPR, each node is updated by its neighbors; in DSBGS and RBGS, batches of nodes are updated by their neighbors; and in RK, the current node is used to update its neighbors. Figure 3.1 illustrates these updating strategies. Updating batches with batches makes MBSA inherently more efficient than its competitors. Besides the choice of updating nodes from batches, MBSA also differs from existing algorithms in the way nodes are updated, as we will see in the sequel.

Let's first establish the notation that will be required for the MBSA algorithm's further explanation. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected and unweighted graph, with $n = |\mathcal{V}|$ the number of nodes and $e = |\mathcal{E}|$ the number of edges. Let $\mathcal{V}_l$ and $\mathcal{V}_u$ denote the sets of labeled and unlabeled nodes, with $n_l = |\mathcal{V}_l|$ and $n_u = |\mathcal{V}_u|$ the number of labeled and unlabeled nodes respectively. From the adjacency matrix $A = [A_{i,j}]_{i,j=1}^{n,n} \in \mathbb{R}^{n\times n}$ representing the graph, and the degree matrix $D = \text{diag}(D_{i,i})_{i=1}^{n}$ with $D_{i,i} = \sum_{j=1}^{n} A_{i,j}$, we define the regularized adjacency matrix $\widetilde{A}$ as $\widetilde{A} = D^{-\delta}AD^{\delta-1} = [\widetilde{A}_{i,j}]_{i,j=1}^{n,n} \in \mathbb{R}^{n\times n}$, where δ is a regularization parameter taking values in $\{0, 0.5, 1\}$. Finally, the graph $\mathcal{G}$ contains information both from the matrix of node features $X = [X_i]_{i=1}^{n} \in \mathbb{R}^{n\times d}$, where d is the number of input features, and from the matrix of node classes $Y = [Y_i]_{i=1}^{n} \in \mathbb{R}^{n\times c}$, where c is the number of classes.

3.1.1 Selection of batches

We outline the MBSA rule of picking a batch of nodes every iteration in this subsection. Let $\mathcal{S} = \{\mathcal{S}_i\}_{i=1}^{s}$ be a set of non-intersecting batches where $\mathcal{S}_i$ is a uniformly sampled batch of nodes without repetition and $s = \lceil \frac{n}{bs} \rceil$ is the number of batches with a predefined batch size bs. Then we denote by $A_{\mathcal{S}_i,\mathcal{S}_j}$ the submatrix of adjacency matrix A on the selected batches. $P = [P_{i,j}]_{i,j=1}^{s}$ is the matrix of the number of edges between batches $P_{i,j} = ||A_{\mathcal{S}_i,\mathcal{S}_j}||_{1,1}$ and $\widetilde{D} = diag(\widetilde{D}_{i,i})_{i=1}^{n}$ is the diagonal matrix where $\widetilde{D}_{i,i} = \sum_{j=1}^{s} P_{i,j}$. Finally, $\widetilde{P} = \widetilde{D}^{-1}P$ is the transition probability matrix between the batches.

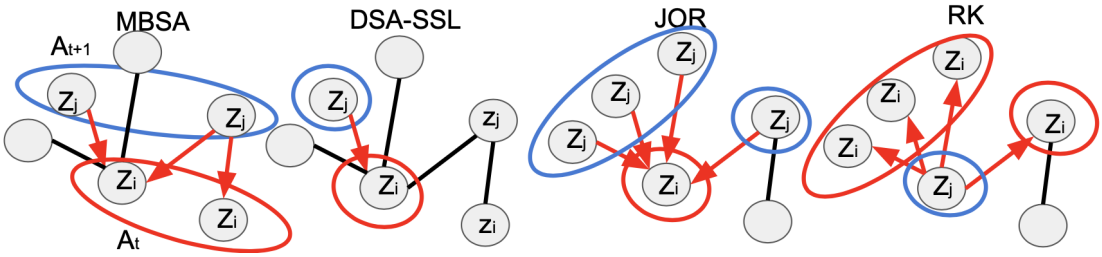

Figure 3.1: Comparison of various strategies for nodes updates, where JOR is a Jacobian over relaxation, RK is a Randomized Kaczmarz [1] and DSA-SSL [2] is a distributed stochastic approximation. The nodes in the red circles will be updates. The blue circles mean the nodes which will be used for updating the nodes in the red circles.

In MBSA, the batches are selected as follows. Let $\mathcal{A}_t$ be the index of the batch chosen from $\mathcal{S}$ at time t. Then, at time $t+1$, the next batch index $\mathcal{A}_{t+1}$ is chosen from the rule:

$$P(\mathcal{A}_{t+1}|\mathcal{A}_t) = Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}, \tag{3.2}$$

where $Q = (1-\epsilon)\widetilde{P} + \frac{\epsilon}{s}E$ is the irreducible counterpart of $\widetilde{P}$, $E \in \mathbb{R}^{s\times s}$ is an all-one matrix and $\epsilon \in (0,1)$ is a damping factor [68]. The first part of Q gives a higher probability of selecting a batch with a large number of edges in common with the current one, while the second part selects a batch with very few edges between them, which in turn allows to update all the batches in $\mathcal{S}$ during the τ iterations. This update rule renders $\{\mathcal{A}_t\}$ an $\mathcal{A}$-valued Markov chain.

3.1.2 Node update

We now move to describe the way nodes are updated in our algorithm. Indeed, it is more complex than PowerIteration or JOR in the sense that it updates a batch of nodes by another batch of nodes chosen by rule Eq. (3.2) per iteration instead of using all neighbour nodes (e.g. Figure 3.1). However, this allows MBSA to have a fixed number of

nodes in a batch depending on bs during the iteration. In other words, MBSA avoids the memory bottleneck where the node for the update has a number of neighbour nodes close to n.

MBSA is defined in Algorithm 3 and relies on the following additional parameter $\eta(t) = 1/(1+t)^{\gamma}$, corresponding to the step size for the node updates where $\gamma \in (0, 1]$ is a parameter of step length regularization.

Algorithm 3: Markov-Batch Stochastic Approximation (MBSA)

INPUT : A, Y, $\widetilde{P}$, Q, δ, α, τ, bs, ϵ, γ

INITIALIZE: $\widetilde{A}$, $Z^0 = Y$, $\mathcal{S}$, $\mathcal{A}_t = \mathcal{A}_0$;

for $t \leftarrow 0$ **to** τ **do**

  Pick $\mathcal{A}_{t+1}$ with probability $Q_{\mathcal{A}_t, \mathcal{A}_{t+1}}$;

  $$Z_i^{t+1} = Z_i^t + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \frac{\widetilde{P}_{\mathcal{A}_t, \mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t, \mathcal{A}_{t+1}}} \left(\frac{\alpha \sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} \widetilde{A}_{i,j} Z_j^t}{\widetilde{P}_{\mathcal{A}_t, \mathcal{A}_{t+1}}} - Z_i^t + (1-\alpha) Y_i \right);$$

  $\mathcal{A}_t = \mathcal{A}_{t+1}$;

end

3.1.3 Theoretical analysis

Finally, another important aspect of MBSA is that it converges almost surely to the exact solution of Eq. (3.1), as stated in Theorem 1.

Theorem 1: Consider MBSA (Algorithm 3) and suppose that $\eta(t) = 1/(1+t)^{\gamma}$ with $\gamma \in (0, 1]$. Then, Z_i^t converges almost surely to Z_i^*:

$$Z_i^t \xrightarrow{a.s.} Z_i^* \quad as\ t \to \infty\ \forall i \in \{1, \ldots, n\},$$

where Z_i^* is the desired solution of (3.1).

Proof. The proof of Theorem 1 relies on techniques from ordinary differential equations (ODE) and is given in Section 3.10. In particular we define $h(Z^t) = (\alpha\widetilde{A} - I)Z^t + (1-\alpha)Y$ as an ODE of MBSA. □

Finally, note that an investigation of the impacts of the γ, ϵ on the step length regularization and the analysis of an optimal bs for MBSA was undertaken, and its results can be seen in Section 3.3.

3.2 The asynchronous parallel MBSA (pMBSA)

Now, we want to show the asynchronous parallel analog of MBSA (Algorithm 3):

Remark 1: Let us redefine the ODE of Algorithm 3 as: $\dot{Z} = Q(t)h(Z^t)$, where $Q(t) = diag(Q(t)_{i,i})_{i=1}^n$ is a diagonal matrix with strictly positive entries on the diagonal. Then using observations from Section 6.4 and Theorem 12.1 from Chapter 12 of [82], the ODE has the same asymptotic behaviour as $\dot{Z} = h(Z^t)$. This means that Theorem 1 holds even for $\dot{Z} = Q(t)h(Z^t)$ as well. In other words we can run Algorithm 3 in an asynchronous parallel mode at each time t and denote it as pMBSA.

With respect to Remark 1 (Theorem 1) we can run MBSA (Algorithm 3) in an asynchronous parallel mode. Indeed, at each time t, we have $Q(t)_{j,j} = 1, \forall j \in \{\mathcal{S}_i^t\}_{i=1}^m$, where $\mathcal{S}^t = \{\mathcal{S}_i^t\}_{i=1}^m$ is a set of batches for updates over m parallel threads at time t, and $Q(t)_{j,j} = 0$, otherwise. Also, note that $\mathcal{S}_i^{t+1}$ is sampled according to Rule 3.2 with respect to the previous batch indices $\mathcal{A}_t$ chosen from $\mathcal{S}^t$. It means that MBSA can updates several different batches $\mathcal{A}_t$ in parallel at time t. We represent the MBSA algorithm in the parallel regime below in Algorithm 4 where m is the number of threads and the cycle

Algorithm 4: parallel Markov-Batch Stochastic Approximation (pMBSA)

INPUT : A, Y, δ, α, τ, bs, m

INITIALIZE: $\widetilde{A}$, $Z^0 = Y$, $\mathcal{S}^t$;
for $t \leftarrow 0$ **to** τ **do**
  for $\mathcal{A}_t$ **to** $\mathcal{S}^t$ **do**
    Pick $\mathcal{A}_{t+1}$ with probability $Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}$, for generation of $\mathcal{S}^{t+1}$;
    $$Z_i^{t+1} = Z_i^t + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}} \left(\frac{\alpha \sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} \widetilde{A}_{i,j} Z_j^t}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}} - Z_i^t + (1-\alpha) Y_i \right);$$
  end
  $\mathcal{S}^t = \mathcal{S}^{t+1}$;
end

over $\mathcal{A}_t$ indexes from $\mathcal{S}^t$ computes parallel by m threads at time t.

3.2.1 Details of parallel implementation on C++

pMBSA was implemented in the multithreading regime in C++, with function and data binding using Cython. For this implementation, we are using the Eigen[1] mathematical library. Eigen is a C++ template library for linear algebra. To eliminate the overhead

[1] https://eigen.tuxfamily.org/index.php?title=Main_Page

of copying large amounts of computational data between Python and C++ code, we used *Eigen* :: *Mapstruct* for application in C++ structures allocated in Python, which avoids the temporary objects, copy constructors, etc. Also, we have involved the SIMD instructions [83] such as AVX and SSE2 for the optimization, which is available in the Eigen library.

Note that we resolved the collisions with resource locking by keeping in memory two matrices: Z^t at the current moment t for reading and Z^{t+1} for writing. This modification holds the correctness of the pMBSA algorithm. The synchronization of these matrices occurs after each cycle of parallel updates of batches. Note that m threads run simultaneously, where m is the number of physical processor cores. We need to point out that the synchronization operation requires copying a contiguous memory segment and is very fast in C++. The sparse graph matrices had to be pre-processed in Python, split into batches and serialized in advance, for use in C++, due to differences in the storage formats of sparse matrices. This is a reasonably cheap operation, but due to the preliminary division of the feature matrix into batches, multiple extractions of a sparse submatrix are not required.

Finally, in Section 3.4 we compare the C++ implementation with Python3.8 and show that multithreading implementation on C++ m times faster than one thread Python implementation.

3.3 Ablation studies of MBSA

3.3.1 Impact of γ and ϵ on convergence rate

In order to analyze the effect of the step size regularization parameter γ and ϵ on MBSA performance, we consider the following grids of values: $\gamma \in [0.1, 0.3, 0.5, 0.7, 0.9]$, $\epsilon \in [0.1, 0.5, 0.9]$, and fix the other parameters $\tau = 500$ and $bs = 512$. Moreover, for stable estimation of convergence rate we repeat 50 times experiments of convergence MBSA for each pair of γ and ϵ. Also, the exact solution Z^* of System (3.1) is taken as reference for the best classification accuracy:

$$Z^* = (I - \alpha\widetilde{A})^{-1}(1 - \alpha)Y. \tag{3.3}$$

Figure 3.2 displays the evolution of accuracy during training for different values of ϵ (columns) and of γ (colored lines). Note that the accuracy computed in the following way:

$$\text{Accuracy}(Z^t, Y^*) = \frac{\sum_{i=0}^{n} I\{\text{argmax}(Z_i^t) = \text{argmax}(Y_i^*)\}}{n},$$

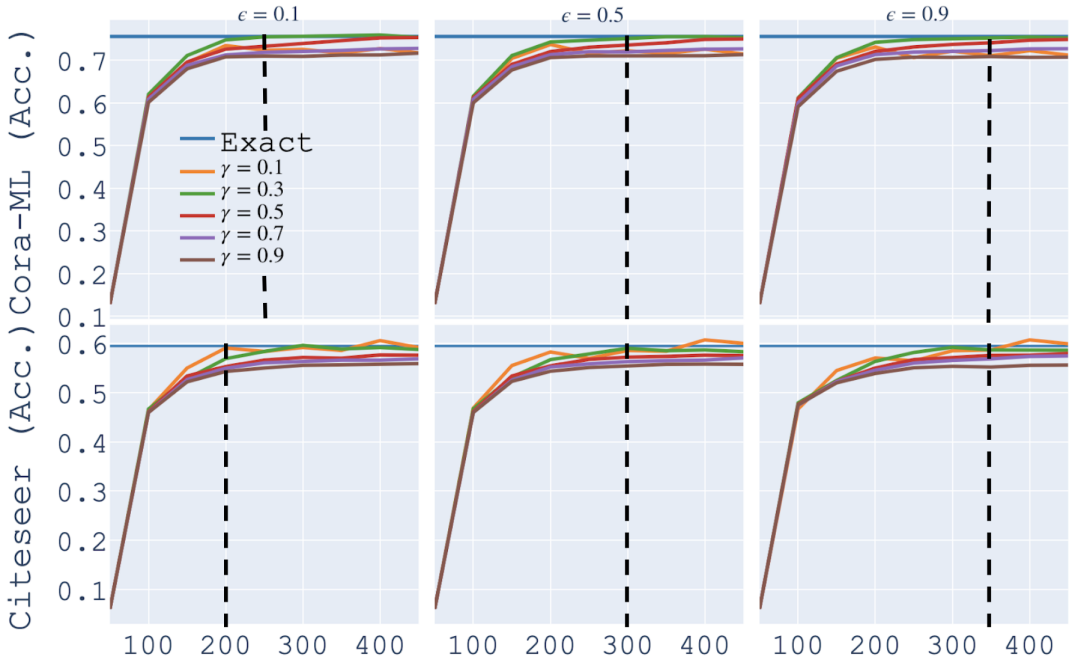

Figure 3.2: Average accuracy of MBSA for each pair of step size power $\gamma \in [0.1, 0.3, 0.5, 0.7, 0.9]$ and damping factor $\epsilon \in [0.1, 0.5, 0.9]$ at each 50 interaction (x-axis). The blue line shows the classification accuracy of exact solution (3.3). The black dashed line shows the first time of convergence. Impact of power of step size γ and damping factor ϵ on convergence to best accuracy.

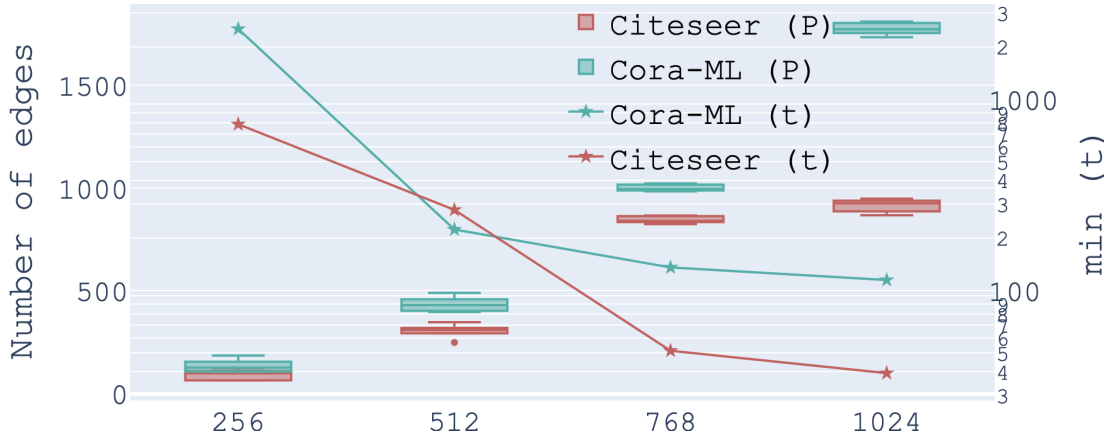

Figure 3.3: Average minimum number of iterations $min(t)$ (right y-axis, lines) and number of edges between batches ($mean(P)$) (left y-axis, boxes) over 50 random runs of MBSA for each batch size bs (x-axis).

where argmax$(\cdot)$ returns the index of the maximum value in the row, $I\{\cdot\}$ is an indicator function and Y^* is a matrix of ground truth labels. It shows that MBSA converges faster to the best classification accuracy with a large step size e.g. $\gamma = 0.3$ in $\eta(t) = (1+t)^{-\gamma}$ and with a small damping factor ($\epsilon = 0.1$). Our intuition behind this observation is large step sizes perform intensive updates of nodes from $\mathcal{S}_{\mathcal{A}_t}$ by $\mathcal{S}_{\mathcal{A}_{t+1}}$, while taking Q with a small ϵ allows selecting a subset $\mathcal{S}_{\mathcal{A}_{t+1}}$ with more edges with $\mathcal{S}_{\mathcal{A}_t}$, and thus improves the update of nodes in the batch. Also note that we can get convergence with a high oscillation to large a step size (e.g. $\gamma = 0.1$, Figure 3.2). For this reason, in next experiments, we consider $\gamma = 0.3$ as an optimal power for step size since it leads to a faster a more stable convergence than others.

3.3.2 Impact of batch size on convergence rate

We now analyze the impact of the batch size bs on the minimum number of iterations necessary to get classification accuracy equal to that of exact solution (3.3). In order to do so, we set $\gamma = 0.3$ and $\epsilon = 0.1$ based on the previous experiment, and we consider $bs \in [256, 512, 768, 1024]$. Figure 3.3 shows that the minimum number of iterations decreases with respect to the batch size, while the mean value of edges between batches ($mean(P)$) increases. This finding is based on the obvious intuition that the more edges between batches $\mathcal{S}_{\mathcal{A}_t}$ and $\mathcal{S}_{\mathcal{A}_{t+1}}$ we have, the more likely it is that we will update every node in batch $\mathcal{S}_{\mathcal{A}_t}$, which accelerates updating of every node in the graph. In particular, Figure 3.3 shows that if $mean(P)$ is close to bs then MBSA

converges faster to the exact solution in terms of classification accuracy (e.g. for Cora-ML: $bs = 512; mean(P) = 445; min(t) = 221.34$). Since $min(\tau)$ significantly decreasing even with small batch size (e.g. $bs = 512$), we propose to use the following equation for minimal optimal batch size:

$$bs^*_{min} = \frac{n}{median(D)}, \tag{3.4}$$

where we recall that n is the number of nodes and $median(D)$ is the median value of node degree over the graph. In particular, this rule (3.4) holds for Cora-ML ($n = 2810$, $median(D) = 5$, $bs^*_{min} = 562$) and Citeseer ($n = 2110$, $median(D) = 3$, $bs^*_{min} = 703$) with respect to results on Figure 3.3. We assume that a low median degree requires using a large batch size to guarantee that $mean(P) \sim bs$.

3.4 Experimental results for MBSA

3.4.1 Convergence analysees

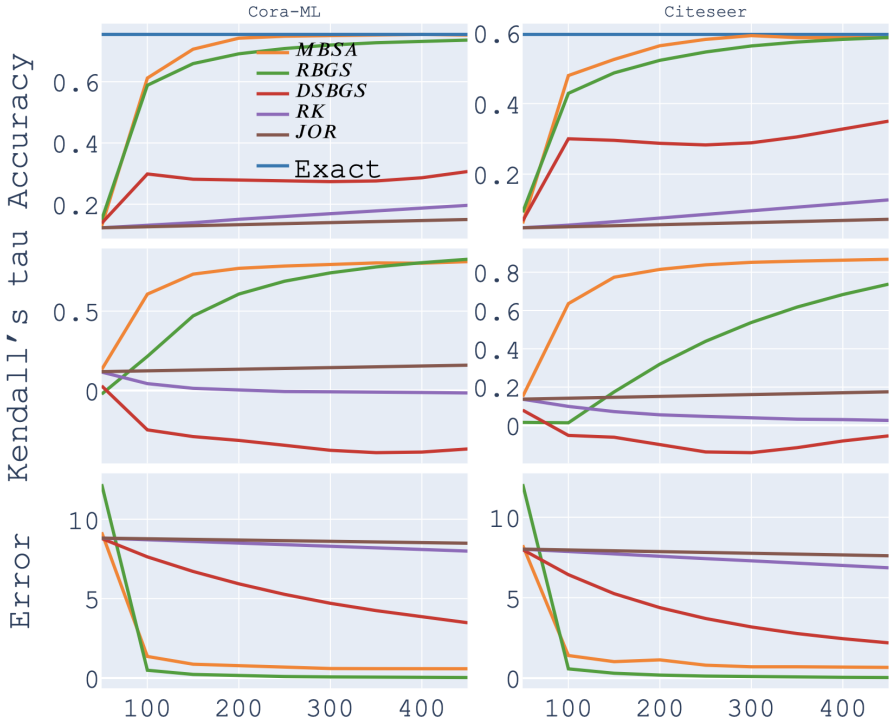

Figure 3.4: Convergence analysis.

In this subsection, we analyze the behavior of MBSA with respect to other updating strategies from the literature, namely Jacobian over relaxation (JOR), Double Stochastic

part Gauss-Seidel (DSBGS) [65], Randomized Kaczmarz (RK) [1] and Randomized part Gauss-Seidel (RBGS) [66]. Note that we excluded DSA-SSL from the comparison due to its extremely slow convergence rate. The results comparison of MBSA with other linear algorithms on Cora-ML and Citeseer are presented in Figure 3.4 in terms of convergence rate to the accuracy of exact solution of System (3.1) (top); Kendall's tau [84] coefficient, giving a rank correlation score between the exact solution and the solution from the algorithms (middle, values close to 1 corresponds to total agreement). Note that for the computation of Kendall's tau we flattened the Z^t and Z^* in arrays with $n * c$ elements. Then, we apply the following equation for estimating Kendall's tau:

$$tau = \frac{(P - Q)}{sqrt((P + Q + T) * (P + Q + U))}$$

where P is the number of concordant pairs, Q the number of discordant pairs, T the number of ties only in Z^t, and U the number of ties only in Z^*; and error of the solution of linear system (3.1) (bottom). In particular, the error is computed using Frobenius matrix norm:

$$\text{Error} = ||(I - \alpha\widetilde{A})Z^t - (1 - \alpha)Y||_F$$

Each quantity is averaged over all runs. Figure 3.4 shows that MBSA outperforms the other algorithms in terms of fast recovery of the accuracy and PPR rank (Kendall's tau correlation), seconded by RBGS. On the other hand, we can see that MBSA's error for System (3.1) is lower than most algorithms, except RBGS. These graphs show that to obtain good classification accuracy, it is more important to rapidly recover the PPR order rather than find the best approximation of the exact solution, unlike what was initially assumed in [4, 23, 71].

3.4.2 Memory vs Time tradeoff

Here, we compare the memory and computation time requirements of MBSA and other linear methods. For example, Figure 3.5 shows the comparison of MBSA with the aforementioned algorithms in terms of the average peak of memory consumption in $\tau = 500$ iterations and average minimum number of iterations for convergence to the accuracy of the exact solution. The minimum number of iterations $min(t)$ gives the moment where the algorithm's accuracy is equal to the accuracy of the exact solution of System (3.1) on the test set, averaged over all runs. In particular, Figure 3.5 shows that MBSA converges at least two times faster to the accuracy of the exact solution of System (3.1) than RBGS (e.g. Citeseer: MBSA min(t)=280.45; RBGS min(t)=576.31). Moreover, it consumes a small amount of memory since it updates a sparse batch of

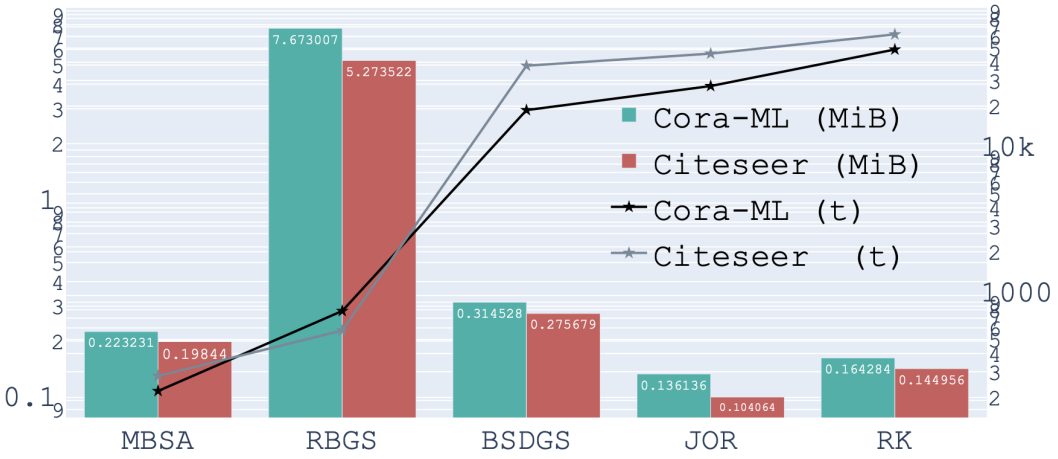

Figure 3.5: Average memory consumption (MiB, left y-axis, bars-log) per iteration and mean minimum iterations (right y-axis, lines-log, $min(t)$) over 50 random runs for each algorithms (x-axis).

nodes per iteration. Finally, Figure 3.5 shows that MBSA offers an optimal very good tradeoff between memory consumption and convergence rate with respect to other linear batch algorithms. The implementation details of the algorithms we took for comparison, a description of the computation environment, the definition of optimal parameters of algorithms for experiments and dataset statistics are in Section 3.11. The implementation of linear algorithms, MBSA on Python, pMBSA on C++ and links on datasets in experiments are available in Section 3.11 as well.

3.5 MBSA for graph convolution networks

This section discusses how MBSA can be used in existing graph convolution network algorithms both at the training and inference steps to overcome memory issues and decrease time consumption. We propose to adapt MBSA for the computation of PPR in a graph convolution network due to the following properties: i) compared to other linear batch algorithms, MBSA has the best memory use and convergence rate (see Figure 3.5); ii) MBSA avoids keeping in memory of the whole adjacency matrix and updating the nodes through the latent representations of its neighbours as is done, e.g. in GCN and APPNP;

3.5.1 Training step

In a similar fashion than is done in APPNP (see Chapter 2), we propose to embed the MBSA updates of the classification results into a graph convolution network (e.g. a multi-layer perceptron). Here, the use of MBSA allows to make this update batchwise and thus overcome memory issues and decrease time consumption while holding guarantees of convergence to the exact solution of PPR. The intuition for adaptation of MBSA for graph convolution neural networks is lying on the idea to update the hidden representation features of one batch of nodes by the hidden representation of features from another batch of nodes. This adaptation of MBSA we named as MBSA-NN. The full algorithm of MBSA-NN is defined in Algorithm 5 and relies on the following parameters: $W^t_{(0)} \in \mathbb{R}^{d\times m}$ and $W^t_{(1)} \in \mathbb{R}^{m\times c}$ are trainable weight matrices at time t for converting input node features form d-space into hidden m-space and from hidden m-space into classes c-space which are training by minimizing of the convex classification loss $\mathcal{L}(\cdot,\cdot)$ as in APPNP, GCN, PPRGO etc.; β is the step size for computing the gradient weight matrices $W^{(0)t}$, $W^{(1)t}$; $\Gamma^{(0)}(\cdot)$ and $\Gamma^{(1)}(\cdot)$ are layer activation functions which can be defined $\Gamma^{(0)}(\cdot) = ReLU(\cdot)$, $\Gamma^{(1)}(\cdot) = softmax(\cdot)$.

Algorithm 5: MBSA-NN

INPUT : $\widetilde{A}$, Y, X, α, γ, τ

for $t \leftarrow 0$ **to** τ **do**
- Pick $\mathcal{A}_{t+1}$ with probability $Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}$;
- $\widetilde{Z}_j = I\{j \in \mathcal{S}_{\mathcal{A}_{t+1}}\}(\Gamma^{(0)}(X_j W^t_{(0)})W^t_{(1)})$;
- $\widetilde{Z}_i = I\{i \in \mathcal{S}_{\mathcal{A}_t}\}(\Gamma^{(0)}(X_i W^t_{(0)})W^t_{(1)})$;
- $Z^{t+1}_i = Z^t_i + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}} \times \left(\frac{\sum_{j\in\mathcal{S}_{\mathcal{A}_{t+1}}} \widetilde{A}_{i,j}\widetilde{Z}_j}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}} - Z^t_i \right)$;
- $\mathcal{L}_{\mathcal{S}_{\mathcal{A}_t}} \leftarrow I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\mathcal{L}(Y_i, \Gamma^{(1)}(Z^{t+1}_i)) + \mathcal{L}(Y_i, \Gamma^{(1)}(\widetilde{Z}_i))$;
- $(W^{t+1}_{(0)}, W^{t+1}_{(1)}) \leftarrow$ update by $\beta \left(\frac{\partial \mathcal{L}_{\mathcal{S}_{\mathcal{A}_t}}}{\partial W^t_{(0)}}, \frac{\partial \mathcal{L}_{\mathcal{S}_{\mathcal{A}_t}}}{\partial W^t_{(1)}} \right)$ gradient step;
- $\mathcal{A}_t = \mathcal{A}_{t+1}$;
- $(W^t_{(0)}, W^t_{(1)}) = (W^{t+1}_{(0)}, W^{t+1}_{(1)})$

end

3.5.2 Theoretical analysis

Moreover, note that MBSA-NN defined in Algorithm 5 converges almost surely to the locally optimal solution of graph convolution network on the entire graph, as stated in

Theorem 2.

Theorem 2: Consider MBSA-NN (Algorithm 5) and suppose that $\eta(t) = (\frac{1}{1+t})^{\gamma}$ where $\gamma \in (0, 1]$, $\frac{\beta}{\eta(t)} \to 0$ where β is the step size for computing the gradient weight matrices $W^{(0)}$, $W^{(1)}$. Then, Z_i^t converges almost surely to $\Gamma^{(1)}(\widetilde{A}(\Gamma^{(0)}(X_i W_{(0)})^* W_{(1)})^*)$:

$$Z_i^t \xrightarrow{a.s.} \Gamma^{(1)}(\widetilde{A}(\Gamma^{(0)}(X_i W_{(0)}^*) W_{(1)}^*)) \quad as\ t \to \infty\ \forall i \in \{1, \dots, n\},$$

where $W_{(0)}^*, W_{(1)}^*$ are locally optimal solutions of $\mathcal{L}(\cdot, \cdot)$ found by computing the gradient steps.

Proof. The proof of Theorem 2 relies on the comparison of two ordinary differential equations (ODE) and is given in Section 3.10. In particular, we define that $h(Z^t) = \widetilde{A}(\Gamma^0(XW_{(0)}^t)W_{(1)}^t) - Z^t$ is an ODE of MBSA-NN. □

Remark 2: In our actual experiments, the number of iterates was not too large and a very small (relative to $\eta(t)$) constant $\beta(t) \equiv \beta$ seemed to suffice.

3.5.3 Implementation details

Since only labeled nodes are used by neural networks to calculate classification loss, we take into account the following remark:

Remark 3: Let us redefine the ODE of Algorithm 5 as $h(Z^t) = Q(t)(\widetilde{A}(\Gamma^0(XW_{(0)}^t)W_{(1)}^t) - Z^t)$, where $Q(t) = diag(Q(t)_{i,i})_{i=1}^n$ is a diagonal matrix with strictly positive entries on the diagonal at each time t. Then by Remark 1 and the results from Chapter 6.4 and Corollary 2.1 in Chapter 2 of [82] the $h(Z^t) = Q(t)(\widetilde{A}(\Gamma^0(XW_{(0)}^t)W_{(1)}^t) - Z^t)$ has the same asymptotic behaviour as $h(Z^t) = (\widetilde{A}(\Gamma^0(XW_{(0)}^t)W_{(1)}^t) - Z^t)$. Indeed, Theorem 2 holds even for $h(Z^t) = Q(t)(\widetilde{A}(\Gamma^0(XW_{(0)}^t)W_{(1)}^t) - Z^t)$.

The Remark 3 allows us to use the diagonal matrix $Q(t)$ in the following way: $Q(t)_{i,i} = 1\ \forall i \in \mathcal{V}_l,\ Q(t)_{i,i} = 0$, otherwise. In other words, it allows to make the following modifications in MBSA-NN (Algorithm 5):

1. we reduce the number of unlabelled nodes to the number of neighbour nodes of labelled nodes. This reduction aims at skipping the unlabelled nodes, which do not have connections with labelled ones and cannott impact their updating process. The unlabeled neighbour nodes are denoted as $\mathcal{V}_{un}$;

2. we generate two sets of non-intersecting batches of labelled and unlabeled nodes: $\widetilde{\mathcal{S}} = \{\widetilde{\mathcal{S}}_i\}_{i=1}^l$ and $\mathcal{S}' = \{\mathcal{S}_i'\}_{i=1}^u$ where $\widetilde{\mathcal{S}}$ and $\mathcal{S}'$ are sets of uniformly sampled

batches of nodes without repetition from $\mathcal{V}_l$ and $\mathcal{V}_{un}$, respectively, and $u = \lceil \frac{|\mathcal{V}_{un}|}{bs} \rceil$, $l = \lceil \frac{|\mathcal{V}_l|}{bs} \rceil$ are the number of batches in each set;

3. we recompute the number of edges between the aforementioned sets of batches $P = [P_{j,k}]_{j,k=1}^{l,u}$.

Then, with respect to the modifications above, we can rewrite MBSA-NN as described in the following Algorithm 6.

Algorithm 6: MBSA-NN (for implementation)

INPUT : $\widetilde{A}$, Y, X, α, γ, *epochs*

for $t \leftarrow 0$ **to** τ **do**
  for $\widetilde{\mathcal{S}}_{\mathcal{A}_t} \in \widetilde{\mathcal{S}}$ **do**
    Pick $\mathcal{A}_{t+1}$ from $\mathcal{S}'$ by rule (3.2) with respect to the recomputed P;
    $\widetilde{Z}_j = I\{j \in \mathcal{S}'_{\mathcal{A}_{t+1}}\}(\Gamma^{(0)}(X_j W^t_{(0)})W^t_{(1)})$;
    $\widetilde{Z}_i = I\{i \in \widetilde{\mathcal{S}}_{\mathcal{A}_t}\}(\Gamma^{(0)}(X_i W^t_{(0)})W^t_{(1)})$;
    $Z_i^{t+1} = Z_i^t + \eta(t) I\{i \in \widetilde{\mathcal{S}}_i\} \frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}} \times \left(\frac{\sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} \widetilde{A}_{i,j} \widetilde{Z}_j}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}} - Z_i^t \right)$;
    $\mathcal{L}_{\widetilde{\mathcal{S}}_{\mathcal{A}_t}} \leftarrow I\{i \in \widetilde{\mathcal{S}}_{\mathcal{A}_t}\}\mathcal{L}(Y_i, \Gamma^{(1)}(Z_i^{t+1})) + \mathcal{L}(Y_i, \Gamma^{(1)}(\widetilde{Z}_i))$;
    $(W^{t+1}_{(0)}, W^{t+1}_{(1)}) \leftarrow$ update by $\beta \left(\frac{\partial \mathcal{L}_{\widetilde{\mathcal{S}}_{\mathcal{A}_t}}}{\partial W^t_{(0)}}, \frac{\partial \mathcal{L}_{\widetilde{\mathcal{S}}_{\mathcal{A}_t}}}{\partial W^t_{(1)}} \right)$ gradient step;
    $(W^t_{(0)}, W^t_{(1)}) = (W^{t+1}_{(0)}, W^{t+1}_{(1)})$
  end
end

Note that Remark 3 shows that Theorem 2 holds for Algorithm 6 even in case when $n_l \ll n_u$. In particular, it means that at stochastic step in Algorithm 6 can updates only $\mathcal{V}_l$ nodes per iteration, this is critical for computation of $\mathcal{L}(\cdot,\cdot)$ and still guaranties that Theorem 2 holds. Also, note that Algorithm 5 as well as Algorithm 6 do not have a fixed number of necessary neighbors for labeled nodes, making the regularized matrix $\widetilde{A}_{i,j}$ $\forall i \in \mathcal{S}_{\mathcal{A}_t}, \forall j \in \mathcal{S}_{\mathcal{A}_{t+1}}$ sparser than the one used in PPRGO [51]. Moreover, Algorithm 5 and Algorithm 6 require only $\lceil 2 * bs \rceil$ nodes at each step compared to $\lceil bs * k \rceil$ in PPRGO [51], where k is the number of neighbors for each labeled node from bs or $\lceil bs * n \rceil$ in AGP [49] in the worst case of preprocessing. Furthermore, MBSA-NN does not make a neighbour node sampling per iteration as Graph-Saint [56], significantly reducing the computational complexity at training. In particular, it is because MBSA-NN makes the batch partitioning ($\mathcal{S}$) at the preprocessing step, and during the training, it relies on the simple batch sampling defined in the Rule (3.2). More generally, MBSA-NN can

be used in any graph convolution networks relying on PowerIteration method [6, 14, 31], making such algorithms scalable for large graphs. For example, we propose to scale APPNP by replacing the stochastic part of MBSA-NN with the following convolution layer:

$$\widetilde{Z}_i = \widetilde{Z}_i + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}} \times \left(\frac{\sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} \alpha \widetilde{A}_{i,j} \widetilde{Z}_j}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}} + (1-\alpha)\widetilde{Z}_i \right) \tag{3.5}$$

Then, the loss function in MBSA-NN will be $\mathcal{L}_{\mathcal{S}_{\mathcal{A}_t}} \leftarrow I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\mathcal{L}(Y_i, \Gamma^{(1)}(\widetilde{Z}_i))$. In the sequel, we denote this modification of MBSA-NN as MBSA-APPNP.

3.5.4 Inference step

Inference can be also expensive in graph convolution networks, especially when the number of non-zero edges in the adjacency matrix is huge. This issue in inference appears for instance in PPRGO, APPNP and GCN as they apply the PowerIteration method to spread logits through the graph. Indeed, PowerIteration updates all the nodes from all their neighbors during iterations, which takes both time and memory. In order to speed up the inference of Algorithms 5,6 we use pMBSA in parallel regine (see Remark 1, Algorithm 4) to spread the logits ($softmax(\widetilde{Y})$) through the graph.

3.5.5 Limitation

Algorithms 3, Algorithms 4, Algorithm 5 and Algorithm 6 do not support the uniform batch selection strategy, which leads to the necessity to compute the transition probability matrix between batches $\widetilde{P}$. This considered as an limitation due to the fact that the dimension of matrix $\widetilde{P}^{s \times s}$ where $s = \lceil \frac{n}{bs} \rceil$ is the number of batches depends on number of nodes and batch size. In particular, this can leads to a computational and memory issue if we consider the extremely large graphs with small batch sizes. This is possible to avoid by modification of these algorithms with $\widetilde{P} = [\widetilde{P}_{i,j}]_{i,j=1}^{s}$ is the transition probability matrix such that $\widetilde{P}_{i,j} = 1/s$ where $s = C_n^{bs}$ is a number of batches and C_n^{bs} denotes the bs-combinations out of n. However, in such a case, we cannot guarantee the theoretical convergence to the optimal solution. In Section 3.8 we proposed solution of this limitation.

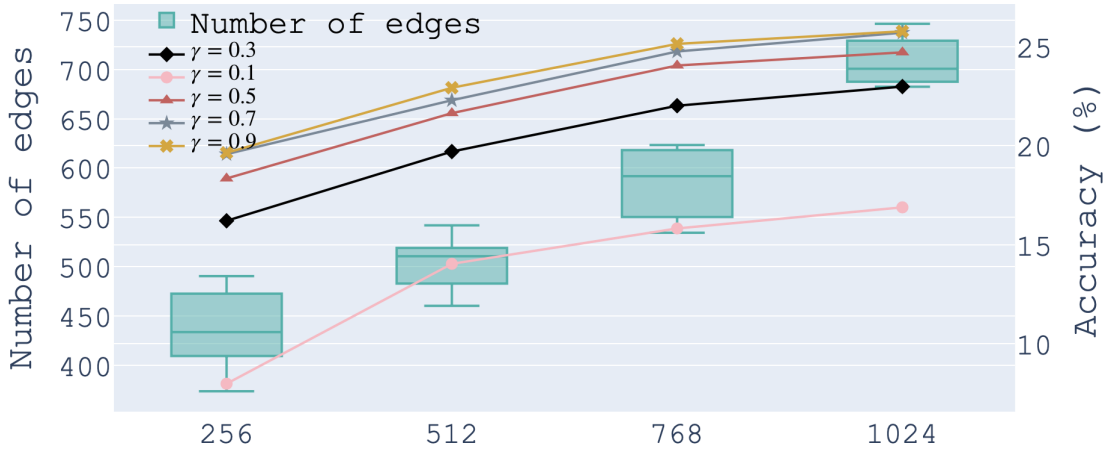

Figure 3.6: Training step for Reddit: Average number of edges between labelled and unlabelled nodes in batch (left y-axis, boxes) and Accuracy (%, right y-axis, lines) over 50 random runs for each batch size (bs, x-axis).

3.6 Ablation studies of MBSA-NN

3.6.1 Impact of γ and bs on the accuracy of MBSA-NN

This subsection focuses on estimating the impact of the step and the batch sizes on the training of MBSA-NN. Because of that we show our experimental results with the following parameters: $bs \in [128, 256, 512, 1024]$; $\gamma \in [0.3, 0.5, 0.7]$; $\tau = 500$ and $\epsilon = 0$. Moreover, for stable performance estimation, we repeat 10 times experiments for each pair of γ and bs. For the sake of generality during the following experiments for inference we use one thread MBSA (Algorithm 3).

Figure 3.6 shows that increasing the number of edges between labelled and unlabelled nodes in batches mainly impacts accuracy as well as increasing the power of step size γ. The intuition under this result is based on the fact that the training of MBSA-NN focuses on updating the labelled nodes, and the more edges we have between labelled and unlabelled nodes, the better representation for labelled nodes we get. Moreover, Figure 3.6 shows that good accuracy is achievable even with a small $bs = 512$ where the number of edges between labelled and unlabelled nodes equals the batch size. Note that this result repeats the intuition of convergence rate of MBSA in Figure 3.3.

3.6.2 Impact of MBSA/pMBSA at inference on the accuracy of MBSA-NN

In this subsection, we assess the effects of MBSA and pMBSA at the inference of classification results, specifically how they affect the performance (accuracy) of MBSA-NN and MBSA-APPNP in the inference. During these experiments, we considered two modifications of the inference MBSA-NN and MBSA-APPNP: one-thread MBSA with $\tau = 120$; parallel multithread pMBSA over max number of cores[2] with $\tau = 20$. Note that for a fair comparison, we used the same neural network hyper-parameters for MBSA-NN and MBSA-APPNP as in the state-of-the-art architecture of APPNP.

Table 3.1 shows that the parallel inference (pMBSA) provides the competitive results in all of the cases for MBSA-NN and MBSA-APPNP. In particular, this means that pMBSA keeps the accuracy close to MBSA and reduces the number of iterations $m = 6$ times where m is the number of threads, which experimentally guarantee that Remark 1 holds.

Table 3.1: Average Accuracy (%) in one thread (o) vs six threads (p) regimes of MBSA at inference over 10 repetitions.

	Cora*	Pubmed	Yelp	Reddit	OGBN*	Mag*
MBSA-NN(o)	**62.1**	76.3	34.9	19.1	**33.2**	**70.2**
MBSA-NN(p)	59.1	**76.8**	32.7	19.7	27.7	70.1
MBSA-APPNP(o)	62.0	75.9	**37.2**	**27.7**	30.2	63.7
MBSA-APPNP(p)	58.6	76.4	36.0	**27.7**	26.3	60.2

3.7 Experimental results for MBSA-NN

Now, we investigate the MBSA-NN performance, time and memory consumption compared to the most recent best scaling state-of-the-art algorithms. In order to do this, we compare the MBSA-NN to alternative scaling algorithms on extremely large datasets and demonstrate how it resolves the out-of-memory issues on CoraFull [85], Pubmed [86], Yelp [56], Reddit [87] OGBN-products [88] and MAG-coarse [51] datasets. A descriptive statistic, details of data preprocessing and references for the datasets used in this section are provided in Section 3.11.

[2] 6 cores in Intel CoreI7

3.7.1 Performance (Accuracy)

The MBSA-NN (Algorithm 5) has been compared with the recent bests: neighbor selection PPRGO [51], model simplification AGP [49], layer sampling (GAS) [62] and subgraph sampling Graph-Saint [56] which also work in a batch regime and outperform other scaling algorithms such as SGC, clusterGCN, GBP, and FastGCN , on large graphs. Please take note that we did not observe Shadow-GNN [55] since there was a critical incompatibility between the environment necessary for Shadow-GNN and the environment used to compute MBSA-NN and the other algorithms. Also, we compare MBSA-NN with APPNP to see the holds of Theorem 2 in practice. In particular, we want to answer the question: **Does the batchwise stochastic approximation in the graph convolution network performs as well as the graph convolution network on the complete graph?** The results in terms of accuracy are presented in Table 3.2 where Cora*, OGBN* and Mag* are a CoraFull, OGBN-products and MAG-coarse datasets respectively.

Table 3.2: Average Accuracy (%) over 5 random train/validation/test splits where ‡ is a notation for OOM and ‡|‡ is a OOM(GPU) | OOM(RAM) respectively.

	Cora*	Pubmed	Yelp	Reddit
APPNP	57.9 ± 0.08	$\underline{\mathbf{79.0 \pm 0.13}}$	‡|‡	‡|‡
PPRGO	59.3 ± 0.39	75.3 ± 2.52	36.6 ± 2.67	22.3 ± 0.69
AGP	59.9 ± 0.67	73.0 ± 2.88	36.4 ± 3.42	‡|‡
Graph-Saint	58.7 ± 0.58	73.6 ± 2.15	‡|42.1 ± 4.65	‡|35.4 ± 0.61
GAS	60.4 ± 0.46	77.1 ± 2.46	$\underline{\mathbf{51.6 \pm 5.87}}$	34.7 ± 1.42
MBSA-NN	$\underline{\mathbf{62.4 \pm 0.64}}$	77.3 ± 3.13	37.3 ± 5.39	$\underline{\mathbf{35.9 \pm 0.45}}$
MBSA-NN(p)	59.4 ± 0.29	75.7 ± 3.91	34.6 ± 4.86	35.8 ± 0.30
MBSA-APPNP	62.0 ± 0.24	76.4 ± 2.85	37.2 ± 4.61	27.7 ± 1.32
PPRGO-MBSA	60.7 ± 0.48	77.1 ± 2.77	35.7 ± 3.72	26.9 ± 0.66

	OGBN*	MAG*
APPNP	‡|‡	‡|‡
PPRGO	35.5 ± 1.04	71.0 ± 0.86
AGP	38.6 ± 2.13	‡|‡
Graph-Saint	‡|39.2 ± 1.92	‡|‡
GAS	$\underline{\mathbf{52.4 \pm 1.37}}$	‡|‡
MBSA-NN	42.9 ± 2.25	74.3 ± 2.35
MBSA-NN(p)	36.9 ± 1.35	70.1 ± 2.43
MBSA-APPNP	30.2 ± 1.44	63.7 ± 2.32
PPRGO-MBSA	37.0 ± 1.67	$\underline{\mathbf{75.1 \pm 0.75}}$

All algorithms in Table 3.2 were run on GPU, except for Graph-Saint which resulted in OOM issues on GPU on several datasets (Reddit, Yelp, OGBN*, Mag*) and was thus run on a CPU. Table 3.2 shows the average results after 5 runs of the algorithms on different train/ validation /test splits. For comparison's sake Table 3.2 contains additional results concerning the use of PPRGO for training and one thread MBSA for inference (instead of PowerIteration), denoted as PPRGO-MBSA; MBSA-APPNP as example of scaling APPNP by MBSA-NN (3.5). Also, note that Table 3.2 shows the best accuracy for MBSA-NN with one thread MBSA and for MBSA-NN(p) with parallel MBSA at inference. The information about hyper-parameters of algorithms in Table 3.2 and description of the technical environment are in Section 3.11.

Table 3.2 provides the comparison in terms of accuracy, from which we can draw the following analysis. First, it shows that MBSA-NN resolves the OOM issues of APPNP on large graphs while retaining its performance. Second, it has a competitive accuracy with respect to the latest scaling algorithms. We can safely assume that MBSA-NN would be even more competitive in terms of accuracy if its parameters were optimally tuned, due to the batch selection strategy (3.2) and the stochastic step size $\eta(t)$. Finally, replacing the inference step in PPRGO with MBSA (PPRGO-MBSA) positively impacts the accuracy of PPRGO. All in all, the results in Table 3.2 highlight the flexibility and quality of MBSA with its ability to scale graph convolution networks (e.g. training in MBSA-NN, MBSA-APPNP) at no loss in accuracy, and to improve other existing scaling algorithms (e.g. inference in PPRGO-MBSA) at inference.

3.7.2 Memory vs Time tradeoff

Training step

We now compare MBSA-NN to its competitors in terms of computational and memory complexity. For a fair comparison, we retained the minimum average accuracy obtained for each dataset from Table 3.2. We repeated 10 times the training of each algorithm for each dataset until they achieved this minimal accuracy and stored the quantities of interest at that point. Fig. 3.7 displays the mean GB (GPU) memory and time consumption during preprocessing, training and inference over all datasets in Table 3.2. Fig. 3.7 shows that MBSA-NN outperforms everywhere in terms of GPU memory consumption and overall running time with one thread/parallel MBSA at inference. The superior performance of MBSA is especially noticeable on large datasets such as Yelp, OGBN* (OGBN-products) and MAG*(MAG-coarse), where the absence of a bar for Graph-Saint, AGP and GAS corresponds to OOM issues (in the case of AGP, due to the storage in

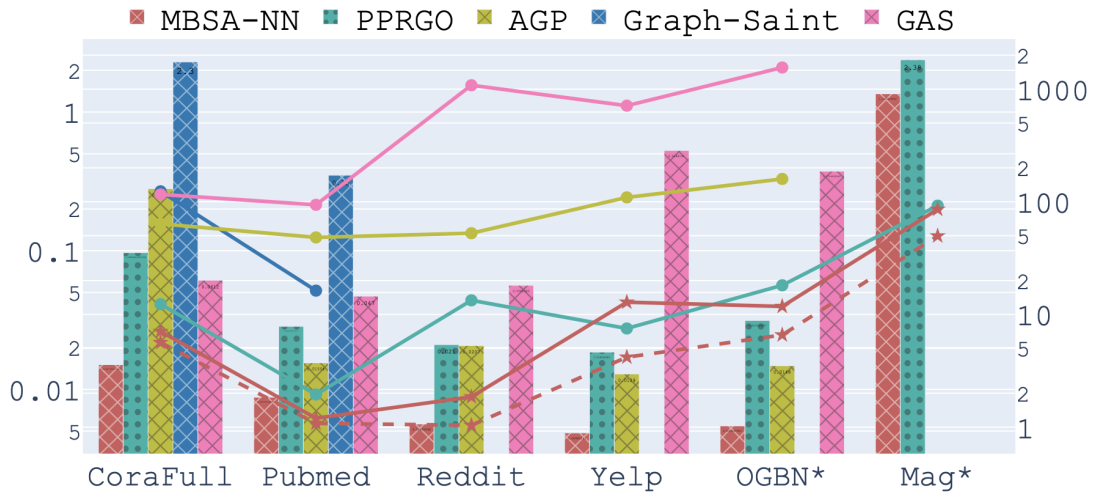

Figure 3.7: Comparison of scaling algorithm w.r.t. Average GPU Memory (GB, left y-axis, bars-logs) and Time (sec., right y-axis, lines) for each dataset over 10 runs. Red non-dash/dash lines are one thread/parallel MBSA receptively.

memory of a dense batch of the feature matrix). Note that only PPRGO and MBSA-NN have handled the MAG* dataset. The improvement of MBSA-NN($\sim 1.3GB$) in terms of GPU memory on that dataset may look less impressive compared to PPRGO ($\sim 2.5GB$) since it primary consumption of GPU comes from training the dense trainable weight matrix $W_0 \in \mathcal{R}^{2M \times m}$, where M is for millions and m is the hidden layer's dimension. Finally, Fig. 3.7 shows that MBSA-NN not only reduces the memory consumption but it also takes **less than one minute to run on MAG-coarse ($\approx 10M^3$ nodes and $2M$ features)**.

Table 3.3 presents details on the RAM memory consumption vs time during preprocessing/training (PR/TR) and inference (IN) steps. Note that Table 3.3 shows the time consumption of MBSA-NN with MBSA and pMBSA at inference. Also, Table 3.3 shows that the number of nodes in batch for PPRGO is always greater than in MBSA-NN, since PPRGO takes $k = 32$ neighbour nodes for each ($bs = 512$) labeled nodes in each batch. On the contrary, MBSA-NN apply everywhere a fixed number of nodes in a batch ($2 * 512$) which allows it to have a stable training time. The worst case of batch generation for PPRGO ($bs * 32$) occurs with Cora* dataset which has a more dense graph. In particular, PPRGO uses 14810 nodes per batch in average on Cora*, which is 10 times higher than MBSA-NN, leading to a significant increase in training time and memory consumption on RAM as well as on GPU.

[3]M is a million.

Table 3.3: Average Memory (MB, RAM)/Time(sec.) complexity at preprocessing/training (PR/TR), inference (IN) steps.

	Cora*		Pubmed	
	PR/TR	IN	PR/TR	IN
PPRGO	430/12.3	28/0.1	372/1.4	39/0.1
AGP	2248 / 62.3	140/1.45	2616.32/ 48.3	1023/0.21
Graph-Saint	745/123.2	44/0.4	516/15.2	42/ 0.5
GAS	38/1172.6	25/ 0.3	13/95.6	15/0.07
MBSA-NN(p)	**19/2.1**	**16/0.08(0.07)**	**7/0.92**	**8/0.07(0.05)**

	OGBN*		MAG*	
	pr/tr	in	pr/tr	in
PPRGO	378/2.3	1372 /15.4	371/42.1	2440/53.7
AGP	5140/113.4	426/ 11.7	OOM	OOM
Graph-Saint	9472/7514	805/518	OOM	OOM
GAS	367/1568.4	412/4.6	OOM	OOM
MBSA-NN(p)	**357/1.9**	**382/8.9(3.8)**	**356/12.1**	**482/59.8(34.1)**

Inference step

Another main bottleneck occurs during inference, which is why we compare memory and time consumption PPRGO with MBSA-NN during inference in Table 3.3 (IN). It shows that pMBSA at inference consumes significantly less memory over all datasets than PPRGO. This is due to the use of batch learning with pMBSA instead of using the complete graph with PI. The difference is significant on the OGBN* and MAG* datasets: PPRGO uses $\sim 1.3GB$ and $\sim 2.3GB$ in RAM over 15 and 53 seconds, compared to $\sim 0.38GB$ and $\sim 0.48GB$ over 3.8 and 34.1 seconds with pMBSA. The implementation of MBSA-NN and MBSA-APPNP on Python3.8 (Tensorflow2.0) and links on datasets in experiments are provided in Section 3.11.

3.8 Uniform MBSA

Since MBSA and MBSA-NN have a limitation based (see Subsection 3.5.5) on the necessity of computing the transition probability matrix between batches $\widetilde{P}$, we propose to replace the Rule 3.2 with the uniform batch selection strategy. Furthermore, we propose this new strategy because neural networks use labelled nodes from batches to compute classification loss during training.

3.8.1 Node update

In order to make a node update step and ensure that each batch contains labelled nodes, we replace the Rule (3.2) for choosing $\mathcal{A}_{t+1}$ in MBSA by a uniform transition probability between batches:

$$Z_i^{t+1} = Z_i^t + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \left(\sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} \frac{\alpha \widetilde{A}_{i,j} Z_j^t}{\frac{2*bs}{n}} - Z_i^t + (1-\alpha) Y_i \right) \tag{3.6}$$

where $\eta(t)$ is defined as in Theorem 1, $\mathcal{S}_i = \mathcal{U}(2 * bs, \mathcal{V})$ is a batch of nodes which we assume it contains bs labelled and unlabelled nodes respectively where $\mathcal{U}(\cdot, \cdot)$ is a function of uniform sampling without repetitions of $2 * bs$ nodes from $\mathcal{V}$, $s = C_n^{2*bs}$ is a number of batches where C_n^{2*bs} the denotes the $(2 * bs)$-combinations out of n and $\widetilde{P} = [\widetilde{P}_{i,j}]_{i,j=1}^s$ is the transition probability matrix such that $\widetilde{P}_{i,j} = 1/s$ (viz. $\widetilde{P}_{\mathcal{A}_t, \mathcal{A}_{t+1}} = Q_{\mathcal{A}_t, \mathcal{A}_{t+1}}$). Note that we define $\widetilde{P}_{i,j} = 1/s$ in order to avoid computation of the number of edges between all possible batches (C_n^{2*bs}). In other words, since $\widetilde{P}_{i,j} = 1/s \ \forall i, j \in \{1, ..., s\}$ we do not need to compute and keep in memory this matrix.

Remark 4: If Z_i^t is updated using the uniform transition probability between batches defined by equation (3.6), then $Z_i^t \to Z_i^*$ with the same conditions as in Theorem 1. The proof is given in Section 3.10.

Since Remark 4 is based on uniform transition probability between batches then we assume that batches at each timestamp can be defined as $\mathcal{S}_{\mathcal{A}_t} = \mathcal{S}_{\mathcal{A}_{t+1}} = [\mathcal{U}(bs, \mathcal{V}_u), \mathcal{S}_l]$ which is the concatenation of bs unlabelled and labelled nodes respectively where $\mathcal{S}_l = \mathcal{U}(bs, \mathcal{V}_l)$ is a sample of labelled nodes. Note that if $bs > \mathcal{V}_l$ then $\mathcal{S}_l = \mathcal{U}(|\mathcal{V}_l|, \mathcal{V}_l)$. This ensures that each batch contains labelled nodes for the computation of the categorical cross-entropy loss $\mathcal{L}$.

3.8.2 Training step

Now we propose a way to adopt uniform MBSA for scaling graph convolution networks. Due to the above conclusions from Remark 4, Equation (3.6) can be adapted for neural networks as in Algorithm 7, that we call uMBSA-NN. Note that uMBSA-NN (Algorithm 7) has the same opportunities as MBSA-NN (Algorithm 5) such as: the fixed number of necessary neighbors for labelled nodes, making the regularized matrix $\widetilde{A}_{\mathcal{S}_{\mathcal{A}_t}, \mathcal{S}_{\mathcal{A}_{t+1}}}$ sparser than the one used in PPRGO. In particular it requires only $2 * bs$ nodes per iteration; also, uniform MBSA can be used in any graph convolution networks relying on PowerIteration method [6,14,31], making such algorithms scalable for large graphs. Furthermore, uniform

Algorithm 7: uMBSA-NN

INPUT : $\widetilde{A}$, Y, X, α, γ, $epochs$

for $t \leftarrow 0$ **to** $epochs$ **do**

$\mathcal{S}_l = \mathcal{U}(bs, \mathcal{V}_l)$;

$\mathcal{S}_{\mathcal{A}_t} = \mathcal{S}_{\mathcal{A}_{t+1}} = [\mathcal{U}(bs, \mathcal{V}_u), \mathcal{S}_l]$;

$\widetilde{Z}_j = I\{j \in \mathcal{S}_{\mathcal{A}_{t+1}}\}(\Gamma^{(0)}(X_j W^t_{(0)})W^t_{(1)})$;

$\widetilde{Z}_i = I\{i \in \mathcal{S}_{\mathcal{A}_t}\}(\Gamma^{(0)}(X_i W^t_{(0)})W^t_{(1)})$;

$\widetilde{Z}_i^{t+1} = \widetilde{Z}_i^t + \eta(t) I\{i \in \mathcal{S}_i\} \left(\frac{\sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} \alpha \widetilde{A}_{i,j} \widetilde{Z}_j}{\frac{2*bs}{n}} - \alpha \widetilde{Z}_i^t \right)$;

$\mathcal{L}_{\widetilde{\mathcal{S}}_{\mathcal{A}_t}} \leftarrow I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\mathcal{L}(Y_i, \Gamma^{(1)}(\widetilde{Z}_i^{t+1}))$;

$(W^{t+1}_{(0)}, W^{t+1}_{(1)}) \leftarrow$ update by $\beta \left(\frac{\partial \mathcal{L}_{\mathcal{S}_{\mathcal{A}_t}}}{\partial W^t_{(0)}}, \frac{\partial \mathcal{L}_{\mathcal{S}_{\mathcal{A}_t}}}{\partial W^t_{(1)}} \right)$ gradient step;

$(W^t_{(0)}, W^t_{(1)}) = (W^{t+1}_{(0)}, W^{t+1}_{(1)})$

end

MBSA and consequently uMBSA-NN have their superiority which is based on the to avoid the computation and keeping in memory the matrix $\widetilde{P}$ contrary to MBSA and MBSA-NN based on sampling Rule 3.2.

3.9 Experimental results for uMBSA-NN

In this subsection, we analyze the differences between uMBSA-NN and MBSA-NN in terms of total RAM memory usage as the number of nodes used during training for both is the same. It should be noted that the preprocessing stage in MBSA-NN and uMBSA-NN focuses on calculating matrix P. Also, emphasize that the memory usage and training time on GPU won't change since the neural network architecture of uMBSA-NN remains the same as MBSA-NN.

3.9.1 Accuracy vs Memory/Time tradeoff

The results in terms of accuracy, time and memory complexity are presented in Table 3.4. Note that the values for memory and time include all steps, namely preprocessing, training and inference. Also, Table 3.4 contains for comparison's sake: PPRGO with $k = 2$, which guarantee that PPRGO will use the same number of nodes in batch as uMBSA-NN during the training (viz. $bs * k = 512 * 2 = 1024$); uMBSA-NN with $\tau = 20$ for reducing the number of iterations for inference by MBSA (Algorithm 4); all of the algorithms in comparison use the number epochs equal to 200. Table 3.4 shows that uMBSA-NN keeps

Table 3.4: Average: Time (sec.), Memory(GB), Accuracy(%) over runs of the algorithms on 50 random train/validation/test splits. The modifications of PPRGO and uMBSA-NN are: PPRGO‡ has $k = 32$, and PPRGO* has $k = 2$; uMBSA-NN‡ has $\tau = 100$ at inference, and uMBSA-NN* has $\tau = 20$ at inference.

	CoraFull			Pubmed			Reddit		
	Time	Mem.	Acc.	Time	Mem.	Acc.	Time	Mem.	Acc.
APPNP	7.2	2.1	57.9	4.3	1.9	**79.0**	-	OOM	-
AGP	63.5	2.4	59.9	48.6	3.7	73.0	68.4	10.3	13.8
PPRGO‡	14.1	0.6	59.3	1.7	0.4	75.3	14.1	1.5	22.3
PPRGO*	4.2	0.5	56.0	1.0	0.4	73.7	12.7	1.5	18.6
MBSA-NN	2.1	0.04	**62.4**	0.9	0.01	77.3	2.5	0.8	**35.9**
uMBSA-NN‡	2.1	0.02	61.1	0.9	0.01	75.7	2.2	0.6	33.7
uMBSA-NN*	**1.9**	**0.02**	60.3	**0.7**	**0.01**	75.5	**1.8**	**0.6**	32.3

the same computation complexity (Time (s.)) through all datasets by batch training ($bs * 2$) and batch inference (3.4). At the same time, PPRGO uses $bs * k$ nodes in training, $k = 32$ being the number of neighbour nodes for each labelled node, and the complete graph with n nodes in inference due to the use of PowerIteration. In particular, Table 3.4 shows that PPRGO with the same batch size $k = 2$ as uMBSA-NN loses accuracy for all datasets and significantly consumes more time on CoraFull and Reddit with respect to uMBSA-NN ($\tau = 100, \tau = 20$). Moreover, we can see that uMBSA-NN outperforms PPRGO even with $k = 32$ in terms of time, memory, and accuracy, significantly noticeably on CoraFull and Reddit datasets. We conjecture that uMBSA-NN as well as MBSA-NN gains accuracy because uniform batch sampling allows considering all neighbors for labelled nodes during training. On the contrary, the training process of PPRGO is based only on the top k PPR neighbor nodes.

Note that uMBSA-NN maintains competitive accuracy compared to the other most recent best scaling algorithms while only outperforming them in terms of time and memory complexity. The accuracy reduction occurs in uMBSA-NN as a result of a uniform selection of unlabelled nodes for updating labelled ones, which does not ensure that the unlabelled nodes will have a large number of connections with labelled ones. On the other hand, uMBSA-NN can beat MBSA-NN in terms of memory and time consumption across all datasets because of the uniform batch sampling. Finally, given that accuracy remains competitive with the other scaling algorithms, we can observe that uMBSA-NN presents an accuracy vs. time/memory complexity tradeoff.

The implementation of uMBSA-NN on Python3.8 (Tensorflow2.0), links on datasets in experiments with parameters details for algorithms in Table 3.4 and description of the

technical environment are in Section 3.11.

3.10 Proofs

3.10.1 Theorem 1

Proof. Let us introduce an Ordinary Differential Equation (ODE) for analysis of Algorithm 3, viz., $\dot{Z}_i = h(Z_i^t)$ where

$$h(Z_i^t) = \mathbb{E}\left[\frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}}\left(\frac{\alpha}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}\sum_{j\in\mathcal{S}_{\mathcal{A}_{t+1}}}\widetilde{A}_{i,j}Z_j^t - Z_i^t + \widetilde{Y}_i\right)\middle| i\in\mathcal{S}_{\mathcal{A}_t}\right].$$

Let $\widetilde{Y}_i = (1-\alpha)Y_i$. Then

$$\begin{aligned} h(Z_i^t) &= I\{i\in\mathcal{S}_{\mathcal{A}_t}\}\Bigg(\sum_{\mathcal{A}_{t+1}\in\mathcal{A}} Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}\frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}}\times \\ &\qquad \left(\frac{\alpha\sum_{\{j\in\mathcal{S}_{\mathcal{A}_{t+1}}\}}\alpha\widetilde{A}_{i,j}Z_j^t}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}} - Z_i^t + \widetilde{Y}_i\right)\Bigg) \\ &= I\{i\in\mathcal{S}_{\mathcal{A}_t}\}\left(\sum_{j=1}^{n}\alpha\widetilde{A}_{i,j}Z_j^t - Z_i^t + \widetilde{Y}_i\right). \end{aligned}$$

Since $\sqcup_{k=1}^{s}\mathcal{S}_k = [1,...,n]$, above transforms to:

$$h(Z^t) = (\alpha\widetilde{A} - I)Z^t + (1-\alpha)Y. \tag{3.7}$$

$M(i)^{t+1}$ is a martingale difference sequence uncorrelated with the past and can be considered as noise:

$$M(i)^{t+1} = \frac{\alpha\sum_{j\in\mathcal{S}_{\mathcal{A}_{t+1}}}\widetilde{A}_{i,j}Z_j^t}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}} - \sum_{j=1}^{n}\alpha\widetilde{A}_{i,j}Z_j^t. \tag{3.8}$$

Rewrite Algorithm 1 as a stochastic approximation algorithm:

$$Z_i^{t+1} = Z_i^t + \eta(t)I\{i\in\mathcal{S}_{\mathcal{A}_t}\}\left(\sum_{j=1}^{n}\alpha\widetilde{A}_{i,j}Z_j^t - Z_i^t + \widetilde{Y}_i + M(i)^{t+1}\right). \tag{3.9}$$

Then, following the results from Corollary 4 and Theorem 7 in Chapters 2 and 3 resp. of [82] we can prove Theorem 1 by showing the fulfillment of the corresponding

assumptions from [82]:

1. As required in [82], Z^* is a globally asymptotically stable equilibrium of the above ODE, because this is a stable linear system. If Z^* is a solution of the Eq. (3.1) then $Z^* = BZ^* + \widetilde{Y}$ where $B = \alpha\widetilde{A}$ and $\widetilde{Y} = (1-\alpha)Y$. Also, if $h(Z^*) = 0$ then Z^* is a globally asymptotically stable equilibrium point of (3.1).

2. **A1**, in Chapter 2 of [82]: The map $h(x) : \mathcal{R}^n \to \mathcal{R}^n$ is Lipschitz for some $0 < \lambda < \infty$:

Let $\Gamma = diag(\pi_1, \ldots, \pi_n)$ with $\pi = [\pi_1, \ldots, \pi_i, \ldots, \pi_n]$ denote the stationary probability vector on diagonal where $\pi_i = \frac{\psi(i,t)}{t}, \psi(i,t) = \sum_{m=0}^{t}\{i \in \mathcal{S}_{\mathcal{A}_m}\}$. The limiting ODE (3.7) can be derived as:

$$\begin{aligned}\dot{x}(t) &= \Gamma(h(x(t))) = \Gamma((\alpha\widetilde{A}x(t) - x(t) + (1-\alpha)Y)) \\ &= (F(x(t)) - x(t)) = F_\Gamma(x(t)) - x(t),\end{aligned}$$

where $F_\Gamma(x) = (I - \Gamma)x + \Gamma F(x)$. Then,

$$\begin{aligned}&||F_\Gamma(x) - F_\Gamma(y)||_w \leq \max_i \Bigg[(1-\pi_i)\left|\frac{(x_i - y_i)}{w_i}\right| \\ &+ \pi_i\left|\frac{\sum_{j=1}^{n}\alpha\widetilde{A}_{i,j}(x_j - y_j)}{w_i}\right|\Bigg] \leq \max_i \Bigg[(1-\pi_i)\left|\frac{x_i - y_i}{w_i}\right| \\ &+ \pi_i\lambda||x-y||_w\Bigg] \leq \hat{\lambda}||x-y||_w,\end{aligned}$$

where $\hat{\lambda} = \max_i(1-(1-\lambda)\pi_i)$. Thus F_Γ is also a contraction w.r.t. $||\cdot||_w$, hence Lipschitz.

3. **A2** in Chapter 2 of [82]: Step sizes $\{\eta(t)\}$ are positive scalars satisfying $\sum_{t=0}^{\infty}\eta(t) = \infty$ and $\sum_{t=0}^{\infty}\eta(t)^2 < \infty$.

This holds due to our assumption about $\eta(t) = 1/(1+t)^\gamma;\ \gamma \in (0,1]$. Based on latest results from Theorem 1.1 and Theorem 1.2 in [89] the $\sum_{t=0}^{\infty}\eta(t)^2 < \infty$ can be replaced by $\lim_{t\to\infty}\eta(t) = 0$ and Z_i^t will still converges almost surely to Z_i^*.

4. **A3** in Chapter 2 of [82]: $\mathbb{E}[M(i)^{t+1}] = 0$ and $\mathbb{E}[||M(i)^{t+1}||^2] \leq K(1 + ||Z_i^t||^2)$ for some constant $K > 0$.

Based on results from (3.8),

$$\mathbb{E}[M(i)^{t+1}|\mathcal{A}_s, s \leq t] = \sum_{j=1}^{n}\alpha\widetilde{A}_{i,j}Z_j^t - \sum_{j=1}^{n}\alpha\widetilde{A}_{i,j}Z_j^t = 0,$$

i.e., $M(i)^t$ is a martingale difference sequence. Also:

$$|M(i)^{t+1}| \leq 1 + \left| \sum_{j=1}^{n} \alpha \widetilde{A}_{i,j} Z_j^t \right|.$$

Then: $\mathbb{E}[||M(i)^{t+1}||^2] \leq K(1 + ||Z_i^t||^2)$. Thus the assumption holds.

5. **A5** in Chapter 3 of [82]: Scaled limit $h_c(x) = \frac{h(cx)}{c}$, $x \in \mathcal{R}^n$ exists and satisfies $h_c(x) \to h_\infty(x)$ as $c \to \infty$, uniformly on compacts for some $h_\infty \in C(\mathcal{R}^n)$. Furthermore, the O.D.E. $\dot{x}(t) = h_\infty(x(t))$ has the origin as it is unique globally asymptotically stable equilibrium.

To ensure that this condition applies in our case let us consider the case for one element:

$$\lim_{c\to\infty} \left(\frac{\sum_{i=1, i\neq j}^{n} \alpha \widetilde{A}_{i,j} Z_j^t + c\alpha \widetilde{A}_{i,i} Z_i^t - c Z_i^t + \widetilde{Y}_i}{c} \right)$$
$$= \lim_{c\to\infty} \left(\frac{c\alpha \widetilde{A}_{i,i}^t Z_i^t}{c} - \frac{c Z_i^t}{c} \right) = \alpha \widetilde{A}_{i,i}^t Z_i^t - Z_i^t.$$

Thus $h_c(x) \to h_\infty(x)$ for suitably defined h_∞ uniformly on compacts. Furthermore, the limiting O.D.E. is a homogeneous linear system with a nonsingular coefficient matrix, so has the origin as the unique globally asymptotically stable equilibrium. Hence the assumption holds.

Since we fulfill all required assumptions, it follows from the results of Chapter 2, [82] that Z^t generated by Algorithm 3 converges to Z^* as $t \to \infty$ with probability one. □

3.10.2 Theorem 2

Proof. The proof relies on comparison of two ordinary differential equations. The first ODE from Algorithm 5 is $\dot{Z}_i = h(Z_i^t)$, where

$$h(Z_i^t) = \mathbb{E}\left[\frac{\widetilde{P}_{\mathcal{A}_t, \mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t, \mathcal{A}_{t+1}}} \left(\frac{1}{P_{\mathcal{A}_t, \mathcal{A}_{t+1}}} \sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} \widetilde{A}_{i,j} (\Gamma^{(0)}(X_j W_{(0)}^t) W_{(1)}^t) - Z_i^t \right) | i \in \mathcal{S}_{\mathcal{A}_t} \right]. \quad (3.10)$$

Thus

$$\begin{aligned}\dot{Z}_i^t = h(Z_i^t) = I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\Bigg(\sum_{\mathcal{A}_{t+1}\in\mathcal{A}} Q_{\mathcal{A}_t,\mathcal{A}_{t+1}} \frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}} \times \\ \left(\frac{\sum_{\{j\in\mathcal{S}_{\mathcal{A}_{t+1}}\}} \widetilde{A}_{i,j}(\Gamma^{(0)}(X_j W_{(0)}^t)W_{(1)}^t)}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}} - Z_i^t\right)\Bigg) \\ = I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\left(\sum_{j=1}^{n} \widetilde{A}_{i,j}(\Gamma^{(0)}(X_j W_{(0)}^t)W_{(1)}^t) - Z_i^t\right).\end{aligned}$$

Since $\sqcup_{i=k}^{s}\mathcal{S}_k = [1, ..., n]$ then the above ODE transforms to:

$$\dot{Z}^t = h(Z^t) = \widetilde{A}(\Gamma^0(XW_{(0)}^t)W_{(1)}^t) - Z^t. \tag{3.11}$$

$M(i)^{t+1}$ is a martingale difference sequence uncorrelated with the past and can be considered as noise:

$$M(i)^{t+1} = \frac{\sum_{j\in\mathcal{S}_{\mathcal{A}_{t+1}}} \widetilde{A}_{i,j}(\Gamma^{(0)}(X_j W_{(0)}^t)W_{(1)}^t)}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}} - \sum_{j=1}^{n} \widetilde{A}_{i,j}(\Gamma^{(0)}(X_j W_{(0)}^t)W_{(1)}^t). \tag{3.12}$$

Rewrite Algorithm 5 in a form of stochastic approximation algorithm where $F_i(W_{(0)}^t, W_{(1)}^t) = \sum_{j=1}^{n} \widetilde{A}_{i,j}(\Gamma^{(0)}(X_j W_{(0)}^t)W_{(1)}^t)$:

$$Z_i^{t+1} = Z_i^t + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\left(F_i(W_{(0)}^t, W_{(1)}^t) - Z_i^t + M(i)^{t+1}\right) \tag{3.13}$$

Let us define the second stochastic approximation algorithm as follows:
Denote $\lambda_i^t = F_i(W_{(0)}^t, W_{(1)}^t)$. $W_{(0)}^{t+1}, W_{(1)}^{t+1}$ can be calculated by gradient descent. Note that we choose the gradient descent method just for simplicity of explanation in proof,i.e.

$$(W_{(0)}^{t+1}, W_{(1)}^{t+1}) = (W_{(0)}^t, W_{(1)}^t) - \beta\nabla\mathcal{L}(Y_i, \Gamma^{(1)}(Z_i^{t+1})))$$

Then, we have

$$\begin{aligned}\lambda_i^{t+1} &= \lambda_i^t - \beta\nabla\mathcal{L}(Y_i, \Gamma^{(1)} Z_i^{t+1}) \\ &= \lambda_i^t + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\left(\lambda_i^t - \lambda_i^t - \frac{\beta}{\eta(t)}\nabla\mathcal{L}(Y_i, \Gamma^{(1)}(Z_i^{t+1})) + o(1)\right).\end{aligned} \tag{3.14}$$

Let us compute the difference between Z_i^{t+1} and λ_i^{t+1}:

$$
\begin{aligned}
Z_i^{t+1} - \lambda_i^{t+1} &= Z_i^t - \lambda_i^t \\
&\quad + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \Bigg(F_i(W_{(0)}^t, W_{(1)}^t) - Z_i^t - \lambda_i^t + \lambda_i^t \\
&\quad + M(i)^{t+1} + \frac{\beta}{\eta(t)} \nabla \mathcal{L}(Y_i, \Gamma^{(1)}(Z_i^{t+1})) + o(1) \Bigg).
\end{aligned} \tag{3.15}
$$

Since $\lambda_i^t = F_i(W_{(0)}^t, W_{(1)}^t)$, we have

$$
\begin{aligned}
Z_i^{t+1} - \lambda_i^{t+1} &= Z_i^t - \lambda_i^t + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \Bigg(-(Z_i^t - \lambda_i^t) \\
&\quad + M(i)^{t+1} + \frac{\beta}{\eta(t)} \nabla \mathcal{L}(Y_i, \Gamma^{(1)}(Z_i^{t+1})) + o(1) \Bigg).
\end{aligned} \tag{3.16}
$$

Denote $y_i^t = Z_i^t - \lambda_i^t$. Then,

$$
\begin{aligned}
y_i^{t+1} &= y_i^t + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \left(-y_i^t + M(i)^{t+1} + \mathcal{O}\left(\frac{\beta}{\eta(t)} \right) \right) \\
&= (1 - \eta(t)) y_i^t + \eta(t) \mathcal{O}\left(\frac{\beta}{\eta(t)} \right).
\end{aligned} \tag{3.17}
$$

If we choose $\beta = \beta(t)$ satisfying $\frac{\beta(t)}{\eta(t)} \to 0$, the updates of $W_{(0)}^t, W_{(1)}^t$ happen on a slower time scale. Hence we freeze them at a constant value $W_{(0)}, W_{(1)}$, for the purposes of analyzing the iteration (3.17). The limiting ODE for (3.17) then is $\dot{y}(t) = -y(t)$, which has the zero vector as its unique globally asymptotically stable equilibrium. The analysis of two time scale stochastic approximation of section 6.1 of [82] applies and leads to $Z_i^t - \lambda_i^t = Z_i^t - F_i(W_{(0)}^t, W_{(1)}^t) \to 0$ a.s. In turn the analysis of *ibid.* for the slow timescale iterate for $W_{(0)}^t, W_{(1)}^t$ implies that it is nothing but a gradient descent for the loss function

$$
E\left[\mathcal{L}(QY, QF(W_{(0)}^t, W_{(1)}^t)) \right]
$$

w.r.t. $W_{(0)}, W_{(1)}$where $Q = diag(Q_{i,i})_{i=1}^n$ is a diagonal matrix with strictly positive entries on the diagonal, that arises out of sampling. Thus it will converge a.s. to a local minimum of the loss function, i.e., a locally optimal $W_{(0)}^*, W_{(1)}^*$. In turn, $Z_i^t - \lambda_i^t \to 0$ a.s. implies

that

$$Z_i^t \xrightarrow{a.s.} \sum_{j=1}^{n} \widetilde{A}_{i,j}(\Gamma^{(0)}(X_j W_{(0)}^*)W_{(1)}^*) \quad as\ t \to \infty\ \forall i \in \{1,\dots,n\},$$

□

3.10.3 Remark 4

Proof. Since, for Remark 4 the ODE corresponding to (3.6) is:

$$\begin{aligned} h(Z_i^t) &= \mathbb{E}\left[\left(\sum_{j\in\mathcal{S}_{\mathcal{A}_{t+1}}} \frac{\sum_{j=1}^{n} \alpha\widetilde{A}_{i,j}Z_j^t}{\frac{2*bs}{n}} - Z_i^t + \widetilde{Y}_i\right)|i \in \mathcal{S}_{\mathcal{A}_t}\right] \\ &= I\{i \in \mathcal{S}_{\mathcal{A}_t}\}Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}C_{n-1}^{2*bs-1}\left(\frac{\sum_{j=1}^{n} \alpha\widetilde{A}_{i,j}Z_j^t}{\frac{2*bs}{n}}\right) - Z_i^t + \widetilde{Y}_i \\ &= I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\sum_{j=1}^{n} \alpha\widetilde{A}_{i,j}Z_j^t - Z_i^t + \widetilde{Y}_i \end{aligned}$$

and martingale:

$$M(i)^{t+1} = \frac{\sum_{j=1}^{n} \alpha\widetilde{A}_{i,j}Z_j^t}{\frac{2*bs}{n}} - \sum_{j=1}^{n} \alpha\widetilde{A}_{i,j}Z_j^t$$

Then, it is easy to see that all assumptions from proof of Theorem 1 (Subsection 1) also fulfill for (3.6) and Z^t converges to Z^* when $t \to \infty$ with probability one. □

3.11 Experimental details

3.11.1 State-of-the-art (SOTA) algorithms

We compare MBSA (Algorithm 3) with other updating strategies from the literature, namely Jacobian over Relaxation (JOR), Double Stochastic Block Gauss-Seidel (DSBGS) [65], Randomized Kaczmarz (RK) [1] and Randomized Block Gauss-Seidel (RBGS) [66].Note that we excluded DSA-SSL [2] from the comparison due to its extremely slow convergence rate.

To show the advantages of MBSA-NN (Algorithm 5), MBSA-APPNP (Eq. (3.5)) and uMBSA-NN(Algorithm 7) in terms of computation time, memory complexity and accuracy, we consider the latest best SOTA scaling algorithms such as: *neighbor selection* PPRGO [51], *model simplification* AGP [49], *subgraph sampling* Graph-Saint [56], *layer sampling* GNNAutoScale (GAS) [62]. Note that all of the aforementioned algorithms work in a batch regime and outperform other scaling algorithms such as SGC, clusterGCN,

GBP, and FastGCN [60], on extremely large graphs. Also, we compare MBSA-APPNP with APPNP to see the impact of batch training on the scaling of this algorithm.

3.11.2 Parameters

Note that for the comparison with MBSA, we speed up the implementation of RBGS algorithm (see details in Subsection 3.11.5). We compute MBSA with $\gamma = 0.3$ and $\epsilon = 0.1$, based on an initial study of the impact of the parameters in Section 3.3. We have used the optimal step size 0.9 and 15 for JOR and DSBGS, respectively, from their works. For a fair comparison between DSBGS, RBGS and MBSA, we set the batch size to the same value $bs = 512$. Moreover, for all of these algorithms, we used $\alpha = 0.9$ since it often gives a high classification accuracy [14].

For achieving consistency between APPNP and MBSA-NN, we took two layers architecture for neural network and a random jump $\alpha = 0.9$. Moreover, for MBSA-NN we used the same batch size $bs = 512$ as in PPRGO. Also, for MBSA-NN we defined: L2 regularization to $5 \cdot 10^{-4}$, power of step size for training and inference $\gamma = 0.3$, optimal batch size for inference for each dataset computed by equation (3.4), damping factor for inference $\epsilon = 0.1$ and learning rate 0.005. The values for γ, ϵ and τ have been selected the same as the above optimal parameters for comparison with MBSA. Moreover, for pMBSA we used 6 cores for parallel computation. The rest best hyper-parameters for MBSA-NN in terms of accuracy (Tables 3.2,3.4) we have selected by the 5 fold cross-validation grid search for each dataset separately. These hyper-parameters were selected from the following range:

- Training: activation function $\Gamma^{(0)}(\cdot) \in \{relu, selu, leaky\_relu\}$, dropout $\in \{0.1, 0.5\}$, $\beta \in \{1, \frac{1}{Q_{\mathcal{A}_t, \mathcal{A}_{t+1}}}\}$, $d \in \{32, 64, 128, 256, 512\}$;
- Inference: MBSA $\tau \in \{100, 300\}$, pMBSA $\tau \in \{30, 90\}$.

.

Table 3.5 shows the selected MBSA-NN hyper-parameter for each dataset, which was used further for computations in Tables 3.2,3.4.

Moreover, the results in Tables 3.2,3.4 for PPRGO, AGP, GAS, APPNP and GraphSaint were achieved by computing these algorithms with the best hyper-parameters defined in their works. Note that in PPRGO-MBSA for the MBSA at the inference, we choose parameters as for inference in MBSA-NN (Table 3.5). Also, take notice that the hyper-parameters we utilized for uMBSA-NN and MBSA-APPNP (Tables 3.2,3.4) were the same as the best for the MBSA-NN. Finally, for consistency of training process across

Table 3.5: Optimal MBSA-NN hyper-parameters in terms of Accuracy (Tables 3.23.4).

Parameters	**Cora***	**Pubmed**	**Reddit**	**Yelp**	**OGBN***	**MAG***
$\Gamma^{(0)}(\cdot)$	leaky_relu	leaky_relu	selu	relu	relu	relu
dropout	0.1	0.1	0.1	0.5	0.5	0.5
β	1	1	$\frac{1}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}}$	1	1	1
d	128	512	512	128	512	32
MBSA τ	100	100	100	300	300	300
pMBSA τ	30	30	30	90	90	90

MBSA-NN, PPRGO, APPNP, AGP, GAS and Graph-Saint on all datasets, we did not use early stopping and computed only validation loss at each epoch during 200 *epochs*.

Table 3.6: Optimal MBSA-NN hyper-parameters in terms of memory and computational complexity (Table 3.3).

min	**Cora***	**Pubmed**	**OGBN***	**MAG***
Algorithm	Graph-Saint	AGP	PPRGO	PPRGO
Accuracy (%)	58.7 ± 0.58	73.0 ± 2.88	35.5 ± 1.04	71.0 ± 0.86
epochs	38 ± 3.84	17 ± 4.72	148 ± 1.61	197 ± 2.14
MBSA τ	10 ± 2.31	4 ± 2.91	82 ± 1.93	98 ± 3.03
MBSA(p) τ	13 ± 1.51	3 ± 1.77	23 ± 2.11	28 ± 1.84

For a fair comparison in terms of memory and computational complexity, we retained the minimum average accuracy obtained for each dataset from Table 3.2. We repeated 10 times the training of each algorithm for each dataset until they achieved this minimal accuracy and stored the quantities of interest at that point. Table 3.6 shows the average minimum number of epochs and iterations for training and inference of MBSA-NN for achieving a minimal accuracy over datasets presented (Table 3.2). In particular, Table 3.6 shows the parameters for MBSA-NN required to convergence to the minimal accuracy for each dataset (Table 3.2) with extremely small consumption of memory and computational complexity (Table 3.3).

3.11.3 Technical environment and links on implementations

The technical environment for our experiments is presented in Table 3.7. In particular, Table 3.7 shows the difference in technical requirements between MBSA-NN and the latest best SOTA scaling algorithms. The Table 3.7 shows that MBSA-NN requires almost **two** times fewer resources everywhere. Moreover, note that the implementations

of the algorithms for experiments are available by the links provided in Table 3.8.

Table 3.7: Environment of latest SOTA scaling algorithms.

Algorithm	GPU Nvidia	CPU Intel	RAM
PPRGO [51]	1080Ti: 11 GB	5 cores	64 GB
AGP [49]	RTX8000: 48 GB	Xeon: 40 cores	512 GB
Graph-Saint [56]	Tesla P100: 16 GB	Dual Xeon 40 cores	512GB
GAS [62]	2080Ti Tesla P100: 11 GB	—	64 GB
MBSA-NN	1070: 7 GB	7 cores	32 GB

Table 3.8: Implementation links.

Methos	URL
APPNP	https://github.com/gasteigerjo/ppnp
PPRGO	https://github.com/TUM-DAML/pprgo_tensorflow
Graph-Saint	https://github.com/GraphSAINT/GraphSAINT
AGP	https://github.com/wanghzccls/AGP-Approximate_Graph_Propagation
GAS	https://github.com/rusty1s/pyg_autoscale

Finally, implementations of MBSA-NN (Tenwsorflow v2), MBSA (Python 3.8), pMBSA (C++) and other linear algorithms are available on google drive link[4]. The implementation of uMBSA-NN (Tensorflow v2) is available on the github link[5].

3.11.4 Dataset description

For each dataset, we extracted large connected components as in [6,14], which avoids cases with unconnected batches. We treated the citation links of these datasets as undirected edges. We regularizes the adjacency matrices with $\delta = 0.5$ since it has been shown to give high performance [6]. The feature matrix X has been generated for each dataset by a bag-of-words [73] model. Moreover, X is used without normalization and it is sparse for CoraFull, Pubmed and MAG-coarse, and dense for the others. We used 20 labeled nodes from each class as a train set. The validation sets for the experiments with MBSA-NN contains twice as many nodes as in the training set, sampled uniformly (without repetition), while for the experiments with MBSA, we just used the remaining nodes as test set. Finally, the experiments are repeated with 5 random splits for each dataset to show stable results. Table 3.9 presents the dataset statistics, where the notations Cora*,

[4]https://drive.google.com/drive/folders/1i3tcV9zHtH20tAs3CCCWXdqeFA6kfcou?usp=sharing
[5]https://github.com/KamalovMikhail/wsdm2022

OGBN* and MAG* are used to improve readability in place of CoraFull, OGBN-products and MAG-coarse datasets respectively.

Table 3.9: Datasets statistic (large connected component).

Dataset	n	e	d	c
Cora-ML [90]	2,810	15,962	-	7
Citeseer [91]	2,110	7,388	-	6
*Cora** [85]	18,800	125,370	8,710	70
Pubmed [85]	19,717	88,648	500	3
Reddit [87]	232,965	114,615,892	602	41
Yelp [56]	703,655	13,927,667	300	21
*OGBN** [88]	2,385,902	123,612,734	100	47
*MAG** [51]	10,541,560	265,219,994	2,784,240	8

The datasets in the experimental sections are available below links:

Cora* [85], Pubmed [85], Reddit [87] :
https://github.com/TUM-DAML/pprgo_tensorflow/tree/master/data

Yelp [56], OGBN* [88] :
https://drive.google.com/drive/folders/1zycmmDES39zVlbVCYs88JTJ1Wm5FbfLz

MAG* [51] :
https://figshare.com/articles/dataset/mag_scholar/12696653

Cora-ML [90], Citeseer [91]:
https://github.com/gasteigerjo/ppnp/tree/master/ppnp/data

3.11.5 Implementation details

All linear algorithms have been implemented in a sparse regime to reduce memory consumption. Moreover, we implemented the node partitioning method for BSA, RBGS, and DSBGS, making the implementation of these algorithms faster than with application of default numpy partitioner. Furthermore, for RBGS we employed the following equations:

$$A_\tau Z' = r; Z^{t+1} = Z^t + EZ'^T \tag{3.18}$$

instead of:

$$Z^{t+1} = Z^t + EA_\tau^\dagger r \tag{3.19}$$

where $A_\tau^\dagger$ is a pseudo inverse matrix. Note that E and A_τ are sparse and first equation in (3.18) can be solved for sparse matrices which significantly reduce memory and time

consumption than in the case of $A_\tau^\dagger$ (e.g. for 2 iterations (3.18) consumes 2.3 MiB, and (3.19) consumes 120 MiB). The detail notation can be found in [[66], p.370].

Chapter 4

Graph-Diffusion & PCA framework

This chapter presents a novel framework called Graph diffusion & PCA (GDPCA) is proposed in the context of semi-supervised learning on graph structured data. By a combination of a modified principal component analysis with the traditional supervised loss, Laplacian regularization in GDPCA, the *Curse of Dimensionality* is avoided and the scenario where the adjacency matrix represented via *Binary edges* is handled. This framework can be applied to non-graph datasets as well, such as images by constructing similarity graph. GDPCA enhances the local graph structure through node covariance, which improves node classification. Furthermore, the proposed combination of Laplacian regularization and a reorganized PCA loss is guaranteed to have an explicit solution by our framework. Additionally, we demonstrate that, on a variety of datasets, GDPCA beats the most recent best state-of-the-art (SOTA) classical diffusion based methods. Even more, we show that our framework performs similarly to SOTA graph convolution networks while having a much lower computational complexity.

The following sections make up this chapter: i) the Graph-Diffusion & PCA (GDPCA) framework is initially defined with a theoretical study. Particularly, we show the theoretical analysis of GDPCA, which claims that our framework explicitly resolves the Laplacian PCA loss. Additionally, this section illustrates how the MBSA algorithm scales the GDPCA framework (see Chapter 3); ii) GDPCA's ablation studies and experimental comparisons with the latest best SOTA classical diffusion-based algorithms and graph convolution networks are shown in the next section. Additionally, this section demonstrates how GDPCA performs better on both graph and non-graph data than traditional diffusion-based and non-graph based SSL methods. Furthermore, we demonstrate that GDPCA is highly accurate even when applied to the actual dataset including the COVID Clinical Trials. iii) as a last step, we demonstrate GDPCA's propositional proofs and provide experimental information. We also go into depth about how we crawled, collected,

and processed the present COVID Clinical Trials data for our studies. Moreover, this section provides information on the datasets, algorithms, and technological environment that were used.

4.1 Graph-diffusion with reorganized PCA loss

Let's first establish the notation that will be required for the GDPCA framework's further explanation. In graph-based SSL, the data consists of the feature matrix $X = [X_i]_{i=1}^n$, where $X_i = (X_{i,j})_{j=1}^d$ lies in a d-dimensional feature space (e.g. from bag-of-words [73]), and of the label matrix $Y = [Y_{i,j}]_{i,j=1}^{n,k}$ such that $Y_{i,j} = 1$ if $X_i \in \mathcal{C}_j$ and $Y_{i,j} = 0$ otherwise, $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ being a set of k classes. The aim of semi-supervised learning is to estimate Y by a classification result $Z = [Z_{i,j}]_{i,j=1}^{n,k}$ when there is a low number of labels available, while X contains information for both labeled and unlabeled observations. We also assume that the dataset (X, Y) can be represented through the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $n = |\mathcal{V}|$ the number of nodes with features (e.g. papers) and $e = |\mathcal{E}|$ is the number of edges (e.g. citations). Let $A = [A_{i,j}]_{i,j=1}^{n,n}$ denote the adjacency matrix associated with the $\mathcal{G}$, and $D = \text{diag}(D_{i,i})$ be a diagonal matrix with $D_{i,i} = \sum_{j=1}^n A_{i,j}$.

Since most of the classical diffusion-based algorithms (see Chapter 1) focus on minimizing the combination of Laplacian and standard classification losses (Loss function 4.1). This section proposes a novel framework that minimizes the losses mentioned below from the point of resolving issues like *Binary edges* and *Curse of dimensionality* (see Chapter 1).

$$\min_{Z \in \mathbb{R}^{n \times k}} \Big\{ \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} ||Z_i - Z_j||_2^2 + \mu \sum_{i=1}^{n} ||Z_i - Y_i||_2^2 \Big\}, \tag{4.1}$$

where μ is a Lagrangian multiplier, n is the number of nodes, $A = [A_{i,j}]_{i,j=1}^{n,n}$ is an adjacency matrix, $Z = [Z_i]_{i=1}^n$ is a classification result and $Y = [Y_i]_{i=1}^n$ is a matrix that represents labels.

This work's current point is the opposite of the point of the latest works [4,23,25] that focus on increasing accuracy. Particularly, this work is motivated by the idea that principal component analysis (PCA) can solve at least the *Curse of dimensionality* issue. Different works [92–96] consider a transformation of the matrix of node features $X \in \mathcal{R}^{n \times d}$ by principal components $XU^T = Z$ to the classification results, where $U \in \mathbb{R}^{d \times k}$ is a matrix of principal component vectors from PCA. Instead, we consider principal components which are straightforwardly related to the classification result ($U \in \mathbb{R}^{k \times n}$, $U^T = Z$), as explained in the sequel.

One of the main ideas of this work is that the nodes from different classes have high covariance. This idea lies under the hood of Linear Discriminant Analysis (LDA) [97], which was developed for supervised learning. We extend this idea so that it can also be applied in both unsupervised (PCA-BC) and semi-supervised learning (GDPCA).

4.1.1 PCA for binary clustering (PCA-BC)

In this section, we restrict the setting to the case where no labels are available, and where the nodes come from two clusters.

Assumption 1: Let us assume that the feature matrix X is sampled from the Gaussian distribution:

$$X_1, \dots, X_{\frac{n}{2}} \sim \mathcal{N}(\mu_1, C) \; and \; X_{\frac{n}{2}+1}, \dots, X_n \sim \mathcal{N}(\mu_2, C), \tag{4.2}$$

where C is the covariance matrix and μ_1, μ_2 are the expectations of classes $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. Furthermore, let $||C||_2 = O(1)$, $||\mu_1 - \mu_2||_2 = O(1)$ and expectations has an identical values through the all coordinates (e.g. $\mu_1 = [0.1, 0.1, 0.1, ...]$, $\mu_2 = [0.8, 0.8, 0.8, ...]$). Moreover, we have the ratio $c_0 = n/d$ be bounded away from zero for large d.

Remark 5: The assumptions $||C||_2 = O(1)$ and $||\mu_1 - \mu_2||_2 = O(1)$ are needed to save the essential variations in d linearly independent directions and define a non-trivial classification case for extremely large d. In particular, this assumption allows us to work with bag-of-words [73] where the d-space is increasing with respect to the number and the length of papers, which leads to the *Curse of dimensionality* issue.

Based on the proof of Theorem 2.2 in [98] and the above restrictions on X, there exists a connection between the binary clustering problem and the PCA loss function objective given by:

$$\max_{U \in \mathbb{R}^{k \times n}} ||\bar{X} U^T||_2^2, \; s.\,t.\; U^T U = 1 \tag{4.3}$$

where $\bar{X} = [\bar{X}_i^T]_{i=1}^d \in \mathbb{R}^{d \times n}$ with $\bar{X}_i^T = X_i^T - \frac{1}{d}\sum_{j=1}^d X_j^T$; $U = [U_i]_{i=1}^k \in \mathbb{R}^{k \times n}$ is a matrix of principal component vectors. Moreover, $U_{i=1} = U_1 = (U_{1,j})_{j=1}^n$ is the direction of maximum variance, and it can be considered as clustering results in the following way: if $U_{1,j} \geqslant median(U_1)$ then $X_j \in \mathcal{C}_1$ otherwise $X_j \in \mathcal{C}_2$. Figure 4.1 illustrates the idea that the covariance between nodes from different classes is high. We further demonstrate the applicability of PCA on the binary clustering task with a small numerical experiment. We generated several synthetic datasets (4.2) with various ratios c_0 and fixed values for expectation ($\mu_1 = (0.5, \dots, 0.5)$; $\mu_2 = (0.1, \dots, 0.1)$;) and covariance matrix ($C = diag(0.1)$) with $\frac{n}{2}$ the number of nodes in each class: $n = 100$, $d = 1000$, $c_0 = 0.1$; $n = 1000$, $d = 100$, $c_0 = 10$. The code of these experiments is publicly available through a GitHub repository [1]. Figure 4.2 shows examples of how U_1 discriminates the

[1] https://github.com/KamalovMikhail/GDPCA

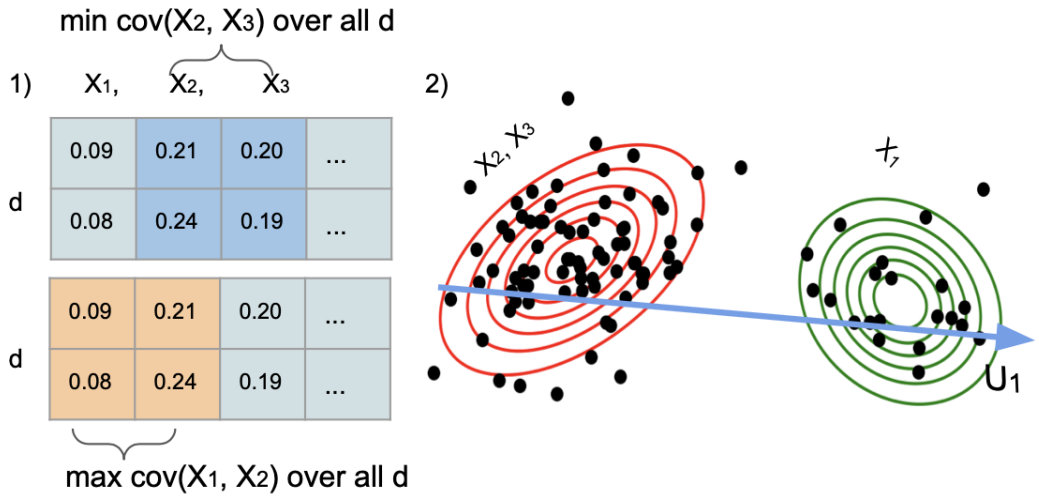

Figure 4.1: The intuition behind PCA-BC: 1) Transpose X and visualise the nodes with the maximum and minimum covariance ($cov(\cdot)$) in between; 2) Normalize transposed X and find the direction of maximum covariance by PCA.

two classes, even for large d-spaces.

4.1.2 Generalization of PCA-BC for GB-SSL

We propose to modify the Loss function (4.1) by adding the reorganized PCA loss (the minus sign being necessary to account for the maximization of the covariance between classes). The optimization problem thus consists in:

$$\min_{Z\in\mathbb{R}^{n\times k}} \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} A_{i,j}||D_{ii}^{\sigma-1}Z_i - D_{jj}^{\sigma-1}Z_j||_2^2 \right. \\ \left. +\mu\sum_{i=1}^{n} D_{ii}^{2\sigma-1}||Z_i - Y_i||_2^2 - 2\delta||\bar{X}Z||_2^2 \right\} \tag{4.4}$$

where δ is a penalty multiplier and σ is the parameter controlling the contribution of node degree. We control the contribution of a node degree through the diagonal matrix D to the power in Loss function (4.4) based on the work in [4].

4.1.3 Theoretical analysis

It should be noticed that in Loss function (4.4) we do not require the orthogonality condition $Z^TZ = 1$ as in Loss function (4.3). An interesting feature of Loss function (4.4)

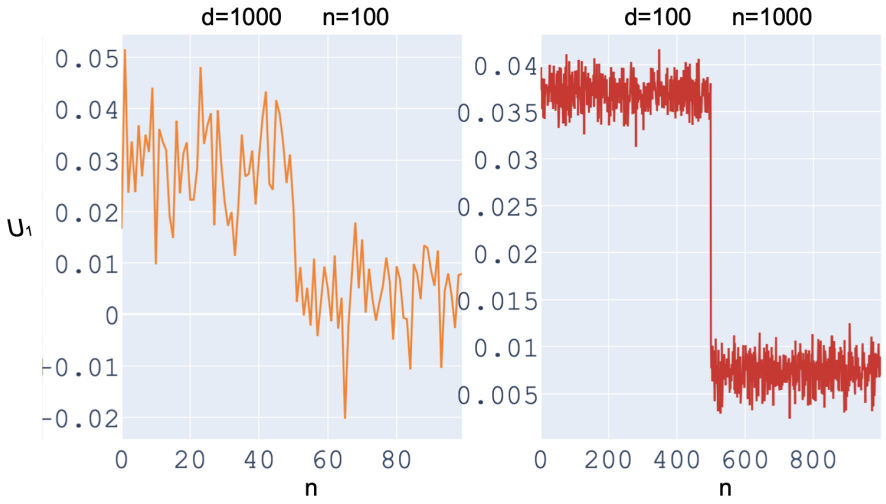

Figure 4.2: Mean value of U_1 (the direction of maximum variance in the PCA) on 100 sets of random synthetic data.

is that there exists an explicit solution given by the following proposition.

Proposition 1: When Loss function (4.4) is convex, the explicit solution is given by:

$$Z = \left(I - \alpha\left(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1}\right)\right)^{-1}(1-\alpha)Y, \tag{4.5}$$

where $\alpha = 2/(2+\mu)$, $I \in \mathbb{R}^{n\times n}$ is the identity matrix and $S = \frac{\bar{X}^T\bar{X}}{(d-1)} \in \mathbb{R}^{n\times n}$ is the sample covariance matrix (Figure 4.3).

Proof. The proof is based on the explicit solution of the first order optimization problem for the Loss function 4.4. The details of this proof are in Section 4.5 □

Remark 6: Proposition 1 provides the global minimum of Problem (4.4) in cases where it is convex, which occurs when the matrix
$I - \alpha\left(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1}\right)$ has positive eigenvalues (Theorem 1 in [99]). This condition can be achieved by values of δ such that the sum in brackets will not be upper then 1 and α always less than 1.

Direct matrix inversion in Equation (4.5) can be avoided thanks to efficient iterative methods such as the PowerIteration (PI) or the Generalized minimal residual (GMRES) [100] methods. PI consists in iterative matrix multiplications[2] and can be applied when the spectral radius verifies $\rho(\alpha(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1})) < 1$. GMRES consists in

[2] $Z = \alpha\left(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1}\right)Z + (1-\alpha)Y$

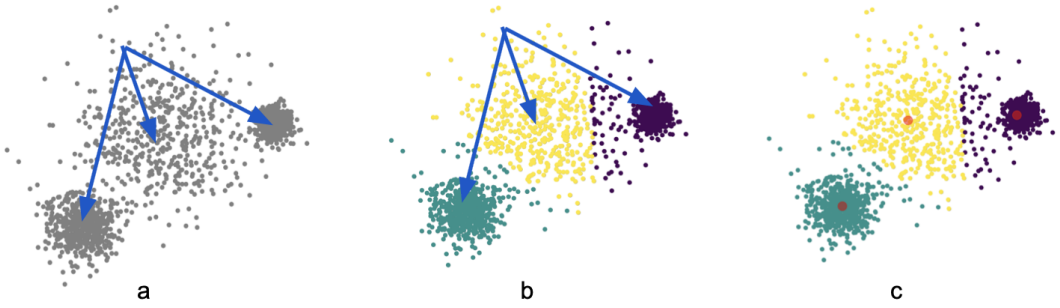

Figure 4.3: Classification steps of GDPCA based on solution (4.5): a - find the minimal k components, b - labeling objects by the information from U_i, c - find the centroids of classes by $\hat{X}U^T$

approximating the vectors' solution in Krylov subspace instead of explicit matrix inversion. In practice, PI is more convenient for the computation of Eq. (4.5) as it converges faster to the best classification accuracy and it can be computed in a distributed regime over nodes [101, p. 135]. The accuracy is computed by comparing maximum values per row between label matrix Y and classification results Z. Furthermore, instead of explicitly computing the spectral radius mentioned above, we can use the following proposition.

Proposition 2: Suppose that $SD^{-2\sigma+1}$ has only real eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. Then the inequality $\rho\left(\alpha\left(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1}\right)\right) < 1$ can be transformed into a simpler one:

$$1 + \delta\gamma < 1/\alpha \tag{4.6}$$

where γ is the maximum singular value of $SD^{-2\sigma+1}$ and δ is the penalty multiplier in Equation (4.5).

Proof. The proof is based on the properties of the sum of spectral radii. The details of this proof are in Section 4.5 □

Remark 7: In order to speed up the computation of singular values, we can use the randomized Singular Value Decomposition (SVD) [102]. Inequality (4.6) can then be rewritten as $1 + \delta(\gamma + \epsilon) < 1/\alpha$, where ϵ is the tolerance of the randomized SVD. The computational complexity of the randomized SVD is $C + O(n)$, where C is the cost of matrix-vector multiplications.

Algorithm 8: GDPCA (Graph diffusion & PCA)

INPUT: X, A, Y, σ, α, δ, $\mathcal{I}$, τ, ϵ;
INITIALIZE:
$\bar{X}_i^T = X_i^T - \frac{1}{d}\sum_j^d X_j^T \forall i \in (1, \ldots, n)$; $S = \frac{\bar{X}^T\bar{X}}{d-1}$
$\gamma = randomizedSVD(SD^{-2\sigma+1})$
IF: $1 + \delta(\gamma - \epsilon) < 1/\alpha$:
  $Z = PI(\alpha(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1}), (1-\alpha)Y, \mathcal{I})$
ELSE:
  $Z = GMRES((I - \alpha(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1})), (1-\alpha)Y, \tau, \mathcal{I})$

4.2 Graph-Diffusion & PCA (GDPCA)

Algorithm 8 gives the outline of our novel Graph diffusion & PCA (GDPCA) framework derived from Propositions 1 and 2. GDPCA uses the following setup: $\mathcal{I}$ is the number of iterations, τ is the tolerance in GMRES, δ is a Lagrangian multiplier, σ is the parameter controlling the contribution of node degree and ϵ is the tolerance in randomized SVD.

Note also that Proposition 1 simplifies to the known results of PRSSL [4] for the value $\delta = 0$. GDPCA can thus be seen as a generalization of PRSSL enriching the default random walk matrix $D^{\sigma-1}AD^{-\sigma}$ thanks to the sample covariance matrix S. Notice that S is retrieved from PCA loss in Loss function (4.4). This enrichment of the binary weights ($A_{i,j} = 0$ or $A_{i,j} = 1$) by node covariance allows bypassing the issue with *Binary edges*. Similarly, we assume that our framework solves the *Curse of dimensionality* issue thanks to the use of PCA loss. Moreover, note that Proposition 1 can provides several semi-supervised learning methods with various σ which can be used in GDPCA (Algorithm 8) such as:

1. if $\sigma = 0.5$ we got the Normalized Laplacian method with PCA regularization:

$$Z = \left(I - \alpha\left(D^{-0.5}AD^{-0.5} + \delta S\right)\right)^{-1}(1-\alpha)Y \tag{4.7}$$

2. if $\sigma = 1$, we got the random walk normalized Laplacian (PageRank) with PCA regularization:

$$Z = \left(I - \alpha\left(AD^{-1} + \delta SD^{-1}\right)\right)^{-1}(1-\alpha)Y \tag{4.8}$$

3. if $\sigma = 0$, we got the Standard Laplacian method with PCA regularization:

$$Z = \left(I - \alpha\left(D^{-1}A + \delta SD\right)\right)^{-1}(1-\alpha)Y \tag{4.9}$$

4.2.1 Scaling of GDPCA by Markov-Batch Stochastic Approximation

In this subsection, we show that the results from Chapter 3 can apply to the scaling of GDPCA. In particular, this application relies on the following assumption, if the Proposition 2 executes, then GDPCA utilises the PowerIteration method, which we can replace by the MBSA. In other words, subject to the fulfilment of Proposition 2, we can rewrite the GDPCA (Algorithm 8) in the following batchwise regime:

Algorithm 9: GDPCA-MBSA

INPUT : A, Y, δ, α, $\mathcal{I}$, bs, ϵ, γ

INITIALIZE: $\widetilde{A}$, $Z^0 = Y$, $\mathcal{S}$, $\mathcal{A}_t = \mathcal{A}_0$;

$\bar{X}_i^T = X_i^T - \frac{1}{d}\sum_j^d X_j^T \forall i \in (1, \ldots, n)$; $S = \frac{\bar{X}^T\bar{X}}{d-1}$

$\widetilde{P}$, Q;

for $t \leftarrow 0$ **to** $\mathcal{I}$ **do**

  Pick $\mathcal{A}_{t+1}$ with probability $Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}$;

  $Z_i^{t+1} = Z_i^t + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}} \left(\frac{\alpha \sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} B_{i,j} Z_j^t}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}} - Z_i^t + (1-\alpha) Y_i \right)$;

  $\mathcal{A}_t = \mathcal{A}_{t+1}$;

end

where $B = [B_{i,j}]_{i=1,j=1}^n = D^{-05}AD^{-05} + \delta SD$ and $\mathcal{I}$ is the number of iterations. Note that this regime of GDPCA avoids the OOM issues on the extremely large graphs. We want to emphasize that the issue with OOM in GDPCA can arise when we have to keep in memory dense covariance matrix S for calculating PowerIteration on extremely large graphs. However, thanks to MBSA, we can process in GDPCA only a small submatrix of the covariance matrix S per iteration, which can be read from SSD in an online regime.

4.3 Ablation studies of GDPCA

4.3.1 Significance of the covariance matrix

In this experiment, the aim is to verify that the use of the covariance matrix S actually leads to an improvement. In order to do so, we compare GDPCA with PRSSL ($\delta = 0$) and other values of δ, as well as with variants of GDPCA where S is replaced with the following efficient similarity matrices: $W_{COS} = \frac{[COS(X_i,X_j)]_{i,j=1}^{n,n}}{d-1}$ and $W_{RBF} = \frac{[RBF(X_i,X_j)]_{i,j=1}^{n,n}}{d-1}$. Table 4.1 displays the average accuracies of each variant along with their statistical significance evaluated with t-tests. It shows that using S in GDPCA is significantly better on the Cora, Citeseer and Pubmed datasets, where it outperforms the others at

least by 7%, 8% and 3% respectively. Notice that Table 4.2 contains accuracy on a test set of fixed dataset splits: as in [23] for Citeseer, Cora, Pubmed and Covid Clinical Trials (CCT) datasets; as in [103] for MNIST dataset, and Table 4.1 has accuracy on test sets averaged over 50 random splits. All experiments mentioned above are available through a GitHub repository[3].

Table 4.1: Average accuracy (%), ▲ denotes the statistical significance for $p < 0.05$.

Dataset	GDPCA $\delta = 1$ (S)	GDPCA $\delta = 10^{-3}$ (S)	PRSSL $\delta = 0$	GDPCA $\delta = 1$ (W_{COS})	GDPCA $\delta = 1$ (W_{RBF})
Cora	77.3 ▲	71.8	69.8	70.1	68.3
Citeseer	73.0 ▲	65.1	44.8	64.8	44.5
Pubmed	68.7	75.8 ▲	67.9	72.6	71.1
CCT	60.4 ▲	54.5	55.6	54.2	56.2
MNIST	62.5	85.3▲	82.6	60.6	59.2

4.3.2 Generation of synthetic adjacency matrix

For selecting the best synthetic adjacency matrix for GDPCA, we have considered three standard distances, such as *Cosine*, *Minkowski*, *Dice* and the number of neighbours from 1 till 14 for KNN algorithm. The accuracy of GDPCA on above parameters on the validation set for MNIST and CCT datasets are shown in Figure 4.4. Figure 4.4 shows that the best GDPCA accuracy on the validation set is obtained with the use of 7 neighbours and *Dice* distance for the CCT dataset is obtained with the use of 7 neighbours and *Cosine* distance for the MNIST dataset.

4.3.3 Hyperparameters selection

We adjusted the parameters for GDPCA on the synthetic dataset generated from the multivariate normal distribution with the following parameters:

$$\mu_1 = [1., 1., 1., 1., ..., 1.], \mu_2 = [-1., -1., -1., -1., ..., -1.], C_1 = C_2 = I, n_l = 100,$$

$n_u = c_0 * 100$ with $d = 1000$, $n_l = 100$, $n_u = 5000$. We utilize the mentioned above parameters taking into account Assumption 1. Note that for each combination of $\gamma \in \{0.5, 0, 1\}$ and $\alpha \in \{0.001, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ we repeat GDPCA

[3]https://github.com/KamalovMikhail/GDPCA

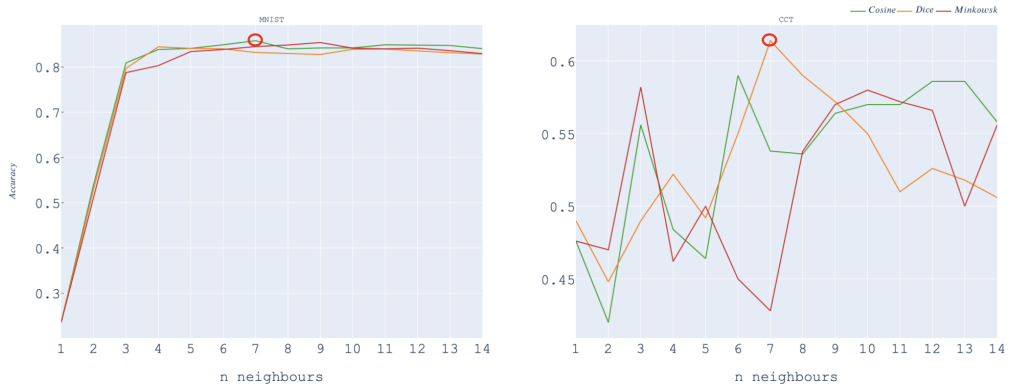

Figure 4.4: Estimate different adjacency matrix for GDPCA.

1000 times. Figure 4.5 presents the medians accuracy of GDPCA. In particular, Figure 4.5 shows that the best performance GDPCA achieves as a Standard Laplacian method ($\delta = 1$) with $\alpha = 0.001$.

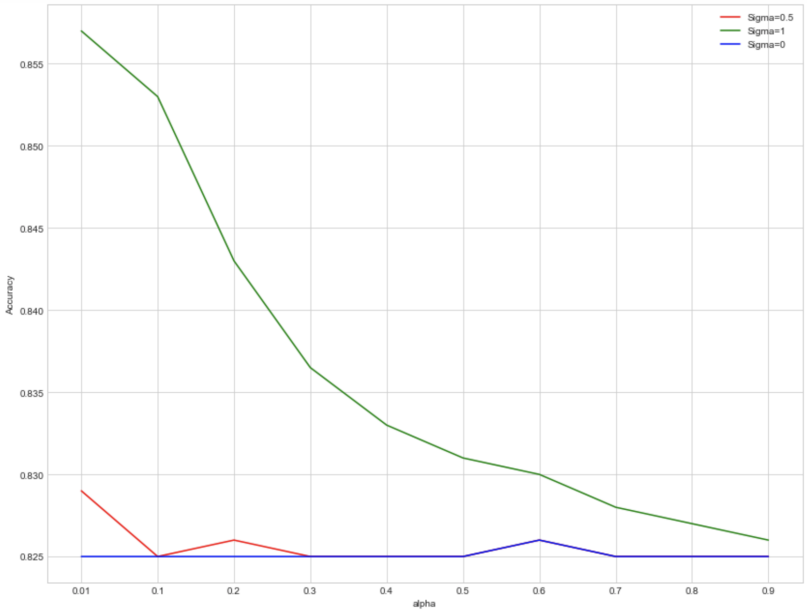

Figure 4.5: Hyperparameter selection for GDPCA.

Table 4.2: Classification accuracy (%) comparison with linear algorithms.

Dataset	Cora	Citeseer	Pubmed	CCT	MNIST
TSVM [9]	57.5	64.0	62.2	–	83.2
KNN [104]	43.9	47.4	63.8	57.1	74.2
LP [71]	68.0	45.3	63.0	53.5	34.2
ManiReg [5]	59.5	60.1	70.7	–	–
PRSSL [4]	69.3	45.9	68.4	55.8	87.2
GDPCA	**77.7**	**73.1**	**76.1**	**61.1**	**88.4**
GDPCA-MBSA	75.3	70.5	72.3	58.3	86.5

4.4 Experimental results for GDPCA

4.4.1 Performance (Accuracy)

The comparisons in terms of accuracy (%) are presented in Table 4.2 and Table 4.3. Table 4.2 shows that GDPCA outperforms other state-of-the-art (SOTA) classical diffusion-based algorithms, especially it is significantly better on the Cora, Citeseer and Pubmed, where it outperforms the others by 8%, 9%, 5% respectively, and even outperforms the linear non-graph based algorithms such as Transductive SVM (TSVM) and KNN. Also, note that even scaled GDPCA-MBSA mostly outperforms SOTA classical diffusion-based algorithms and fair outperforms the linear non-graph based algorithms. Moreover, Table 4.3 shows that our linear GDPCA framework as well as GDPCA-MBSA provides performance that is close to the best SOTA graph convolution networks. Note that GDPCA has a fixed explicit solution (4.5) as opposed to the graph convolution networks, which depend on the layer's weights initialization process. Furthermore, we want to underline that Table 4.2 and Table 4.3 show that GDPCA has a good performance on standard Cora, Citeseer, Pubmed and MNIST as well as on real dataset Covid Clinical Trials (CCT). It is critical to point out that as GDPCA shows high accuracy on both graph-based and non-graph-based datasets, *Versatility limitations* can be addressed using GDPCA. For instance, in Chapter6, we demonstrate how semi-supervised learning on time series data can be accomplished using the GDPCA modification.

4.4.2 Memory vs Time tradeoff

We finish this experimental section by comparing the computational complexity of GDPCA with the SOTA algorithms that obtained the most similar performance, namely GCN and Planetoid. The computational complexity of GDPCA is $\mathcal{O}(\mathcal{I}n^2k)$ in the case of

Table 4.3: Classification accuracy (%) comparison with neural network algorithms.

Dataset	Cora	Citeseer	Pubmed	CCT	MNIST
SemiEmb [25]	59.0	59.6	71.1	–	–
DeepWalk [24]	67.2	43.2	65.3	–	–
Planetoid [23]	75.7	64.7	77.2	–	–
GCN [6]	**81.5**	70.3	**79.0**	55.2	81.4
GDPCA	77.7	**73.1**	76.1	**61.1**	**88.4**
GDPCA-MBSA	75.3	70.5	72.3	58.3	86.5

Table 4.4: Comparison of computational complexity, where l is the number of layers, n is the number of nodes, d is the number of features, r is the number sampled neighbors per node; ϕ is the number of random walks; p is the walk length; w is the window size; m is a representation size; k is the number of classes; bs is a batch size; e' is the number of non-zero elements in matrix $(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1})$ and e'_{bs} is a submatrix of matrix e'.

Algorithm	GCN	GDPCA	GDPCA-MBSA	Planetoid
time	$\mathcal{O}(led + ln^2dm)$	$\mathcal{O}(\mathcal{I}n^2k)$	$\mathcal{O}(\mathcal{I}(bs)^2k)$	$\mathcal{O}(\phi npw(m + m\log n))$
Memory	$\mathcal{O}(lnd + ld^2)$	$\mathcal{O}(e')$	$\mathcal{O}(e'_{bs})$	$\mathcal{O}(nld^2)$

PowerIteration, and $\mathcal{O}(\mathcal{I}nk)$ in the case of GMRES. Since we can replace PowerIteration with MBSA, the complexity can reduce almost n times ($bs \ll n$). Moreover, note that GMRES can be distributed over classes. The comparison of GDPCA framework with GCN and Planetoid algorithms in big-$\mathcal{O}$ notation is presented in Table 4.4. Figure 4.6 provides the time (in seconds) of 50 completed trainings on CPU (1.4GHz quad-core Intel Core i5) for each algorithms. It shows a clear advantage of GDPCA over the GCN and Planetoid especially with GMRES, in terms of computational time. Note that datasets description, parameters of algorithms for comparison and links on datasets/implementations of algorithms are in Section 4.6.

4.5 Proofs

4.5.1 Proposition 1

Proof. This proof uses the same strategy as the proof of Proposition 2 in [4]. Rewriting Problem (4.4) in matrix form with the standard Laplacian $L = D - A$ and

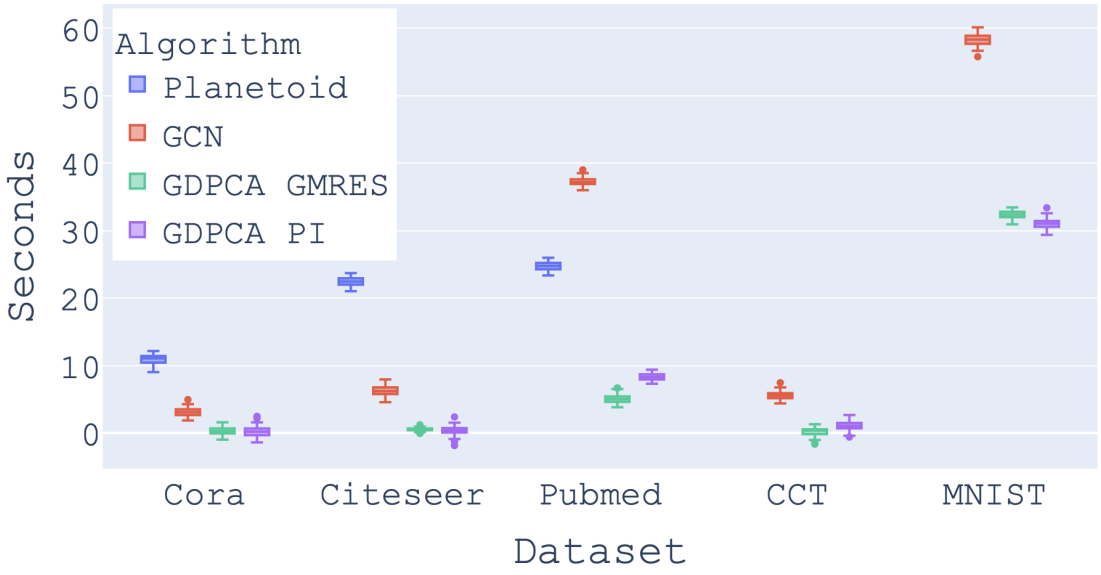

Figure 4.6: Computational time of 50 completed trainings on CPU.

with $Z_i, Y_i \in \mathbb{R}^{n\times 1}$:

$$\begin{aligned} Q(Z) = 2\sum_{i=1}^{k} Z_i^T D^{\sigma-1} L D^{\sigma-1} Z_i \\ + \mu \sum_{i=1}^{k} (Z_i - Y_i)^T D^{2\sigma-1} (Z_i - Y_i) \\ - \delta \sum_{i=1}^{k} Z_i S Z_i^T \end{aligned}$$

where $S = \bar{X}^T \bar{X}/(d-1) \in \mathbb{R}^{n\times n}$. Considering $\frac{Q(Z)}{\partial Z} = 0$:

$$\begin{aligned} & 2Z^T (D^{\sigma-1} L D^{\sigma-1} + D^{\sigma-1} L^T D^{\sigma-1}) \\ & + 2\mu (Z - Y)^T D^{2\sigma-1} \\ & - \delta Z^T (S + S^T) = 0 \end{aligned}$$

Multiplying by $D^{-2\sigma+1}$ and replacing $L = D - A$ results in:

$$\begin{aligned} & Z^T (2I - 2D^{\sigma-1} A D^{-\sigma} \\ & + \mu I - 2\delta S D^{-2\sigma+1}) - \mu Y^T = 0 \end{aligned}$$

Taking out the μ over the parentheses and transposing the equation:

$$Z = \frac{\mu}{(2+\mu)}(I - \frac{2}{(2+\mu)}(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1}))^{-1}Y$$

Finally, the desired result is obtained with $\alpha = 2/(2+\mu)$. □

4.5.2 Proposition 2

Proof. Apply Theorem 1 of sums of spectral radii [105] for the following inequality:

$$\rho(D^{\sigma-1}AD^{-\sigma} + \delta SD^{-2\sigma+1}) \leq \rho\left(D^{\sigma-1}AD^{-\sigma}\right) + \rho(\delta SD^{-2\sigma+1}) < 1/\alpha$$

based on the fact that spectral radius of a matrix similar to the stochastic matrix is equal to 1 (Gershgorin bounds):

$$1 + \delta\rho(SD^{-2\sigma+1}) < 1/\alpha$$

apply the Theorem 7 [106] for replacing $\rho(SD^{-2\sigma+1})$ by the γ maximum singular value of $SD^{-2\sigma+1}$ we obtain the desired result in (4.6). □

4.6 Experimental details

4.6.1 State-of-the-art (SOTA) algorithms

As some of the SOTA algorithms cannot be applied to all types of datasets, we consider specific SOTA algorithms depending on the datasets. For the graph-structured Citeseer, Cora and Pubmed datasets, we compare GDPCA to the LP [71] and ManiReg [5] linear graph diffusion algorithms and to the SemiEmb [25], Planetoid [23], GCN [6] and DeepWalk [24] graph convolution-based neural networks. For MNIST, we compared it to the transductive SVM (TSVM) [9] and KNN [104] linear algorithms, and to the GCN neural network. Finally, for CCT, we compared it to the linear LP [71], KNN [104], and PRSSL [4], and to GCN.

4.6.2 Parameters

Accuracy for non-reproduced benchmarks

Since for training and estimation of the GDPCA framework, we use the train/validation/test split strategy for Citeseer, Pubmed, Cora and CCT datasets as in [23] we can use the accuracy of SOTA algorithms from work [23]. In particular, we can take the accuracy of LP [71], ManiReg [5], TSVM [9], SemiEmb [25], Planetoid [23] algorithms from work [23],

and the GCN [6], DeepWalk [24] algorithm's accuracy from work [6]. Since for MNIST dataset we use the train/validation/test split strategy as in [103] we can use the value of accuracy of KNN [104] and TSVM [9] algorithms from work [103].

Algorithm parameters for reproduced benchmarks

We trained LP,PRSSL, KNN and GCN on CCT and MNIST datasets with the best hyperparameters defined in the articles describing these algorithms: LP [71] $RBF(\cdot)$ kernel function; GCN [6] 0.5 dropout rate, $5 \cdot 10^{-4}$ L2 regularization, 16 hidden units and 200 epochs; KNN parameters selected by Randomized Search [107] for Cora, Citeseer, Pubmed and CCT datasets.

For a fair model comparison between GDPCA, PRSSL and GCN, we replaced A by $A + I$ as was done in [6, 23]. Also, for GDPCA and PRSSL we fixed $\alpha = 0.9$ and $\sigma = 1$ on all datasets as it was shown in [4] that these parameters provide the best accuracy result for PRSSL. We trained GDPCA on Cora, Citeseer and CCT with $\delta = 1$, $\mathcal{I} = 10$, $\tau = 10^{-3}$, $\epsilon = 10^{-3}$, and the same for MNIST and Pubmed but changing the value of δ to 10^{-3}. We selected these specific $\mathcal{I}$, ϵ, τ parameters by Random Search algorithm [107] as a trade-off between fast computation with GMRES and PowerIteration and accuracy on the validation set. Moreover, for MNIST and CCT we generated a synthetic adjacency matrix A by KNN with respect to the results from Ablation studies 4.3. In particular, we generated synthetic adjacency matrices based on the following parameters of KNN for datasets: for CCT - *Dice* distance and 7 nearest neighbours; for MNIST - *Cosine* distance and 7 nearest neighbours. We used these synthetic adjacency matrices for the training of GDPCA, PRSSL and GCN algorithms. It is important to note that the same hyperparameters utilized for MBSA in the experimental section of Chapter 3 were also used for MBSA in GDPCA-MBSA.

4.6.3 Datasets description

In this part of work, we consider two types of datasets: datasets with an underlying graph structure, and datasets that are non-graph based. The latter allow us to test the flexibility of our framework.

Graph-based datasets.

We consider the citation networks datasets of Cora, Citeseer, and Pubmed [108]. These datasets have bag-of-words [73] representation for each node (paper) features and a

citation network between papers. The citation links are considered as edges in the adjacency matrix A. Each paper has a class label ($X_i \in \mathcal{C}_j$).

Non-graph based datasets.

Images. We consider the standard MNIST image dataset [109] composed of square 28×28 pixel grayscale images of handwritten digits from 0 to 9. Besides, we flattened square pixels in 784 d-space features for this dataset. *Text data. Covid clinical trials (CCT) crawled dataset.* We consider a second non-graph based dataset which we prepared and processed from the ClinicalTrials resource[4] from summaries of evidence-based [110] clinical trials on COVID. This dataset is particularly important given the current need from medical experts on this topic. We analyzed 1001 xml files as follows:

1. the feature matrix X was generated from a bag-of-words model based on the descriptive fields "official_title", "brief_summary", "detailed_description", "eligibility";

2. the label matrix Y was generated from the field "masking", which takes values in (*Open, Blind*)[5], as it is one of the essential parameters of evidence-based medicine EBM [111]. The type of masking corresponds to the way of conducting clinical trials: the *Open* way is a less expensive and complicated procedure than the *Blind* one.

Note that the CCT dataset could be useful to other researchers who wish to improve even further the labeling of COVID clinical trials. The registration procedure of clinical trial is useful when authors forget to create masking tag for their work. Particularly after analyzing 1001 xml files, we found that from 3557 clinical trials 1518 of them do not have a masking tag.

As the non-graph based datasets do not have a predefined graph structure, we apply the K-nearest neighbours (KNN) [104] algorithm to generate the adjacency matrix. In Ablation studies 4.3, we show on validation sets of MNIST and CCT datasets how the choice of distances and number of neighbours for the generation of the adjacency matrix by KNN influence GDPCA. We followed the strategy for train/validation/test splitting as in [23] for Pubmed, Citeseer, Cora and CCT, and as in [103] for MNIST. The above datasets and code with GDPCA are available through a GitHub repository[6]. Table 4.5 provides a description of these datasets, where $LR = n_l/n$ is the learning rate with n_l the number of labeled nodes.

[4]https://clinicaltrials.gov/ct2/resources/download#DownloadMultipleRecords

[5]In order to simplify the labeling process, we replaced the long description of masking by a shorter version (e.g. Single Blind (Participant, Investigator) by *Blind*).

[6]https://github.com/KamalovMikhail/GDPCA

Table 4.5: Dataset statistic.

	Citeseer	Cora	Pubmed	CCT	MNIST
n	3327	2708	19717	2039	50000
e	4732	5492	44338	–	–
k	6	7	3	2	10
d	3703	1433	500	7408	784
LR	0.036	0.052	0.003	0.019	0.002
c_0	0.898	1.889	39.43	0.275	63.77

Chapter 5

Generative PageRank

Nowadays, graph convolution networks are a rapidly growing research direction in graph-based semi-supervised learning. At the same time, the recently proposed graph convolution networks employ a default adjacency matrix with binary weights on edges (citations), which results in the loss of the nodes (papers) similarity information. Therefore, in this chapter, we offer a framework that aims to include PageRank into a generative model called GenPR. Through the reweighted adjacency matrix and node similarities in the latent space, this framework enables cooperative training of the node's latent space representation and label spreading. In particular, we describe how a generative model might enhance precision and lessen the number of PageRank SSL iterations. Furthermore, we show that GenPR supports batchwise training by application of Markov-Batch Stochastic approximation algorithm. On four open citation graph data sets, we further demonstrate that GenPR beats the best graph convolution networks and enhances the interpretability of classification findings. Finally, we demonstrate that GenPR performs well on datasets that are not graph-based, such as picture datasets (e.g. MNIST).

The main goal of this chapter is as follows. We propose the GenPR framework, with an emphasis on the simultaneous enhancement of classification accuracy and node latent space representation. We demonstrate how the PageRank-based method's classification results are favourably impacted by the reweighted existing adjacency matrix by the similarity matrix in latent space. Additionally, we demonstrate that GenPR uses fewer PageRank PowerIteration steps and considerably outperforms the most recent state-of-the-art (SOTA) graph convolution networks on all datasets. Last but not least, we demonstrate how GenPR may be used to analyze and explain the classification results.

5.1 Generative PageRank (GenPR)

Let's first establish the notation that will be required for the GenPR framework's further explanation. The standard input for graph-based SSL algorithms is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $n = n_l + n_u = |\mathcal{V}|$ nodes (papers) where n_l and n_u are the number of labelled and unlabelled nodes respectively, $e = |\mathcal{E}|$ edges (citations), $A = [A_{i,j}]_{i,j=1}^{n,n} \in \mathbb{R}^{n \times n}$ is the adjacency matrix and $X = (x_{i,j})_{i,j=1}^{n,d}$ is a matrix of nodes where each node $x_i = (x_{i,1}, \ldots, x_{i,d}) \in \mathbb{R}^d$ has a feature representation in d-space. In the context of citation graphs X is a bag-of-words representation for the nodes (papers). Moreover, each node belongs to one of c classes $\{\mathcal{C}_1, \ldots, \mathcal{C}_c\}$. Also we have the labels matrix $Y = (y_{i,j})_{i,j=1}^{n,c} \in \mathbb{R}^{n \times c}$ such that $y_{i,j} = 1$ if $x_i \in C_j$ and $y_{i,j} = 0$ otherwise.

5.1.1 Intuition of GenPR

Before we go into the details of our framework, let us define the motivation and intuition behind GenPR. The main idea of GenPR is to resolve the following *Binary edges* issue:

1. adjacency matrix with edge: $A_{i,j} = 1$ does not provide the information about impact of cited paper j on the citing one i;

2. adjacency matrix with edge: $A_{i,j} = 0$ may show that author i did not cite the paper j, but he could have used some information from it.

Let us define additional useful notation for the GenPR intuition: $x_i \in X$ is a i.i.d. samples of some continuous random variable x, then an output of multi layer perceptron (MLP) $Y^* = [y_i^*]_{i=1}^n \in \mathbb{R}^{n \times c}$ is a sample from random variable y^* given x as an input; $Z = [z_i]_{i=1}^n \in \mathbb{R}^{n \times d'}$ where z_i is a latent representations of each node x_i sampled from latent random variable z in d'-space; $W = [w_{i,j}]_{i,j=1}^n \in \mathbb{R}^{n \times n}$ is a similarity matrix where each element $w_{i,j} = h(z_i, z_j)$, $\forall z_i, z_j \in Z$ is an output of some positively defined kernel h; $A' = A + \gamma W$ is a reweighted adjacency matrix A with a regularization parameter $\gamma \in [0, 1]$ for W; $D'_{i,i} = \sum_{j=1}^n A'_{i,j}$ is a diagonal matrix.

Let us redefine the recurrent formula of PRSSL [4] using Y^* at each training epoch as a replacement of real labels $Y = Y^*$:

$$F^t = \alpha D'^{(-\sigma)} A' D'^{(\sigma-1)} F^{t-1} + (1 - \alpha) Y^*; \tag{5.1}$$

where $F^t = [F_i^t]_{i=1}^n \sim [y_{i,j}^{pr}]_{i,j=1}^{n,c} \in \mathbb{R}^{n \times c}$. Here $y_i^{pr} = (y_{i,1}^{pr}, \ldots, y_{i,c}^{pr})$ is a sample from random variable y^{pr} since (5.1) is a transformation of the random variable y^* and $F^0 = Y^*$.

Then assume that F^t will improve the accuracy of Y^* by using the information of nodes similarity in latent space during the t-th iterations. We named it the PageRank spreading assumption. Moreover, we propose to use Y^* as a new labels. Let us notice that and $y^* \sim y^{pr}$ due to the PowerIteration PageRank property $||F^t - Y^*||_1 \leq \frac{1}{1-\alpha}||F^1 - Y^*||_1$ [112](Property 12). This allows us consistently use the aforementioned PageRank spreading assumption in training process of the generative model:

$$p(x, y^*, z) \approx p(x|z, y^{pr})p(y^{pr})p(z) \tag{5.2}$$

where $p(\cdot)$ is a PDF of a random variable.

5.1.2 Objective function of GenPR

In this subsection we consider the inductive regime of GenPR which allows us to train jointly the generative model (5.2) and PRSSL (5.1). Before we go into details of GenPR, we have to define the central assumption of Variational Autoencoder (VAE), which is also used in VAE for SSL [12] :

Assumption 2: Assume that the set of points X are i.i.d. samples of variable x. It is also assumed that x is generated with respect to a latent continuous random variable z in two steps (e.g., see [33]):

1. a value for z_i is generated from some prior distribution $p_\theta(z)$;
2. a value for x_i is generated from conditional distribution $p_\theta(x|z)$.

Hence, we have the following generative model with parameters θ:

$$p_\theta(x, z) = p_\theta(z)p_\theta(x|z) \tag{5.3}$$

where the posterior density $q_\theta(z|x) = \frac{p_\theta(z)p_\theta(x|z)}{p_\theta(x)}$ is typically intractable.

However, under above assumption, we can apply the main idea of VAE [33] using a variational approximation posterior $q_\phi(z|x_i)$ to the true posterior $p_\theta(z|x_i)$ with parameters ϕ.

Now let us define GenPR objective function. It is obtained by maximizing the variational lower bound of the data log-likelihood of (5.2) with variance ϕ and generative θ parameters [33]:

$$\begin{aligned} &\log p(x, y^*) \geq \mathbb{E}_{q_\phi(z|x,y^*)}\big[\log p_\theta(x|z, y^{pr})\big] \\ &+ \mathbb{E}_{q_\phi(z|x,y^*)}\big[\log p_\theta(y^{pr})\big] - D_{KL}(p(z)||q_\phi(z|x, y^*)) \end{aligned} \tag{5.4}$$

where $q_\phi(z|y^*, x) = \mathcal{N}(z|\mu(y^*, x), \sigma^2(x))$ is a multivariate Gaussian distribution parameterized by $\mu(y^*, x)$ and $\sigma(x)$ that are inferred from neural netowrk (NN) layers for expectation and variance respectively; $p_\theta(x|z, y^{pr}) = f_\theta(z, y^{pr})$ is a nonlinear transformation of z and y^{pr} by NN layer; $p_\theta(y^{pr}) = PR(y^*, \mu(y^*, x), A)$ is a linear transformation of y^* by (5.1) (the NN layer version will be defined in the next Subsection 5.1.3), $p(z) = \mathcal{N}(z|0, I)$ is a multivariate Gaussian distribution and $D_{KL}(\cdot||\cdot)$ is the Kullback-Leibler divergence.

Since we can trade the quality generation of x for the quality of y_i^{pr} and estimate y_i^{pr} using the information from the labelled nodes, we can use $\beta \in [0, 1]$ as a weight parameter for $p_\theta(x|z, y^{pr})$ and the categorical crossentropy $\mathcal{U}(F^t, Y) = \sum_{i=1}^{n_l} \sum_{j=1}^{c} (y_{i,j} \cdot \log(y_{i,j}^{pr}))$ for y_i^{pr} estimation. Thus, we obtain from (5.4) the final inductive (I) GenPR objective function:

$$\begin{aligned} &\mathcal{L}(\theta, \phi, x, Y) = \beta \mathbb{E}_{q_\phi(z|x,y^*)}\big[\log p_\theta(x|z, y^{pr})\big] + \log p_\theta(y^{pr}) \\ &- D_{KL}(p(z)||q_\phi(z|x, y^*)) - \mathcal{U}(F^t, Y) \end{aligned} \tag{5.5}$$

The difference between inductive (I) and transductive (T) regimes of GenPR is that transductive GenPR does not use the proposition that y^* is a new labels and an objective function looks as follows:

$$\begin{aligned} &\mathcal{L}_T(\theta, \phi, x, Y) = \beta \mathbb{E}_{q_\phi(z|x)}\big[\log p_\theta(x|z)\big] \\ &- D_{KL}(p(z)||q_\phi(z|x)) - \mathcal{U}(F^t, Y) \end{aligned} \tag{5.6}$$

.

5.1.3 Architecture of GenPR

Since we have defined the objective function of GenPR (5.5) we can explain the GenPR layers architecture. The part of z inference contains the following layers:

$$Y^* = \pi_\theta(X);\ \pi_\theta(X) = h_1(XW_1 + B_1) \tag{5.7}$$

$$\mu(X, Y^*) = h_\mu(concat(X, Y^*)W_\mu + B_\mu) \tag{5.8}$$

$$\sigma(X) = h_\sigma(XW_\sigma + B_\sigma) \tag{5.9}$$

where $h_\cdot$ and $B_\cdot$ are activation functions and biases for neural networks (NN) layers respectively; $W_1 \in \mathbb{R}^{d \times c}$, $W_\mu \in \mathbb{R}^{(d+c) \times d'}$ and $W_\sigma \in \mathbb{R}^{d \times d'}$ are trainable weight matrices of MLP (5.7), expectation (5.8) and variance (5.9) for a NN layer respectively; $(m_i)_{i=1}^n = \mu(X, Y^*)$ is an output of (5.8) layer with $m_i \in \mathbb{R}^{d'}$; $concat(\cdot, \cdot)$ is a matrix concatenation column-wise.

To avoid the issues with high variance of the gradient estimation of $\mathbb{E}_{q_\phi(z|x,y^*)}\big[\log p_\theta(x|z,y^{pr})\big]$ by Monte Carlo method, we follow [33] in using the reparameterization trick to compute a low-variance gradient estimator for $q_\phi(z|x,y^*)$:

$$q_\phi(z|x,y^*) \sim Z, \quad Z = \mu(X,Y^*) + \sigma(X) \odot \epsilon, \epsilon \sim \mathcal{N}(0,I) \tag{5.10}$$

where $\odot$ is an element-wise product and ϵ is a random variable.

Now we can define (5.1) as a sequential sublayers in $PR(Y^*, \mu(Y^*,X), A)$:

1. the reweighting of A:

$$w_{i,j} = h\big(m_i, m_j\big);\ \forall\ w_{i,j} \in W; \tag{5.11}$$

$$A' = A + \gamma W = [A'_{i,j}]_{i=1,j=1}^{n,n}; \tag{5.12}$$

   where γ is a parameter of involvement W in reweighting of A within the range $[0,1]$ and $A'_{i,j}$ is an element of matrix A'. Here we compute the similarities between the outputs of (5.8) because we assume that the expectation of the latent variable z more correctly defines the differences between nodes in latent space.

2. the regularization of A':

$$\hat{A}' = D'^{(-\sigma)} A' D'^{(\sigma-1)};\ D'_{i,i} = \sum_{j=1}^{n} A'_{i,j} \tag{5.13}$$

   where σ is a parameter for selection of regularization type: $\sigma = 1$ is a Standard Laplacian; $\sigma = 0$ is a PageRank; $\sigma = 1/2$ is a Normalized Laplacian;

3. the redefined PRSSL [4]:

$$F^t = \alpha \hat{A}' F^{t-1} + (1-\alpha) Y^*;\ t \geq 0; \tag{5.14}$$

   where $F^0 = Y^*$ (5.7) and F^t is a result of the t-th iterations, smoothly changing the node labels Y^* during iterations.

The final layer is the reconstruction of nodes (papers) $\hat{X} = f_\theta(Z, F^t)$ where $\hat{X} = [\hat{x}_i]_i^n \in \mathbb{R}^{n \times d}$:

$$f_\theta(Z, F^t) = h_2(concat(Z, F^t) W_2 + B_2) \tag{5.15}$$

where $W_2 \in \mathbb{R}^{(d'+c)\times d}$, B_2 are weight and bias for x generation $p_\theta(x|z,y^{pr}) = f_\theta(z, y^{pr})$. We can turn to transductive regime of the aforementioned GenPR layers architecture by using modified loss as in (5.6). The Figure 5.1 presents the difference between inductive (I) and transductive (T) GenPR architectures.

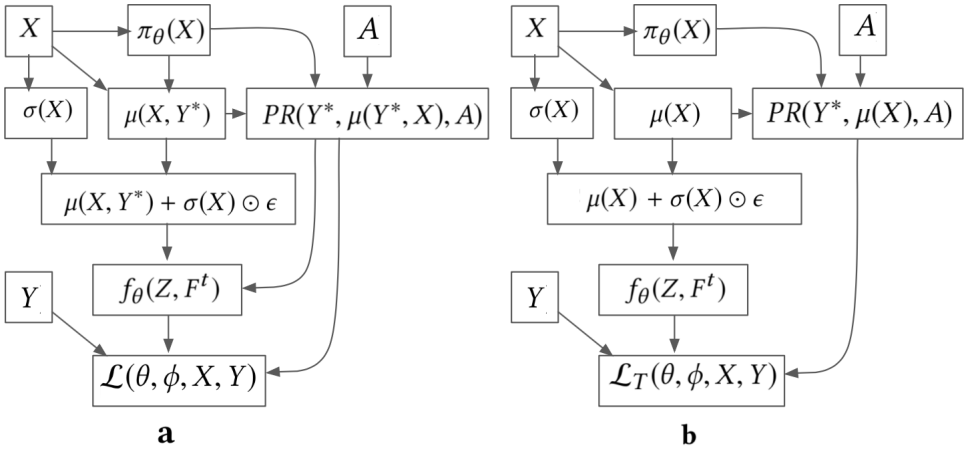

Figure 5.1: The I-inductive (a) and T-transductive (b) architectures of GenPR.

5.1.4 Scaling GenPR by Markov-Batch Stochastic Approximation

It is important to note that the main bottleneck of GenPR arises in redefined PRSSL (5.14), which means that GenPR updates the nodes by PowerIteration algorithms and requires keeping in memory the full adjacency matrix and matrices of node features and weights. In the worst scenario, this bottleneck of GenPR can lead to OOM issues on large graphs. In order to resolve the aforementioned issue, we propose to apply MBSA algorithm for scaling GenPR framework. In particular, we propose to utilize the rule of batch selections and nodes update strategy of MBSA for scaling GenPR in the Algorithm 10:

Note that for reducing computational complexity in GenPR-MBSA we simplify the regularization of reweighted adjacency matrix step (5.13) replacing it with simple matrix normalization: $\frac{A_{i,j}+\gamma h(m_i,m_j)}{\sum_{j\in\mathcal{S}_{\mathcal{A}_{t+1}}} A_{i,j}+\gamma h(m_i,m_j)}$. Additionally, take notice that GenPR-MBSA maintains the same architecture as GenPR, with the exception that all blocks in Figure 5.1 work under a batch regime, and the block for PR is swapped out for the MBSA nodes update strategy. Experimental results for GenPR-MBSA are provided in the following section.

Algorithm 10: GenPR-MBSA

INPUT : A, Y, X, α, γ, τ

for $t \leftarrow 1$ **to** τ **do**
  Pick $\mathcal{A}_{t+1}$ with probability $Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}$;
  $y_i^* = I\{i \in \{\mathcal{S}_{\mathcal{A}_t} \cup \mathcal{S}_{\mathcal{A}_{t+1}}\}\}\pi_\theta(x_i)$;
  $m_i = I\{i \in \{\mathcal{S}_{\mathcal{A}_t} \cup \mathcal{S}_{\mathcal{A}_{t+1}}\}\}\mu(x_i, y_i^*)$;
  $z_i = I\{i \in \{\mathcal{S}_{\mathcal{A}_t} \cup \mathcal{S}_{\mathcal{A}_{t+1}}\}\}m_i + \sigma(x_i) \odot \epsilon$;
  $F_i^t = F_i^{t-1} + \eta(t) I\{i \in \mathcal{S}_{\mathcal{A}_t}\} \frac{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}}{Q_{\mathcal{A}_t,\mathcal{A}_{t+1}}}$
  $\times \left(\frac{1}{\widetilde{P}_{\mathcal{A}_t,\mathcal{A}_{t+1}}} \sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} \frac{A_{i,j}+\gamma h(m_i,m_j)}{\sum_{j \in \mathcal{S}_{\mathcal{A}_{t+1}}} A_{i,j}+\gamma h(m_i,m_j)} y_j^* - F_i^{t-1} \right)$;
  $\hat{x}_i = I\{i \in \mathcal{S}_{\mathcal{A}_t}\} f_\theta(z_i, F_i^t)$;
  $\mathcal{L}_{\mathcal{S}_{\mathcal{A}_t}} \leftarrow I\{i \in \mathcal{S}_{\mathcal{A}_t}\}\mathcal{L}(\theta, \phi, x_i, y_i)$;
  $(W_1, W_\mu, W_\sigma, W_2) \leftarrow$ update by gradient step $\nabla \mathcal{L}_{\mathcal{S}_{\mathcal{A}_t}}$;
  $\mathcal{A}_t \leftarrow \mathcal{A}_{t+1}$;
end

5.2 Experimental results for GenPR

5.2.1 Performance (Accuracy) & Explainability

Performance (Accuracy)

Tables 5.1 shows that the performance of the classification based only on the default adjacency matrix A or on the node features X leads to loss of classification quality because we do not use all available information. In the case of the combination of X and A, GenPR significantly and consistently outperform the others due to the intuition that default A contains incomplete information about nodes similarity. Moreover, Table 5.1 shows that the scaled GenPR by MBSA (GenPR-MBSA) save the high accuracy and outperforms the rest algorithms. In particular, Table 5.1 presents that inductive GenPR-MBSA (I) not only keep the high accuracy but also outperforms non-batch GenPR on Cora-ML and MSA datasets. Furthermore, Table 5.2 shows that GenPR keeps the high performance and outperforms the latest state-of-the-art algorithms on datasets without default graph structure, which means that GenPR allows avoiding the Versatility limitations.

Explainability

Note that the inductive version of GenPR/GenPR-MBSA outperforms the transductive one on Citeseer, Cora-ML and MSA datasets. In particular, since we have reached the

Table 5.1: Average accuracy (%) on citation graphs. △ and ▲ denote the statistical significance (t-test) of GenPR for $p < 0.05$ and $p < 0.01$, respectively, compared to the APPNP.

Input	Algorithms	Citeseer	Cora-ML	Pubmed	MSA
A	PRSSL	71.21	78.12	72.51	76.12
	LP	45.32	68.31	63.12	65.32
X	M2	70.81	79.22	77.6	86.12
X,A	Planetoid	64.71	75.78	77.23	92.88
	APPNP	75.74	85.09	79.71	93.28
	GAT	75.43	84.41	77.73	91.18
	GCN	75.31	83.52	78.65	92.09
	Graphite	71.04	82.12	79.31	92.53
	GenPR (I)	**77.18** ▲	85.52 △	80.09 △	94.08 ▲
	GenPR (T)	76.91△	86.19 ▲	**81.13** ▲	93.81 △
	GenPR-MBSA (I)	77.03	**86.22**	80.74	**95.38**
	GenPR-MBSA (T)	75.61	84.32	79.06	92.45

best results with GenPR (I) and $\gamma = 1$ for Citeseer, it means that latent information is helpful for reweighting the default adjacency matrix A (citation graph). In other words, Figure 5.3 (c) shows that GenPR can be used for the explanation of classification results, by filter the edges by weight and observe nodes with more influence on considered one (e.g. node 545). We have to underline that Figure 5.3 shows the way how GenPR can be used to solve the Explainability of the classification result issue defined in the Introduction.

5.2.2 Memory vs Time tradeoff

Figure 5.2 shows that GenPR outperforms APPNP not only in terms of accuracy, but also in number of PowerIteration steps, because GenPR takes less steps to converge for better accuracy than APPNP. Moreover, GenPR is less complex than APPNP because it uses just one layer for MLP rather than 2 in APPNP. Additionally, Figure 5.4 shows that GenPR-MBSA outperforms the latest state-of-the-art algorithms such as GCN, APPNP, Planetoid and Graphite in terms of memory and time consumption. Even more, Figure 5.4 presents that GenPR-MBSA is significantly reducing the memory and time consumption of default non-batch GenPR. Note that dataset/environment description, parameters of algorithms for comparison and links on datasets/implementations of algorithms are in Section 5.3.

Table 5.2: Accuracy on MNIST.

Algorithm	MNIST
KNN	74.19
TSVM	83.19
CAE	86.53
MTC	87.97
M1 + TSVM	88.18
M2	88.03
GenPR-MBSA	**89.8 ± 0.7**
GenPR	**91.8 ± 0.9**

5.2.3 Denoising

During the experiments with the MNIST dataset, we discovered the positive side effect of GenPR, which allows making image denoising by applying the inductive GenPR. In particular, this makes it possible since GenPR training the VAE part concerning the PageRank classification results, which includes the graph's structure during the training. The inference of VAE from GenPR is given by the generative model $p_\theta(x|z, y^{pr})$ with parameter θ which involves training of GenPR. Figure 5.5 shows the effect of denoising where the left column is the original image and the right column is an image inferred from the VAE part of the inductive GenPR. This side effect can explain by adding the class information spread through PageRank to the latent variable. In future work will be interesting to estimate the impact of PageRank on the denoising process in GenPR.

5.3 Experimental details

5.3.1 Technical environment & Implementations

The computations for GenPR, GCN, Planetoid, Graphite and APPNP for estimation on computational complexity was done on CPU (1.4GHz quad-core Intel Core i5), RAM (16 GB), and we took implementation provided by the authors Table 5.3.

5.3.2 State-of-the-art (SOTA) algorithms

For conducting an experiment in the graph-based SSL area we have taken following citation-graph data sets: Citeseer, Cora-ML, Pubmed, MS-Academic (MSA). The description and statistics of data sets is available in Table 5.4. These datasets are available

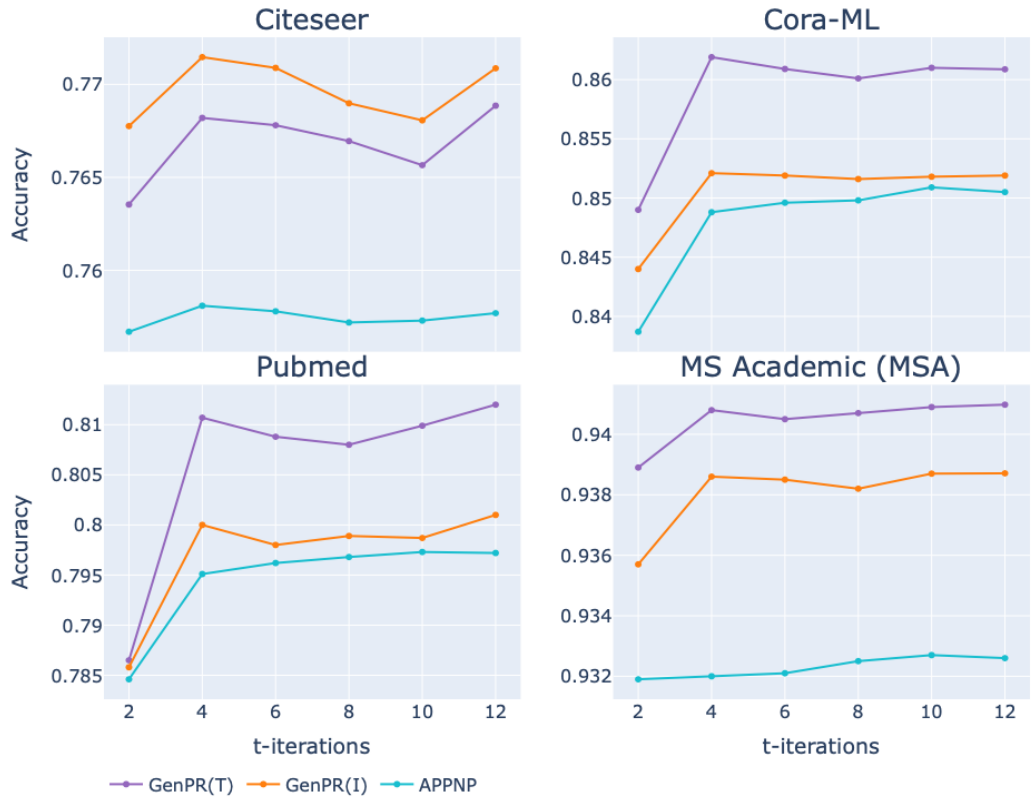

Figure 5.2: Average accuracy of GenPR (I) inductive, GenPR (T) transductive and APPNP over the t-iteration steps.

here[1]. As a baseline algorithms for citation networks we have considered: the classical diffusion-based linear algorithms (LP [3], PRSSL [4]); the recent graph convolution - based NN (Planetoid [23], GCN [6], GAT [7], APPNP [14]); the deep generative model (M2 [12], Graphite [31]). In addition to that, for experiments on the image dataset (MNIST) we took M1, M2 [12] the modifications of VAE SSL, trunsductive SVM (TSVM) [9], manifold tangent classifier (MTC) [10], KNN and contractive auto-encoders (CAE) from [12].

5.3.3 Parameters

To avoid overfitting issue we applied in all experiments L_2 regularization with parameter $\lambda = 0.05$ for weights $W_.$, dropout for $\hat{A}'$ with rate $dr = 0.5$ at each PowerIteration step and learning rate $l = 0.001$ for Adam optimizer. Moreover, we have used the random

[1]https://github.com/klicperajo/ppnp/tree/master/ppnp/data

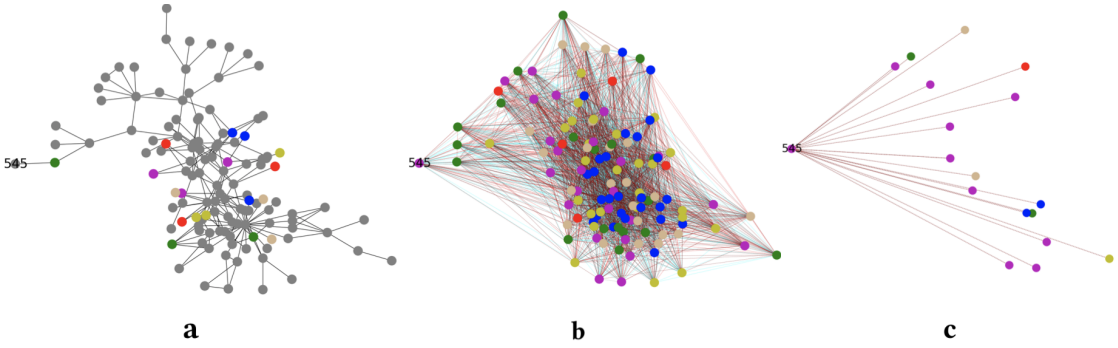

Figure 5.3: Sample nodes from Citeseer data set: a - A before GenPR, where colored nodes are labeled and grey are unlabeled, straight black edges are citations between nodes (papers); b - A' after GenPR, where all colored nodes are result from F^t, and color of an edges by weights from A' (cyan is a lower weights, maroon is a higher weights); c - the result of filtering lower weight edges for the node 545.

Table 5.3: Implementation links.

Methos	URL
APPNP	https://github.com/gasteigerjo/ppnp
Planetoid	https://github.com/kimiyoung/planetoid
GCN	https://github.com/tkipf/gcn
Graphite	https://github.com/ermongroup/graphite
M1, M2	https://github.com/dpkingma/nips14-ssl/
GAT	https://github.com/Diego999/pyGAT
GenPR	https://github.com/KamalovMikhail/GenPRmac

train-test-validation splitting strategy described in [14] and repeated experiments on each data set 500 times. For a fair model comparison we have made an architecture and parameters of GenPR/GenPR-MBSA that are very close to APPNP and GCN. In particular, for all data sets use the intermediate embedding layer $f_0(X) = relu(XW_0 + B_0)$ with $W_0 \in \mathbb{R}^{d \times \hat{d}}$ as the input for (5.7) with $\hat{d} = 64$, $W_1 \in \mathbb{R}^{\hat{d} \times c}$ and $h_1(\cdot) = softmax(\cdot)$, $d' = 64$ in (5.8) and (5.9), $\sigma = 0.5$ and $t = 4$ in (5.13), $B_{\cdot} = 0$. In (5.14) for MSA $\alpha = 0.8$, for Cora-ML, Pubmed and Citeseer $\alpha = 0.9$.

We have selected the specific parameters of GenPR for the non-batch regime and with MBSA (GenPR-MBSA) by the 5 fold cross-validation grid-search[2]. For all data sets GenPR/GenPR-MBSA use $h(m_i, m_j) = (m_i^T m_j)^3$ in (5.11) and $\beta = 0.001$ in (5.5) and (5.6). More precisely, for GenPR we have used: for Citeseer: $\gamma = 1$ in (5.12),

[2]https://scikit-learn.org/stable/modules/grid_search.html

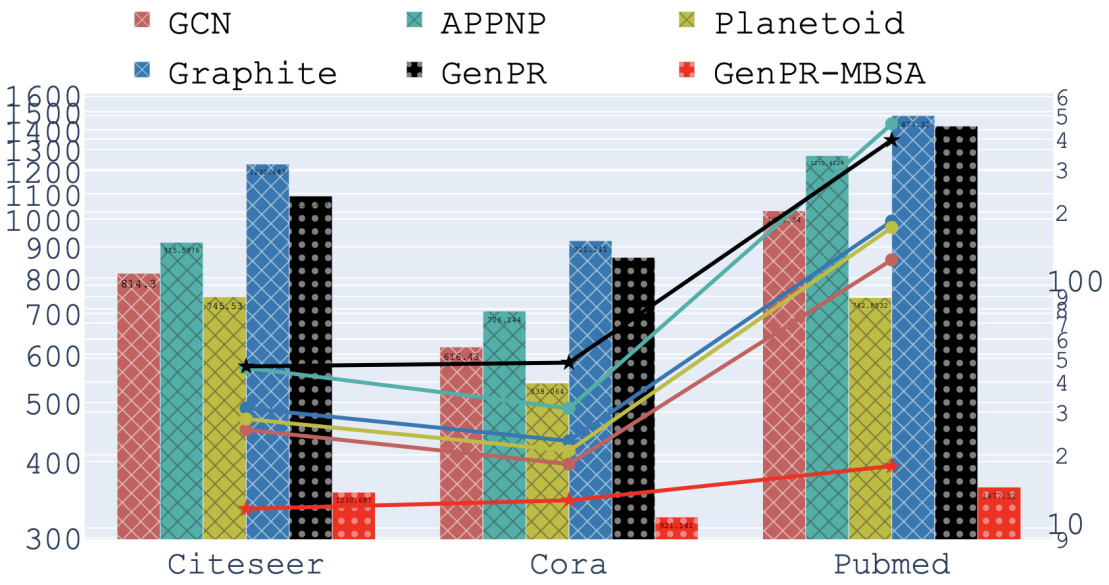

Figure 5.4: Average computational (milliseconds, right axis, lines) and memory (MiB, left axis, bars) complexity per training iteration in epoch over 100 runs for each algorithm (legend, header) on each dataset (x-axis). For this experiment we consider batch GenPR-MBSA (Algorithm 10, $b = 512$).

$h_\mu(\cdot) = h_\sigma(\cdot) = relu(\cdot)$ in (5.8) and (5.9), $h_2(\cdot) = sigmoid(\cdot)$ in (5.15); for Cora-ML, Pubmed and MSA: $\gamma = 0.001$ in (5.12), and $h_\cdot(\cdot) = linear(\cdot)$; for MNIST: $\gamma = 1$ in (5.12), and $h_2(\cdot) = softmax(\cdot)$; for GenPR-MBSA we have used: for all citation networks $h_\mu(\cdot) = h_\sigma(\cdot) = linear(\cdot)$, $\gamma = 0.05$ in (5.12), and $h_2(\cdot) = softmax(\cdot)$; for MNIST: $\gamma = 0.9$ in (5.12), and $h_2(\cdot) = softmax(\cdot)$. Moreover, we emphasize that for MBSA (GenPR-MBSA), we used the same best values of hyper-parameters as defined in ablation studies of MBSA in Chapter 3 (e.g. $\gamma = 0.3, \epsilon = 0.1, bs^*_{min} = \frac{n}{median(D)}, \tau = 100$). Additionally, note that the performance for the latest state-of-the-art algorithms in Tables 5.1, 5.2 was achieved by computing these algorithms with the best hyper-parameters defined in their works.

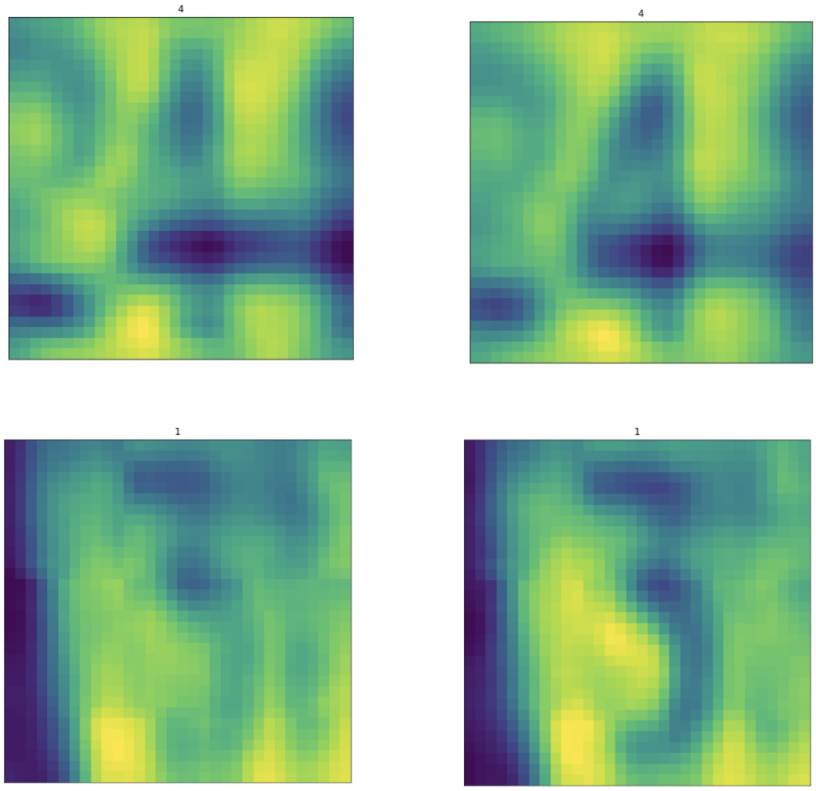

Figure 5.5: Sample of MNIST images denoising (the left column are original images, the right column are images after denoising).

Table 5.4: Dataset statistics.

Parameters	Citeseer	Cora-ML	Pubmed	MSA	MNIST
Nodes	2110	2810	19717	18333	10000
Edges	3668	7981	44324	81894	-
Classes	6	7	3	15	10
Features	3703	2879	500	6805	784
Label rate	0.036	0.047	0.003	0.016	0.01

Chapter 6

PaZoe:classifying time series with few labels

Semi-Supervised Learning (SSL) on graph-based datasets is a rapidly growing area of research, but its application to time series is difficult due to the time dimension. We propose a flexible SSL framework based on the stacking of PageRank, PCA (GDPCA, Chapter 4) and Zoetrope Genetic Programming (ZGP) algorithms into a novel framework: PaZoe. This self-labelling framework shows that graph-based and non-graph based algorithms jointly improve the quality of predictions and outperform each component taken alone. We also show that PaZoe outperforms state-of-the-art SSL algorithms on three time series datasets close to real world conditions. A first set was generated in house, taking data from industrial graded equipment in order to mimick DC motors during operation. Two other datasets, which include the recording of gestures, were taken from the public domain.

The following sections make up this chapter: (i) we start out by describing the key insight that drives our PaZoe framework. Additionally, we highlight this framework's primary elements and describe its training strategy; (ii) finally, we demonstrate that, for time series data, PaZoe outperforms both cutting-edge graph-based and non-graph-based SSL algorithms in terms of accuracy and computational complexity. Additionally, we describe the process through which we gathered the dataset, which includes the many imbalanced states of the real motor. We further demonstrate that PaZoe maintains great performance with varied particular motor specifications.

6.1 PageRank & PCA & Zoetrope Genetic Programming (PaZoe)

Before we go into details of our framework, let us define the central intuition behind it. Our idea is based on the assumption that any type of data can be represented through a graph structure. Since GDPCA (see Chapter 4) outperforms the graph-based as well as non-graph based SSL algorithms, we use it in PaZoe to extend the training set to the self-labelling regime [78]. Additionally, take notice that in this study, we utilize a particular GDPCA learning technique based on Standard Laplacian that we have named PageRank PCA (PRPCA). Then, we assume that training on PRPCA predictions in a supervised regime will enable the Zoetrope mechanism in ZGP to extract meaningful information. We combine these two algorithms on the basis of this supposition.

Let's first establish the notation that will be required for the PaZoe framework's further explanation. Let $X = [X_i]_{i=1}^{n} \in \mathcal{R}^{n \times d}$ be the matrix of input features, with dimension d and total number of observations n. Then let $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ be the set of k classes, and $Y = [Y_i]_{i=1}^{n}$ be a label matrix where $Y_i = (Y_{i,j})_{j=1}^{k}$, such that $Y_{i,j} = 1$ if $X_i \in \mathcal{C}_j$ and $Y_{i,j} = 0$ otherwise. Y is composed of two parts: a labelled one of size n_l, and an unlabelled one of size n_u, typically for SSL $n_l \ll n_u$ and Y_i being the null vector for all unlabelled data. We also define the following graph-based setup which will be used in the sequel: $A = [A_{i,j}]_{i,j=1}^{n,n}$ is an adjacency matrix, $D = \mathrm{diag}(D_{i,i})$ is a diagonal matrix with $D_{i,i} = \sum_{j=1}^{n} A_{i,j}$. The problem of semi-supervised classification is to find an accurate classification result $\hat{Y} = [\hat{Y}_i]_{i=1}^{n}$ for Y, with $\hat{Y}_i = (\hat{Y}_{i,j})_{j=1}^{k}$, based on both labelled and unlabelled data where the amount of labelled data is extremely low.

6.1.1 PageRank & Principal component analysis (PRPCA)

The main idea of PRPCA is to enrich the adjacency matrix A by the information of estimated covariance between objects $S \in \mathcal{R}^{n \times n}$. This enrichment allows spreading information about labelled objects to unlabelled ones. This means that even in the absence of edge between two objects where $A_{i,j} = 0$, we can still spread the information about labels between these objects weighted by their covariance value. The explicit classification solution of PRPCA is given by

$$\hat{Y} = \left(I - \alpha \left(AD^{-1} + \delta S D^{-1}\right)\right)^{-1} (1 - \alpha) Y \tag{6.1}$$

where $\delta \in (0, 1)$ sets the influence of S on A and $\alpha \in (0, 1)$ is the random jump parameter for PageRank. Let us note that in the normalised $\hat{Y}$ if $\left(AD^{-1} + \delta S D^{-1}\right)$ is a stochastic

matrix, equation 6.1 is an explicit PageRank [68] problem. The classification solution 6.1 is obtained through the differentiation of the combination Laplacian regularization[1], supervised[2] and PCA[3] losses. Note that the computation of the matrix inversion can be avoided, thanks to numerical iterative methods [15]. PRPCA presents the following interesting and practical features: first, it has an explicit classification solution (Eq. 6.1) enabling the interpretation of the object's values in each column of Y as the value of its importance in that particular class/column, through the PageRank model; second, it can work in a distributed regime, handling the high amount of unlabelled data without memory issues; and finally, it can support the online learning regime, appending data from a new observed sensor as a new object in a graph and labeling it through its neighbours.

6.1.2 Zoetrope Genetic Programming

The Zoetrope Genetic Programming (ZGP) algorithm is a genetic programming approach for symbolic regression (GPSR) which iteratively evolves mathematical formulae towards the one that best fits the data. The particularity of ZGP among symbolic regression methods lies in its formula construction, which allows efficient computations and prevents models to overgrow and become complex, a common drawback in GPSR. This construction mechanism is illustrated in Figure 6.1 and works as follows. First, a number m_e of elements $(E_1, \ldots, E_{m_e})$ are randomly selected among input features (resp. random constants), with a 90% (resp. 10%) probability. Then, these elements undergo m_m "maturation steps" or "stages", which consists in applying the fusion operation

$$f(E_i, E_j) = r \cdot_1 (E_i, E_j) + (1 - r) \cdot_2 (E_i, E_j),$$

on couples of elements, where $\cdot_i$, $i = 1, 2$ are operators[4] uniformly chosen in a predefined set $\mathcal{O}$, and $r = U[0, 1]$; the result of $f(E_i, E_j)$ replaces either E_i or E_j. At the end of the m_m stages, the matured elements – called *"zoetropes"* and denoted by $(Z_1, \ldots, Z_{m_e})$ – are linearly combined via multinomial logistic regression penalized by Elastic net [113]; this last step allows to jointly select the most relevant zoetropes and optimally estimate their weights. The operator set can be adapted to the problem at hand, but is typically taken as $\mathcal{O} = \{+, -, \times, /, \cos, \sin, \text{sqrt}\}$.

Genetic programming considers models as individuals of a "species", and evolves them with random perturbations (*mutations*) and by mating pairs into new individuals

[1] Laplacian regularization: $\sum_{j=1}^{n} A_{i,j} ||\hat{Y}_i - \hat{Y}_j||_2^2$

[2] Supervised loss: $\sum_{i=1}^{n} ||\hat{Y}_i - Y_i||_2^2$

[3] PCA loss: $||\bar{X}\hat{Y}||_2^2$

[4] In case $\cdot_1$ or $\cdot_2$ is unary, only E_i is taken into account

(*crossover*). ZGP's mutation and crossover are also nonstandard in GPSR: the mutation consists in selecting couples of models, and replace the worst one with a "mutant" of the first one, while the crossover consists in selecting the best and worst in a pool of m_t models, and randomly propagate elements and fusions of the best to the worst model. Note that the "worst" and "best" models are defined with respect to their accuracy on the training set. At the end of each iteration, all the models are evaluated on the validation set, and the best ever is stored. Also, like PRPCA, ZGP can work in distributed regime. For the complete description of the algorithm, see [79].

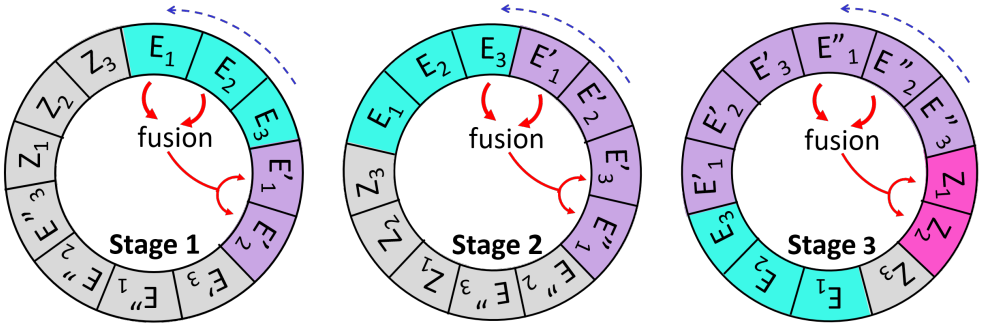

Figure 6.1: Illustration of ZGP's model construction with $m_e = m_m = 3$. For the sake of readability, the third fusion, generating $(E"_2, E"_3)$ from (E'_2, E'_3) is not represented. Note that $Z_3 = E"_3$ as no element is left for a fusion.

6.1.3 PaZoe strategy

Our PaZoe framework is given in Algorithm 11 and consists in three main sequential steps:

1. *Transforming data into graph structure.* For non-graph based datasets, where no adjacency matrix A is available, we first generate a synthetic graph structure and retrieve A by K-nearest neighbours (KNN) with Euclidean distance;

2. *Labelling the unlabelled data.* We then compute PRPCA based on the input matrix X and the adjacency matrix A. Predictions generated by PRPCA consider the graph structure, which could be valuable for stacking with existing object features X for further training of ZGP. Also, self-labelling [78] by PRPCA predictions extends the training set for further ZGP training in the supervised regime;

3. *Classifying and recovering the boundary formulae.* We stack the input data X with the predictions from PRPCA and feed the augmented dataset to ZGP for supervised training (where train/test split of dataset is 70%/30%).

This framework is applicable to any kind of data. In order to adapt it to temporal data obtained from sensors, we propose to modify step 1 of PaZoe as follows: we first separately train a KNN algorithm and generate different adjacency matrices for each type of features, e.g. the magnetometer[5] and the gyroscope[6] in the DC motor dataset (see next section for details); then we linearly combine these adjacency matrices into the final one. Similarly in PRPCA, we compute the covariance between objects separately for each feature.

The outline of PaZoe with the modification for sensor data is illustrated in Figure 6.2. The PRPCA part of the code is publicly available through this link[7]. As for ZGP, we used an open source version of the proprietary algorithm, which is still under testing and has not been released yet.

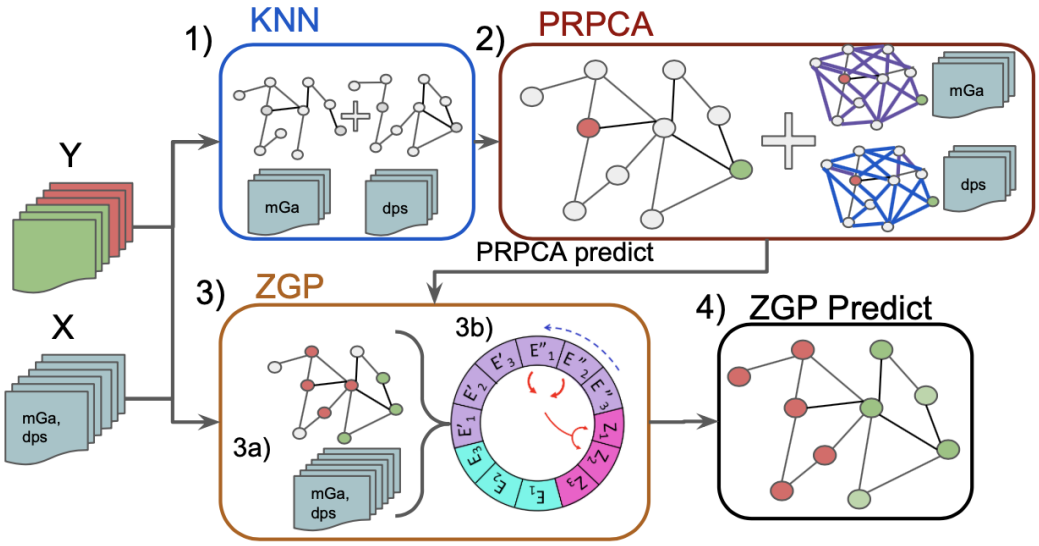

Figure 6.2: PaZoe sequence: 1) Generation of graph structure; 2) Self-labelling by PRPCA; 3): 3a) Stack X with PRPCA predictions; 3b) ZGP training; 4) Final predictions from ZGP. Note, X and units therein, refer to the dc motor dataset.

6.2 Experimental results for PaZoe

We apply the PaZoe framework on three time series datasets, the first generated for this work, the others obtained from the public domain. **DC motor dataset (RPM)** –

[5] $X_{mga} \in \mathcal{R}^{n \times d_{mga}}$ where d_{mga} is dimension of magnetometer
[6] $X_{dps} \in \mathcal{R}^{n \times d_{dps}}$ where d_{dps} is dimension of gyroscope
[7] https://github.com/KamalovMikhail/PaZoe

Algorithm 11: PaZoe

INPUT: X,A, Y, α, δ ;
INITIALIZE: ;
$\bar{X}_i^T = X_i^T - \frac{1}{d}\sum_j^d X_j^T \forall i \in (1, \ldots, n)$;
$S = \frac{\bar{X}^T \bar{X}}{d-1}$;
IF: $A = NaN$:
  $A = KNN(X)$
$\hat{Y} = \left(I - \alpha\left(AD^{-1} + \delta SD^{-1}\right)\right)^{-1}(1-\alpha)Y$
$\hat{X} = stack(X, \hat{Y})$
$\hat{Y} = ZGP(\hat{X}, \hat{Y}, m_p, m_i, m_e, m_m, m_t)$

generated with six classes of imbalance failure on a real motor, by collecting data from a sensor tile (see next section); **UWaveGesture (UWave)** [114] – with eight classes of gestures from (x, y, z) accelerometer features; **Gesture WIImote (WII)** [115] – with ten classes of gestures from (x, y, z) accelerometer features by Nintendo Wiimote (see Figure 6.3).

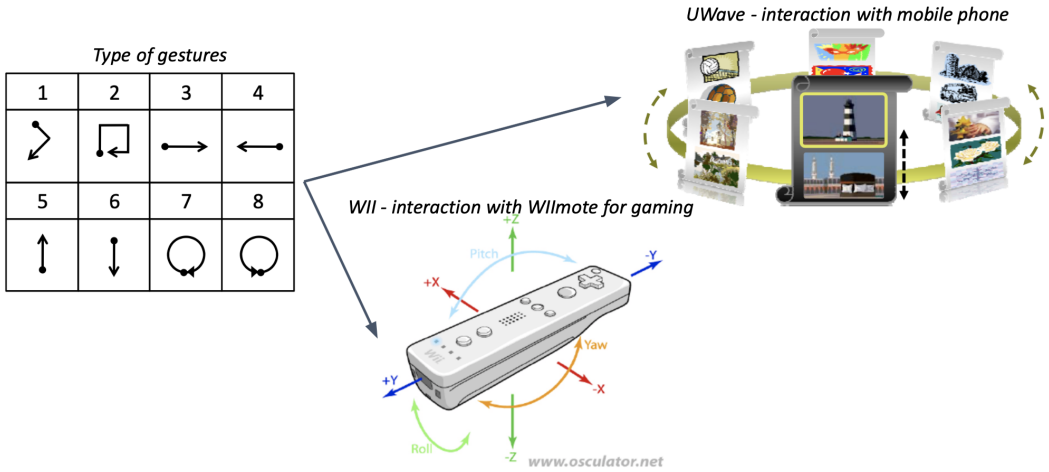

Figure 6.3: Datasets: UWaveGesture (UWave), Gesture WIImote (WII).

6.2.1 DC motor data collection

In order to profit from a real dataset on motor failures, we conducted our own experiment to simulate anomalies of DC motors in a production environment. These are later used as classification targets with labelled data generated for training. Motor axis imbalance were generated by loading weights onto a disk plate mounted on top of the motor at varying distances from its axis. In particular, the five imbalanced states of DC motor

are present in Figure 6.5. The dataset was obtained with industrially graded equipment made of a STMicroelectronics (STM) acquisition board (Figure 6.4), a STM SensorTile with three sensors - accelerometer, magnetometer and gyroscope - and a SD card for data storage. The three components (x, y, z) of each sensor signal were acquired at the default rate of 20 Hz, kept throughout.

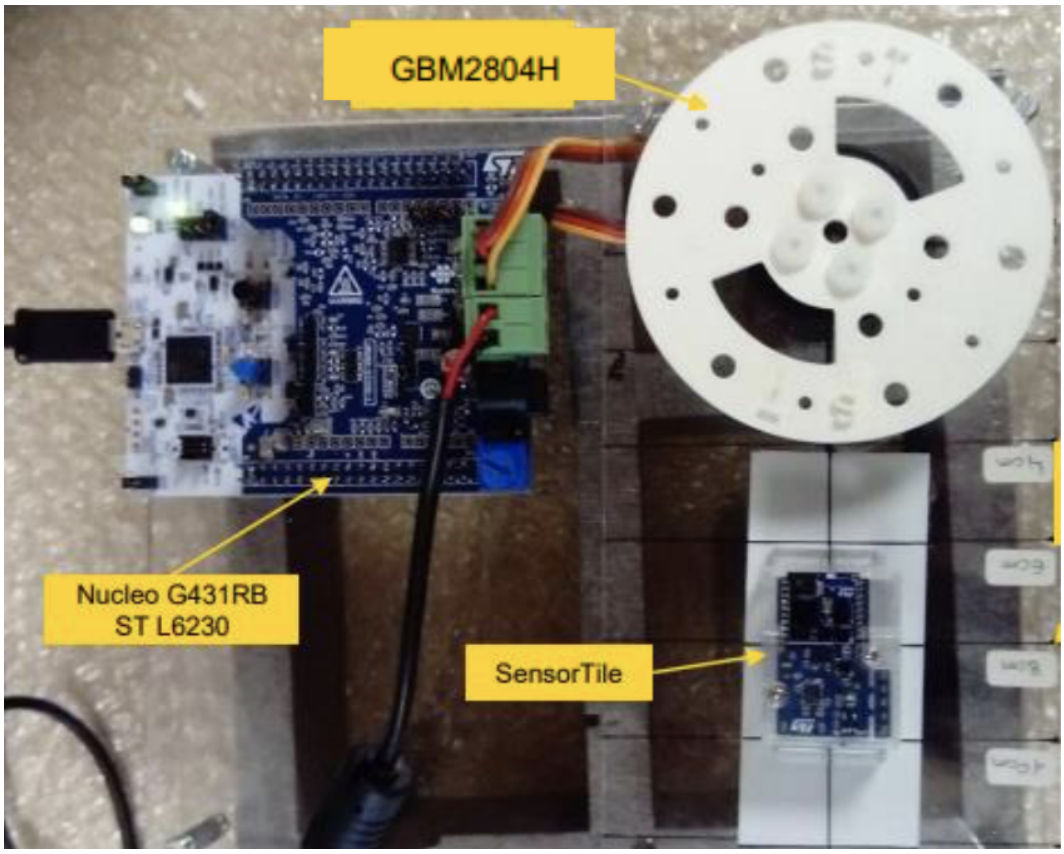

Figure 6.4: STMicroelectronics (STM) acquisition board:Nucleo G431RB ST L6230 with a GBM2804H brushless motor and STM SensorTile.

As environmental variables (temperature, pressure, and humidity) vary over time scales orders of magnitude lower than the dynamic variables, the former were only recorded during preliminary measurements along the day (no further samples were collected during the campaigns). To rule out any daily variations, all measurements taken in the morning were repeated in the afternoon under identical conditions. To account for potential drifts (systematic errors) blank measurements were taken in the absence of loads, at the beginning and at the end of each campaign. These comparisons did not show significant differences. We recorded three rotation speeds, 620, 420 and 220 RPM. We chose these speeds to show how the performances tend to drop the lower the speed, making the

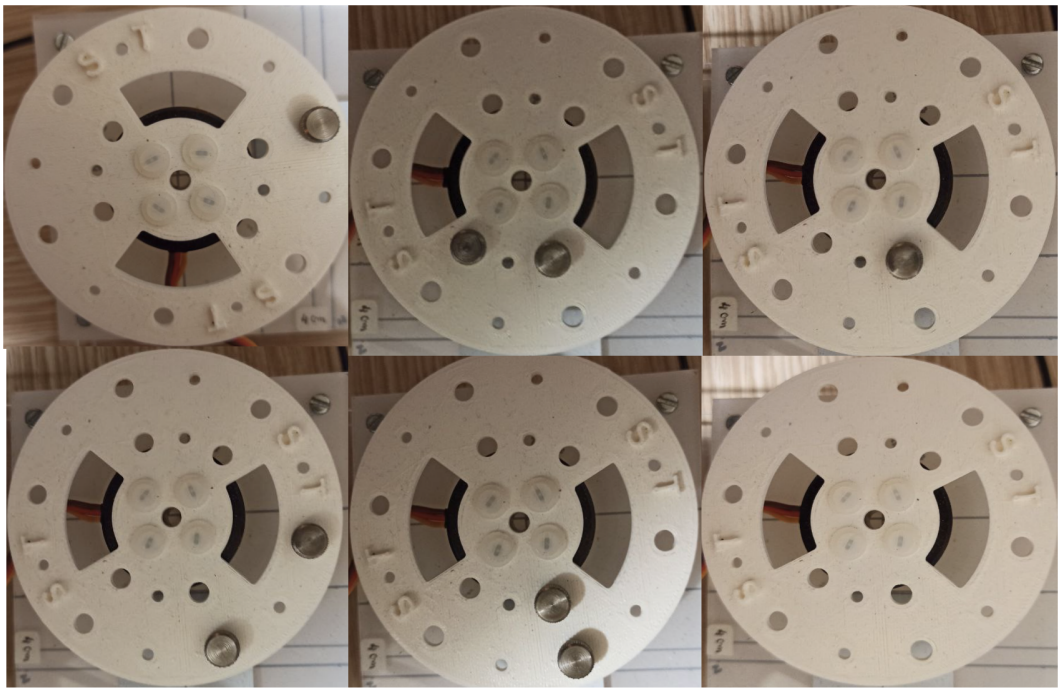

Figure 6.5: The five imbalance states vs the balance state of the DC motor.

discrimination of anomalies more difficult.

The three sensor quantities and units are as follows:

- **Accelerometer (mg)** - acceleration values in units of mg, where $g = 9.81\ m/s^2$ is the gravity acceleration. Here mg, along one of the axis measures as much as O(1000 mg) = 1 g whereas the other axes display values which are less by a factor O(50);

- **Magnetometer (mGa)** - generally used for tracking of moving objects - with values in mGa, where 'Ga' means gauss and 1 $gauss = 10^{-4}T$. The acquisition board is quoted for a 50 gauss magnetic field dynamic range ;

- **Gyroscope (DPS)** - measures rotations in DPS (deg. per seconds), e.g., one needs to convert to rad/s if time or geometric calculations are needed. (ex. 2xPIxR etc). STM claims 0.01 dps/sqrt(Hz) accuracy;

The duration of each experiment is close to one (~ 1) minute.[8]

[8]Each datapoint has a timestamp dd/mm/yyyy hh:mm:ss.xxx, with differences between adjacent points from 2 to 5 ms around the nominal 50 ms. As the time scale is approximately uniform, the absolute value of the time can be safely ignored and timestamps swapped for indexes as necessary.

6.2.2 Data utilization strategy

We used the following train/test split strategy for all of these datasets: 20 labelled objects for each class for training and the rest for testing. Note that all these datasets are balanced, e.g. the number of objects in each class is similar. This strategy is standard for SSL learning algorithms [116], [3]. Also, we have to mention that for the DC motor dataset, we considered objects as sensor quantities (e.g. accelerometer, magnetometer, gyroscope) at each moment in time (recording individual data points). In other words, the length of time series (l) for the DC motor dataset was equal to $l = 1$. In practice, it allows us to check the motor's state and signal imbalance failures at any moment. This is because the position of the motor is stable but, at a successive time instant it might not be.

Since WII and UWave datasets have only observations from accelerometers, we considered an object as a time series with length equal to the motion's length (e.g. following the time evolution of the three different coordinates, (x, y, z) during the complete gesture recording). These three datasets, summarised in Table 6.1, and the code for their processing are available through the provided link[9]. Note that in Table 6.1 the number of observation for 620, 420, 220 rpm datasets is close to 6100. This is because the sensor tile collected with a small number of missing values at the end of each observation time ($\sim$ 1 minute), which leads to slightly unequal number of observations for each type of rpm.

Table 6.1: Dataset statistics

		620,420,220 RPM	UWave	Wii
n	No. observations	$\sim$6100	4478	1000
n_l	No. labels	120	160	200
n_l/n	Ratio of labels	$\sim$ 1.9%	3.6%	20%
l	Sequence length	1	315	326
k	No. classes	6	8	10
d	No. features	9	315	326

6.2.3 Performance (Accuracy) & Computational complexity

For a fair comparison, we used three types of algorithms: (1) GB-SSL such as label propagation (LP), PageRank & PCA (PRPCA) and graph convolution network (GCN) (is a neural network); (2) non-GB-SSL, such as logistic regression (LR) and K-Nearest

[9]https://github.com/KamalovMikhail/PaZoe

Neighbours (KNN); and (3) supervised algorithms such as Support-vector machine (SVM), ZGP and the combination of algorithms such as PRPCA & LR (PaLR) and PRPCA & SVM (PaSVM). For each of these algorithms, we took the best hyperparameters defined in their respective works and for PRPCA we used $\alpha = 0.9$, $\delta = 10^{-3}$. We use accuracy as the performance metric since all datasets are balanced. We report the average accuracy on the test set, taken over 20 random splits (k-folds strategy).

The results of PaZoe on the DC motor dataset obtained with various features combinations are presented in Table 6.2. It shows that the best classification accuracy is achieved by using magnetometer (mGa) or gyroscope (dps) with respect to RPM. Since magnetometer (mGa) and gyroscope (dps) separately provide a high classification accuracy for the DC motor dataset, we use the best of them for each RPM (ie. dps for 620, 420 rpm and mGa for 220 RPM) to train the rest of the algorithms. We believe that in order to improve the performance on 220rpm dataset, we should extend the length of the considered time series (instead of $l = 1$, because the increased number of observations should improve the classification results). Furthermore, Figure 6.6 shows that PaZoe outperforms the LP, GCN and LR in terms of computational complexity. Note that results in Figure 6.6 base on the 20 completed trainings on CPU(1.4GHz quad-core Intel Core i5).

The rest results of PaZoe compared with all the other algorithms on the DC motor dataset are shown in Table 6.3, along with the performance on the WII and UWave datasets. Several comments can be made on those results: first, PRPCA clearly outperforms the other SSL algorithms on all three datasets; second, combining PRPCA with a supervised classification algorithm only leads to an improvement with ZGP (PaZoe); third, PaZoe considerably outperforms its separate components (PRPCA, ZGP) as well as the rest of the SSL and supervised algorithms on all three datasets, even with only one sensor (accelerometer) in the gesture datasets.

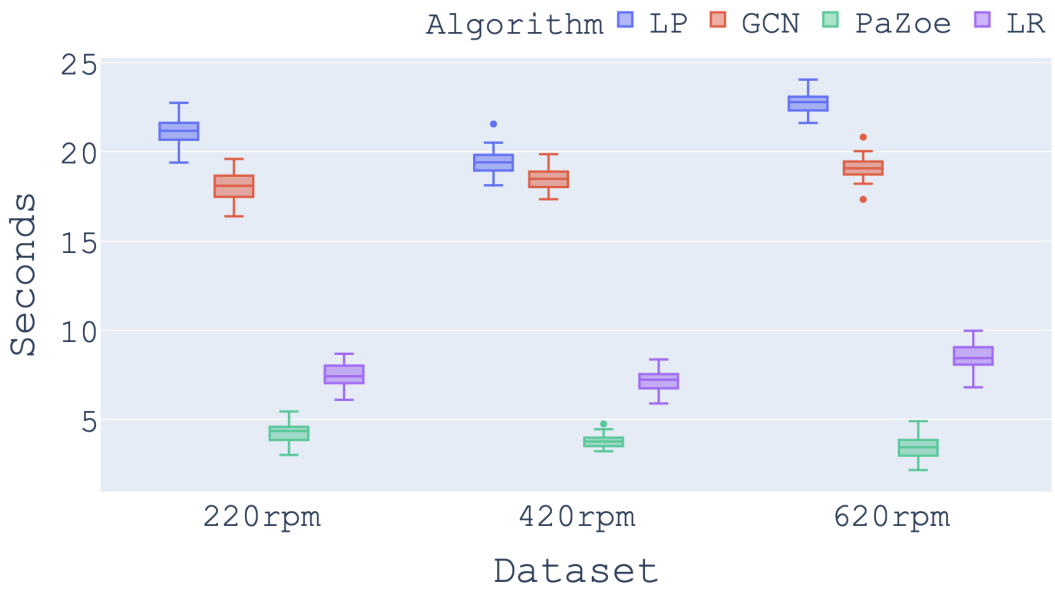

Figure 6.6: Computational complexity of PaZoe.

Table 6.2: Accuracy for the DC motor dataset with various feature sets

RPM	Algorithm	dps, mGa, mg	mGa	mg	dps	mGa,dps
	PRPCA	61.2	19.2	44.2	71.6	68.2
	ZGP	48.9	16.2	35.7	60.1	63.2
620	PaLR	18.4	17.2	19.5	42.9	18.8
	PaSVM	46.7	17.0	41.2	65.8	66.4
	PaZoe	**65.6**	**97.0**	**96.8**	**98.8**	**79.3**
	PRPCA	38.8	60.8	28.3	66.2	51.8
	ZGP	62.4	64.6	29.2	62.3	65.2
420	PaLR	18.0	28.7	22.7	35.2	17.9
	PaSVM	18.7	51.1	26.0	52.5	44.2
	PaZoe	**63.5**	**96.2**	**95.2**	**97.8**	**67.2**
	PRPCA	30.6	66.3	20.1	29.5	37.2
	ZGP	20.2	18.5	16.8	26.1	27.1
220	PaLR	18.5	16.4	17.1	19.0	17.4
	PaSVM	19.8	61.2	18.8	16.5	33.0
	PaZoe	**36.2**	**94.2**	**90.6**	**93.1**	**44.8**

Table 6.3: Accuracy for DC motor, WII and UWave datasets

Dataset	620RPM	420RPM	220RPM	WII	UWave
PRPCA	71.6	66.2	66.3	67.8	70.1
LP	31.2	17.2	16.6	15.2	12.4
KNN	28.6	33.9	60.1	23.7	58.8
GCN	16.9	21.6	18.3	16.7	18.3
ZGP	60.1	62.3	26.1	14.6	17.4
LR	29.7	27.9	16.8	52.9	55.8
SVM	64.1	38.9	25.6	43.3	68.3
PaLR	42.9	35.2	16.4	34.9	62.8
PaSVM	65.8	52.5	61.2	37.3	69.1
PaZoe	**98.8**	**97.8**	**94.2**	**71.8**	**72.3**

Chapter 7

Conclusion

In this thesis, we have proposed batchwise linear and neural network solutions for the main problems with memory and computational limitations, which arise in all directions of graph-based semi-supervised algorithms (GB-SSL) area. In particular, we have shown that the Markov-batch stochastic approximation (MBSA) and its application for scaling graph convolution networks (MBSA-NN) significantly reduce the memory and computational complexity of RAM and GPU. Furthermore, we have proved that the solutions of Personalised PageRank and Graph Convolution Network (GCN) are reached by MBSA and MBSA-NN, respectively. Additionally, we have discovered and investigated generic Laplacian regularization issues, which are encountered by both graph convolution networks and the majority of classical diffusion-based algorithms. To avoid the Laplacian regularization problems, we have defined the linear (Graph-diffusion & PCA, GDPCA) and non-linear (GenPR) frameworks. We have also demonstrated that the suggested GDPCA and GenPR frameworks may be scaled by the MBSA algorithm without sacrificing performance or memory efficiency. Finally, we have demonstrated how the GDPCA framework suggested in this thesis may be modified for the use on a real dataset while guaranteeing minimal computational cost and good accuracy.

In the first part of this work, we have proposed a novel stochastic approximation algorithm called Markov-Batch Stochastic Approximation (MBSA) resolving the Personalized PageRank (PPR) problem, which can be used by itself as a linear algorithm or within graph convolution networks. We have proved the convergence of MBSA to the exact solution of PPR and experimentally showed that MBSA outperforms other linear algorithms such as JOR, RBGS, DSBGS and RK in terms of ordinal convergence rate to an optimal classification result and consumes a small amount of memory. Specifically, we have shown that it is more important to rapidly recover the PPR ranking than find the exact solution's best approximation. Also, we have analysed the convergence of

MBSA with a uniform batch selection strategy to avoid the limitation in the computation of a matrix with the number of edges between batches. Moreover, we have adapted the MBSA algorithm to scale existing graph convolution networks on large graphs and named it as MBSA-NN. In particular, we have proved the convergence of MBSA-NN to a local minimum of the graph convolution network on the entire graph, with MBSA-NN solving the *critical out-of-memory (OOM) and computational* issues by the significant reduction of memory and time consumption with respect to the latest scaling algorithms. Moreover, we have shown on the APPNP algorithm that MBSA-NN can scale other graph convolution networks based on PPR. We have also shown that MBSA could be used in the inference part of scaling algorithms. For instance, using it in PPRGO results in a decrease in both computation time and memory consumption.

In the next part of the thesis, we have proposed a novel minimization problem for semi-supervised learning that can be applied to both graph-structured and non-graph based datasets. We have provided an explicit solution to the problem, leading to a new linear framework called *Graph diffusion & PCA*. This framework allows to overcome the *Curse of dimensionality*, through the use of reorganized PCA, and the *Binary edges*, by considering the covariance matrix, which are both common issues in algorithms base on Laplacian regularizsation loss. Furthermore, we have described how MBSA algorithm might scale GDPCA, and we experimentally demonstrated that the scaled version of GDPCA (GDPCA-MBSA) maintains the same level of accuracy as the default GDPCA. Note that we have demonstrated the impact of these improvements in experiments on several datasets with and without an underlying graph structure. We have also compared it to state-of-the-art algorithms and showed that GDPCA, GDPCA-MBSA clearly outperforms the other linear graph-based diffusion algorithms in terms of accuracy. As for the comparison with neural networks, the experiments showed that the performance (accuracy) are similar, while GDPCA has a significantly lower computational time in addition to providing an explicit solution.

In the following part of the thesis, we have proposed a graph-based SSL (I)/(T) framework created by embedding PRSSL in generative model VAE. Based on the experimental findings, we have demonstrated that the generative model application for PRSSL can be used to understand the classification results, which may be utilized to address the *Explainability issue*, in addition to improving label spreading. We have also shown that GenPR significantly and consistently outperforms all other algorithms on every dataset and requires less number of PageRank PowerIteration steps. Also, we have embedded MBSA into GenPR, which can be executed in both the transductive and inductive training modes in batch training. Moreover, GenPR-MBSA demonstrates

lower computation complexity than the other competing methods with keeping a high performance (accuracy). Furthermore, we have shown that GenPR and GenPR-MBSA support training with or without default graph structure in the datasets, which allows applying these frameworks to avoid of *Versatility limitations*.

Finally, we have shown that the problem of label scarcity in data gathered from industrial equipment under working conditions is addressed by generating labels via an efficient SSL algorithm (PRPCA). Its outcomes are then fed into the genetic programming approach for symbolic regression (GPSR) based algorithm ZGP, which provides interpretable predictions expressed by a mathematically explicit formula. The working of the two algorithms has been briefly explained, and their joint use is described as the PaZoe framework. It has been shown that the use of this stacked framework provides a combined performance which overcomes the two algorithms individually. These results have been obtained on realistic data, partly generated for this purpose with industrially graded equipment, partly on sensor data available from the public domain. We have observed that similarly to other SSL algorithms (like LP, GCN, KNN), PaZoe does not assume any kind of data distribution (or even requires the data to be i.i.d.), while it performs better than those.

We conclude that, compared to the most advanced state-of-the-art GB-SSL methods, the algorithms and frameworks we have presented in this study offer good performance in practice and considerably lower memory and computational complexity on a variety of types of data. Additionally, all of the algorithms, frameworks and newly collected datasets that were suggested in this thesis are freely accessible and can be used in future investigations. In the future, we consider investigating an extension of our Markov-Batch Stochastic Approximation algorithm by applying a more complex batch selection strategy as in the linear case for MBSA in neural networks MBSA-NN. In particular, we are planning to apply the recurrent batch of nodes updating strategy for MBSA, which we hope can be adapted for memory and time complexity reduction in recurrent neural networks.

Bibliography

[1] S. Karczmarz, "Angenaherte auflosung von systemen linearer glei-chungen," *Bull. Int. Acad. Pol. Sic. Let., Cl. Sci. Math. Nat.*, pp. 355–357, 1937.

[2] K. Avrachenkov, V. S. Borkar, and K. Saboo, "Distributed and asynchronous methods for semi-supervised learning," in *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 2016, pp. 34–46.

[3] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," *CMU Technical report*, 2002.

[4] K. Avrachenkov, A. Mishenin, P. Gonçalves, and M. Sokol, "Generalized optimization framework for graph-based semi-supervised learning," in *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 2012, pp. 966–974.

[5] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of machine learning research*, vol. 7, no. Nov, pp. 2399–2434, 2006.

[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations*. ICLR, 2017.

[7] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[8] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5453–5462.

[9] T. Joachims, "Transductive inference for text classification using support vector machines," in *Icml*, vol. 99, 1999, pp. 200–209.

[10] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[11] J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov, "Neighbourhood components analysis," *Advances in neural information processing systems*, vol. 17, 2004.

[12] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in neural information processing systems*, 2014, pp. 3581–3589.

[13] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Icml*, 2011.

[14] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," *arXiv preprint arXiv:1810.05997*, 2018.

[15] K. Avrachenkov, A. Boisbunon, and M. Kamalov, "Graph diffusion & pca framework for semi-supervised learning," in *The 15th Learning and Intelligent Optimization (LION)*, 2021.

[16] M. Kamalov, A. Boisbunon, C. Fanara, I. Grenet, and J. Daeden, "Pazoe: classifying time series with few labels," in *2021 29th European Signal Processing Conference (EUSIPCO)*. IEEE, 2021, pp. 1561–1565.

[17] E. Buchnik and E. Cohen, "Bootstrapped graph diffusions: Exposing the power of nonlinearity," in *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*, 2018, pp. 8–10.

[18] M. Kamalov and K. Avrachenkov, "Genpr: Generative pagerank framework for semi-supervised learning on citation graphs," in *Conference on Artificial Intelligence and Natural Language*. Springer, 2020, pp. 158–165.

[19] J. W. Vaughan, "Making better use of the crowd: How crowdsourcing can advance machine learning research." *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 7026–7071, 2017.

[20] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[21] D. Zhou and C. J. Burges, "Spectral clustering and transductive learning with multiple views," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 1159–1166.

[22] X. Mai and R. Couillet, "Consistent semi-supervised graph regularization for high dimensional data," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 3074–3100, 2019.

[23] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," ser. Proceedings of Machine Learning Research, vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 40–48.

[24] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[25] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 639–655.

[26] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," *arXiv preprint arXiv:1803.07294*, 2018.

[27] H. Lu, S. H. Huang, T. Ye, and X. Guo, "Graph star net for generalized multi-task learning," 2019.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[29] R. Mises and H. Pollaczek-Geiringer, "Praktische verfahren der gleichungsauflösung." *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 9, no. 1, pp. 58–77, 1929.

[30] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph wavelet neural network," in *International Conference on Learning Representations*, 2019.

[31] A. Grover, A. Zweig, and S. Ermon, "Graphite: Iterative generative modeling of graphs," in *International conference on machine learning*. PMLR, 2019, pp. 2434–2444.

[32] X. Zhang, Y. He, N. Brugnone, M. Perlmutter, and M. Hirn, "Magnet: A neural network for directed graphs," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[33] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the 2nd International Conference on Learning Representations*. ICLR, 2013.

[34] H. Wang, C. Zhou, X. Chen, J. Wu, S. Pan, and J. Wang, "Graph stochastic neural networks for semi-supervised learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 839–19 848, 2020.

[35] S. Pal, S. Malekmohammadi, F. Regol, Y. Zhang, Y. Xu, and M. Coates, "Non parametric graph learning for bayesian graph neural networks," in *Conference on uncertainty in artificial intelligence*. PMLR, 2020, pp. 1318–1327.

[36] J. Ma, W. Tang, J. Zhu, and Q. Mei, "A flexible generative framework for graph-based semi-supervised learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[37] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," *Advances in neural information processing systems*, vol. 31, 2018.

[38] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 9267–9276.

[39] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5453–5462.

[40] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *international conference on machine learning*. PMLR, 2019, pp. 21–29.

[41] S. Luan, M. Zhao, X.-W. Chang, and D. Precup, "Break the ceiling: Stronger multi-scale deep graph convolutional networks," *Advances in neural information processing systems*, vol. 32, 2019.

[42] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 338–348.

[43] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1725–1735.

[44] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," *arXiv preprint arXiv:1907.10903*, 2019.

[45] A. Hasanzadeh, E. Hajiramezanali, S. Boluki, M. Zhou, N. Duffield, K. Narayanan, and X. Qian, "Bayesian graph neural networks with adaptive connection sampling," in *International conference on machine learning*. PMLR, 2020, pp. 4094–4104.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[47] ——, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[49] H. Wang, M. He, Z. Wei, S. Wang, Y. Yuan, X. Du, and J.-R. Wen, "Approximate graph propagation," *arXiv preprint arXiv:2106.03058*, 2021.

[50] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J.-R. Wen, "Scalable graph neural networks via bidirectional propagation," *arXiv preprint arXiv:2010.15421*, 2020.

[51] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann, "Scaling graph neural networks with approximate pagerank," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2464–2473.

[52] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in

Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, 2018, pp. 974–983.

[53] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[54] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 942–950.

[55] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen, "Decoupling the depth and scope of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[56] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph sampling based inductive learning method," in *International Conference on Learning Representations*, 2020.

[57] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.

[58] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[59] J. Klicpera, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," *arXiv preprint arXiv:1911.05485*, 2019.

[60] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[61] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," *Advances in neural information processing systems*, vol. 32, 2019.

[62] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec, "Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings," in *International Conference on Machine Learning*. PMLR, 2021, pp. 3294–3304.

[63] S. Park, W. Lee, B. Choe, and S.-G. Lee, "A survey on personalized pagerank computation algorithms," *IEEE Access*, vol. 7, pp. 163 049–163 062, 2019.

[64] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989, vol. 23.

[65] K. Du and X. Sun, "A doubly stochastic block gauss-seidel algorithm for solving linear equations," *arXiv preprint arXiv:1912.13291*, 2019.

[66] W. M. WU, "Convergence of the randomized block gauss-seidel method," 2018.

[67] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International conference on machine learning*. PMLR, 2019, pp. 6861–6871.

[68] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[69] I. Ipsen and R. M. Wills, "Analysis and computation of google's pagerank," in *7th IMACS international symposium on iterative methods in scientific computing, Fields Institute, Toronto, Canada*, vol. 5, no. 8, 2005.

[70] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in neural information processing systems*, 2004, pp. 321–328.

[71] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[72] R. Baeza-Yates and G. Navarro, "Block addressing indices for approximate text retrieval," *Journal of the American Society for Information Science*, vol. 51, no. 1, pp. 69–82, 2000.

[73] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[74] A. Grover and 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.

[75] S. Liang, Y. Wang, C. Liu, L. He, L. Huawei, D. Xu, and X. Li, "Engn: A high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1511–1525, 2020.

[76] J. Chen, G. Lin, J. Chen, and Y. Wang, "Towards efficient allocation of graph convolutional networks on hybrid computation-in-memory architecture," *Science China Information Sciences*, vol. 64, no. 6, pp. 1–14, 2021.

[77] T. Baruah, K. Shivdikar, S. Dong, Y. Sun, S. A. Mojumder, K. Jung, J. L. Abellán, Y. Ukidave, A. Joshi, J. Kim *et al.*, "Gnnmark: A benchmark suite to characterize graph neural network training on gpus," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2021, pp. 13–23.

[78] I. Triguero, S. García, and F. Herrera, "Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study," *Knowledge and Information systems*, vol. 42, no. 2, pp. 245–284, 2015.

[79] A. Boisbunon, C. Fanara, I. Grenet, J. Daeden, A. Vighi, and M. Schoenauer, "Zoetrope genetic programming for regression," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2021.

[80] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using pagerank vectors," in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 2006, pp. 475–486.

[81] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *International conference on database theory*. Springer, 1999, pp. 217–235.

[82] V. S. Borkar, *Stochastic approximation: a dynamical systems viewpoint*. Springer, 2009, vol. 48.

[83] S. Jubertie, K. Peou, G. Quintin, and F. Dupros, "Portability across arm neon and sve vector instruction sets using the nsimd library: a case study on a seismic spectral-element kernel," in *HPCS 2020*, 2021.

[84] M. G. Kendall, "The treatment of ties in ranking problems," *Biometrika*, vol. 33, no. 3, pp. 239–251, 1945.

[85] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," *arXiv preprint arXiv:1707.03815*, 2017.

[86] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015.

[87] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *arXiv preprint arXiv:1706.02216*, 2017.

[88] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22 118–22 133, 2020.

[89] G. Thoppe and V. Borkar, "A concentration bound for stochastic approximation via alekseev's formula," *Stochastic Systems*, vol. 9, no. 1, pp. 1–26, 2019.

[90] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, no. 2, pp. 127–163, 2000.

[91] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[92] E. Bair, T. Hastie, D. Paul, and R. Tibshirani, "Prediction by supervised principal components," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 119–137, 2006.

[93] R. Johnson and T. Zhang, "Graph-based semi-supervised learning and spectral kernel design," *IEEE Transactions on Information Theory*, vol. 54, no. 1, pp. 275–288, 2008.

[94] A. Ritchie, C. Scott, L. Balzano, D. Kessler, and C. S. Sripada, "Supervised principal component analysis via manifold optimization," in *2019 IEEE Data Science Workshop (DSW)*. IEEE, 2019, pp. 6–10.

[95] F. Roli and G. L. Marcialis, "Semi-supervised pca-based face recognition using self-training," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2006, pp. 560–568.

[96] C. Walder, R. Henao, M. Mørup, and L. Hansen, *Semi-Supervised Kernel PCA*, ser. IMM-Technical Report-2010-10. Technical University of Denmark, DTU Informatics, Building 321, 2010.

[97] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[98] C. Ding and X. He, "K-means clustering via principal component analysis," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 29.

[99] R. M. Freund, "Quadratic functions, optimization, and quadratic forms," 2004.

[100] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

[101] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice hall Englewood Cliffs, NJ, 1989, vol. 23.

[102] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.

[103] S. Rifai, Y. N. Dauphin, P. Vincent, Y. Bengio, and X. Muller, "The manifold tangent classifier," *Advances in neural information processing systems*, vol. 24, pp. 2294–2302, 2011.

[104] E. Fix, *Discriminatory analysis: nonparametric discrimination, consistency properties*. USAF school of Aviation Medicine, 1951.

[105] M. Zima, "A theorem on the spectral radius of the sum of two operators and its application," *Bulletin of the Australian Mathematical Society*, vol. 48, no. 3, pp. 427–434, 1993.

[106] O. Rojo, R. Soto, and H. Rojo, "Bounds for the spectral radius and the largest singular value," *Computers & Mathematics with Applications*, vol. 36, no. 1, pp. 41–50, 1998.

[107] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[108] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[109] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[110] I. Masic, M. Miokovic, and B. Muhamedagic, "Evidence based medicine–new approaches and challenges," *Acta Informatica Medica*, vol. 16, no. 4, p. 219, 2008.

[111] S. J. Day and D. G. Altman, "Blinding in clinical trials and other studies," *Bmj*, vol. 321, no. 7259, p. 504, 2000.

[112] C. Brezinski and M. Redivo-Zaglia, "The pagerank vector: properties, computation, approximation, and acceleration," *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 2, pp. 551–575, 2006.

[113] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[114] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan, "uWave: Accelerometer-based personalized gesture recognition and its applications," *Pervasive and Mobile Computing*, vol. 5, no. 6, pp. 657–675, 2009.

[115] J. Guna, I. Humar, and M. Pogačnik, "Intuitive gesture based user identification system," in *35th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2012, pp. 629–633.

[116] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations (ICLR)*, 2017.