UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Graph Neural Networks contributions and advancements

Thales de Oliveira Gonçalves

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Thales de Oliveira Gonçalves

Graph Neural Networks contributions and advancements

Thesis submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Luis Gustavo Nonato

USP – São Carlos
July 2024

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

d635g

de Oliveira Gonçalves, Thales
Graph Neural Networks contributions and advancements / Thales de Oliveira Gonçalves; orientador Luis Gustavo Nonato. -- São Carlos, 2024.
92 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2024.

1. Graph Neural Networks. 2. Extreme Learning Machine. 3. Dynamic Graphs. I. Nonato, Luis Gustavo, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Thales de Oliveira Gonçalves

Contribuições e avanços em Redes Neurais em Grafos

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Luis Gustavo Nonato

USP – São Carlos
Julho de 2024

This work is dedicated to those who would rather stick
to the questions they cannot answer,
than to the answers they cannot question.

ACKNOWLEDGEMENTS

First of all, I would like to thank my parents, who taught me my greatest personal values and gave me every possible opportunity to develop myself both educationally and in my inner virtues. Also my sisters and friends, who always motivate me and try to shorten geographical distances between us which now exists.

I would also like to thank Luis Gutavo Nonato, it is only being supported by such great people that we manage to make ourselves bigger. In addition, thanks to Cláudio T. Silva for receiving me during my 1-year collaboration program abroad and allowing me to get in touch with great researchers. I hope to keep collaborating with both of you after my Ph.D. period.

Furthermore, thanks to the professors of the examination committee, which accepted to read and evaluate the work developed during the Ph.D. period. It is of large importance that such great names can make contributions and bring different perspectives to this dissertation.

I wouls also like to thank all members of VICG, GIVA, VIDA and UFES, which might have helped me in any way, even if simple. Also thanks to USP and NYU for providing the necessary structure for the campuses maintenance.

Last but not least, I would like to thank FAPESP, process numbers 2018/24516-0 and 2021/12013-6, without which I would be totally unable to develop my research project.

“Intelligence is that faculty of mind
by which order is perceived in a situation
previously considered disordered.”
(R. W. Young”)

RESUMO

GONÇALVES, T. O. **Contribuições e avanços em Redes Neurais em Grafos**. 2024. 92 p.Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

A aplicação de Redes Neurais no contexto de grafos é um campo de crescente interesse nos últimos anos. Uma das principais razões para isso é o grande número de aplicações do mundo real que dão origem à produção de dados tendo este objeto matemático como estrutura, tais como sistemas de recomendação em redes sociais, moléculas em química, planeamento urbano, análise de esportes, etc. No entanto, além dos desafios comuns envolvidos no projeto de uma solução clássica de *Machine Learning* para lidar com problemas do mundo real (e.g. *overfitting*, desequilíbrio de classes, esparsidade, busca de hiperparâmetros), existem alguns obstáculos adicionais que precisam ser tratados ao lidar com problemas de *Machine Learning* em grafos. Nesta tese, apresentamos as contribuições propostas em relação a uma série de desafios recentes de Redes Neurais em Grafos. Mais especificamente, primeiro propomos *Extreme Learning Machine to Graph Convolutional Networks* (ELM-GCN), uma extensão da teoria de ELM para ser aplicada a GCNs, um modelo de Rede Neural projetado para operar em grafos. Esta extensão dá origem a um algoritmo de treinamento analítico com bases teóricas sólidas e que é capaz de atingir uma precisão semelhante aos métodos concorrentes, mas reduzindo consideravelmente o tempo de treinamento. Posteriormente, propomos uma nova arquitetura de GNN para ser aplicada em grafos dinâmicos, i.e. grafos nos quais seus elementos (nós, arestas e vetores de características) mudam ao longo do tempo. Essa formulação deu origem ao *Graph Neural Networks for Valuing Soccer Players* (GNN-VSP), uma metodologia de avaliação de atletas de futebol baseada em um algoritmo de explicabilidade capaz de considerar a interação da equipe. Por fim, mostramos as linhas futuras que o autor pretende seguir em sua carreira de pesquisador.

Palavras-chave: Redes Neurais em Grafos, *Extreme Learning Machine*, Grafos Dinâmicos.

ABSTRACT

GONÇALVES, T. O. **Graph Neural Networks contributions and advancements**. 2024. 92 p.Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

Applying Neural Networks to the context of graphs is a field of increasing interest in recent years. One of the main reasons for that is the large number of real-world applications that give rise to data being produced with this mathematical object as the underlying structure, like social networks recommendation systems, molecules in chemistry, urban planning, sports analytics, etc. However, besides the common challenges involved in designing a classic Machine Learning solution for tackling real-world issues (e.g. overfitting, class imbalance, sparsity, hyperparameter search, etc), there are some additional obstacles that need to be overcome when dealing with Machine Learning problems on graphs. In this dissertation, we present the proposed contributions with respect to a number of the recent Graph Neural Networks challenges. More specifically, first we propose Extreme Learning Machine to Graph Convolutional Networks (ELM-GCN), an extension of the ELM theory to be applied to GCNs, a Neural Network model designed to operate on graphs. This extension gives rise to an analytical training algorithm that comes with solid theoretical foundations and that is able to reach an accuracy similar to competing methods, but reducing the training time considerably. Afterward, we propose a novel GNN architecture to be applied in dynamic graphs, i.e. graphs in which its elements (nodes, edges, and feature vectors) change over time. This formulation led to Graph Neural Networks for Valuing Soccer Players (GNN-VSP), a methodology for scoring soccer athletes based on an explainability algorithm that is able to account for the team interplay. Finally, we show the future lines that the author plans to follow in his research career.

Keywords: Graph Neural Networks, Extreme Learning Machine, Dynamic Graphs.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

CNN	Convolutional Neural Network
CTDG	Continuous-Time Dynamic Graph
DL	Deep Learning
DTDG	Discrete-Time Dynamic Graph
ELM	Extreme Learning Machine
ELM-GCN	Extreme Learning Machine to Graph Convolutional Networks
GCN	Graph Convolutional Network
GFT	Graph Fourier Transform
GNN	Graph Neural Networks
GNN-VSP	Graph Neural Networks for Valuing Soccer Players
GSP	Graph Signal Processing
IGFT	Inverse Graph Fourier Transform
MLP	Multi-Layer Perceptron
MSE	Mean Square Error
RELM	Regularized Extreme Learning Machine
XAI	EXplainable Artificial Intelligence

LIST OF SYMBOLS

a — Attention Coefficient

A — Graph Adjacency Matrix

$\tilde{A}$ — Graph Adjusted Adjacency Matrix

D — Graph Degree Matrix

$\tilde{D}$ — Graph Adjusted Degree Matrix

E — Set of Graph Edges

F — Dimension of the Feature Space

g — Spectral Kernel

G — Graph

$\mathscr{G}$ — Graph Endowed with Node Features

h — Filtered Signal (Graph Domain)

H — Hidden Dimension

$\hat{h}$ — Filtered Signal (Spectral Domain)

I — Identity Matrix

$\mathscr{I}$ — Importance given by the Explainability Algorithm

K — Order of the Chebyshev Polynomials

L — Graph Laplacian Matrix

$\bar{L}$ — Normalized Laplacian Matrix

N — Number of Graph Nodes

P — Graph Pooling

$\mathbb{R}$ — Set of Real Numbers

s — Shots to Goal

T — Time Window

$\mathbb{T}$ — Total Played Time over the Competition

U — (Normalized) Laplacian Eigenvectors Matrix

V — Set of Graph Nodes

x — Graph Signal (Graph Domain)

$\hat{x}$ — Graph Signal (Spectral Domain)

X — (Graph Node) Features Matrix

α — LeakyReLU parameter

γ — Regularization Parameter

Γ — Dynamic Graph with Time-Evolving Node Features

λ — Laplacian Eigenvalue

Λ — (Normalized) Laplacian Eigenvalues Matrix

θ — Learnable Parameter

Θ — Learnable Parameter Matrix

σ — Layer Activation Function

ω — Graph Edge Weights

CONTENTS

CHAPTER

1

INTRODUCTION

1.1 Graph Neural Networks Overview

Applying Neural Networks to the context of graphs has long been of great interest due to the potential applications (KÜCHLER; GOLLER, 1996; SCARSELLI *et al.*, 2008). However, it was only a few years ago that this area has emerged as a main topic in Machine Learning.

A graph is a powerful and versatile structure that arises naturally from many real-world applications. In a simple mathematical abstraction, a graph can be understood as a set of entities (nodes) and relationships between these entities (edges), enabling complex systems such as social networks (ARAZZI *et al.*, 2023), molecules (WANG; LI; FARIMANI, 2023), street maps (LI; ZHU, 2021), athletes interplay (XENOPOULOS; SILVA, 2021), documents citations (ACHARYA; ZHANG, 2020), etc, to be modeled as graphs.

Data can be associated to the elements of a graph, making viable the use of Machine Learning tools to extract non-trivial knowledge, perform classifications, and even group parts of a graph based on data similarity. For instance, feature vectors can be associated to the graph nodes (e.g a citation network in which the data is related to the content of each document), graph edges (e.g. the data is the message between two users in a social network) and/or the whole graph (e.g. solubility of a molecule). Among typical Machine Learning tasks performed on graph structures are node-level prediction (e.g. identify the genre of new movies belonging to a streaming platform), edge-level prediction (e.g. forecast crime incidence in the streets of a city), link prediction (e.g. recommend songs in a bipartite user-song graph) or graph-level prediction (e.g. predict how many goals a team will make in a match) (ZHANG *et al.*, 2019).

Analogously to the classical Convolutional Neural Networks (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) designed to operate on regular grid structures like images and videos, learning mechanisms on graphs typically aim to build local operators that aggregate information from the

nodes and their neighborhoods. However, one of the main differences between graph and classic convolution is that the graph demands flexibility with respect to the number and ordering of the neighbors, as graph nodes can have an arbitrary number of neighbors with no canonical order.

Besides of the common challenges involved in designing a classic Machine Learning solution for a real-world problem (e.g. overfitting, class imbalance, sparsity, hyperparameter search, etc), there are some additional issues that need to be tackled when dealing with Machine Learning problems on graphs. In Section 1.2 we discuss the challenges that we address in this dissertation.

1.2 Some Challenges in Graph Neural Networks

Deep Graph Neural Networks and the Oversmoothing Effect. The term "over-smoothing" in the context of Deep Learning is often used to describe a situation where a neural network model becomes too generalized, losing sensitivity to fine-grained details in the data. This phenomenon is commonly associated with graph-based neural networks, in particular Message-Passing Graph Neural Networks. Oversmoothing can occur when the model aggregates information from neighboring nodes iteratively, assigning similar representations to all nodes (LI; HAN; WU, 2018), effectively smoothing out the differences between them.

Kipf and Welling (2016a) when introducing the Graph Convolutional Network (GCN) model (discussed in detail in Section 2.2), empirically observed that the accuracy of the GCN dropped when stacking multiple layers. Moreover, they also showed that employing residual connections (HE *et al.*, 2016) – which helps the training process in non-graph scenarios for alleviating the Vanishing Gradient problem – did not properly work for GCNs.

Li, Han and Wu (2018) show theoretically that oversmoothing is guaranteed to asymptotically happen when the number of layers grows indefinitely, making data representation converge to the same vector, hampering downstream task performance.

Additionally to the result of Li, Han and Wu (2018), Oono and Suzuki (2020) showed that GNNs lose expressive power exponentially with the depth of the network. However, the neighborhood aggregation mechanism responsible for the Oversmoothing is beneficial for shallow networks (BARANWAL; FOUNTOULAKIS; JAGANNATH, 2021; BARANWAL; FOUNTOULAKIS; JAGANNATH, 2022) and thus there must exist an ideal number of stacking layers that leads to optimal performance (KERIVEN, 2022; WU *et al.*, 2022). As a result, several works rely on shallow GNN architectures for tackling real-world issues, being the 2-layered model largely used (JIN *et al.*, 2020; ZHAO *et al.*, 2019; YAN; XIONG; LIN, 2018).

Dynamic Graphs. Several works design Graph Neural Network architectures to work on static graphs, that is, a graph or set of graphs that do not change over time (KIPF; WELLING, 2016a; VELIČKOVIĆ *et al.*, 2017; YING *et al.*, 2021). However, it is not uncommon that a real-world application gives rise to dynamic graphs, which are time-evolving graphs endowed with time-dependent features. Some examples of problems being naturally modeled as dynamic graphs are: social networks in which new users join or delete accounts, networks where new interactions between nodes show up over time; and a soccer match in which players can be substituted or expelled, players may change their interplay during the match, and also their personal positioning and play style may vary.

A possible manner of categorizing dynamic graph problems is the distinction between Continuous-Time Dynamic Graphs (CTDG) and Discrete-Time Dynamic Graphs (DTDG) (KAZEMI *et al.*, 2020). The former can be defined as an initial graph and a sequence of changes indexed by a real-valued time variable, e.g. node i added at time t_1 and edge j,k deleted at time t_2 (NGUYEN *et al.*, 2018; CHANG *et al.*, 2020). In contrast, DTDG is a discrete temporal sequence of graphs and features vector snapshots (MICHELI; TORTORELLA, 2022; SHARMA *et al.*, 2023; ZHANG *et al.*, 2023).

While both CTDG and DTDG approaches are valid and application-dependent, DTDG stands out as a popular choice as it enables the use of known architectures designed for static graphs (KAZEMI *et al.*, 2020). For instance, this discrete modeling proves advantageous in scenarios where a static graph is derived by summarizing the temporal sequence of snapshots, and subsequently, a Graph Neural Network tailored for static graphs can be applied.

In this context, some works (LIBEN-NOWELL; KLEINBERG, 2003; HISANO, 2018) tackle this issue simply by taking the sum (or union) of the temporal sequence of adjacency matrices. Moreover, an extension of this approach is to summarize the graph by giving more weight to more recent snapshots given by a decaying kernel function (IBRAHIM; CHEN, 2015; AHMED *et al.*, 2016).

Irrespective of the chosen approach, the exploration of dynamic graphs to address real-world problems remains of fundamental importance. There is still ample room for growth in this field, as the number of existing dynamic graph network alternatives is considerably reduced when compared to the established literature on Graph Neural Networks for static graphs.

1.3 Scope of this Dissertation

This dissertation describes the author's contributions with respect to the Graph Neural Networks challenges described in Section 1.2. More specifically, since the Oversmoothing problem pushes for GNNs architectures with a few hidden layers, we propose an analytical training algorithm suitable for a Graph Neural Network model with 2 layers, described in Chapter 3. In Chapter 4 we propose a novel GNN architecture to handle dynamic graphs. This architecture is built upon a temporal summarizing scheme which lets the model learn the aggregation by an attention mechanism. Moreover, this formulation led to a methodology for scoring soccer athletes based on a graph explainability algorithm that, unlike other player valuing mechanisms, is able to also account for the team interplay.

CHAPTER

2

BASIC CONCEPTS

In this chapter, we discuss in details the main baseline works for this dissertation. More specifically, Section 2.1 states basic concepts about Graph Signal Processing (GSP), which is of fundamental importance to discuss Graph Convolutional Networks (GCN) (KIPF; WELLING, 2016a) in Section 2.2. Section 2.3 presents Extreme Learning Machine (ELM) (HUANG; ZHU; SIEW, 2006) approach that is the basis of the proposed training mechanism.

2.1 Graph Signal Processing (GSP)

Let $G = (V, E)$ be a graph, where V is the set of N vertices (or nodes) and E is the set of edges, where each edge in E connects two vertices in V. Let $A \in \mathbb{R}^{N \times N}$ be the (optionally weighted) adjacency matrix of G, i.e., a matrix such that each entry $A_{i,j}$ encodes the strength of the connection between the j-th and the i-th node, being $A_{i,j} = 0$ if and only if nodes i and j are not connected. Note that: 1) unweighted graphs are special cases of weighted graphs where all weights are equal to 1; and 2) a graph G is undirected if and only if A is symmetric. We will assume G is an undirected graph, unless explicitly stated the contrary.

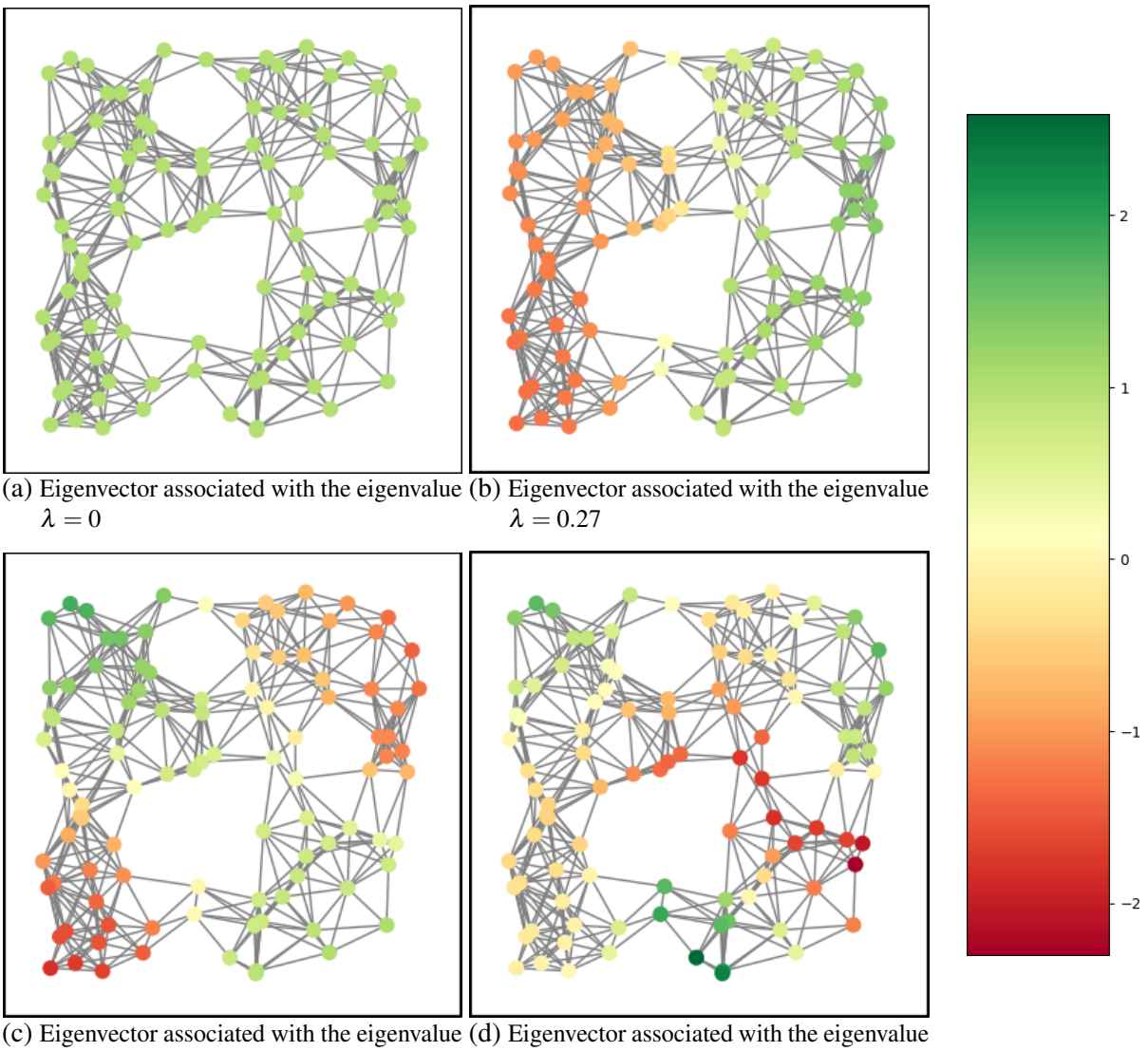
A graph signal is a function $x : V \to \mathbb{R}$, i.e., a mapping that associates a real number to each node. Moreover, by fixing an arbitrary ordering of the N graph vertices, there is an isomorphism between the space of graph signals and $\mathbb{R}^N$. In other words, we can interpret a particular N-dimensional vector as a graph signal and vice-versa, which is especially important for viewing eigenvectors of matrices derived from the graph topology as graph signals, as we show further.

The degree matrix of G is $D = \text{diag}(\sum_j A_{i,j})$, a diagonal matrix such that each entry $D_{i,i}$ is the sum of the edge weights with an end in the i-th node. Note that if G in unweighted, then $D_{i,i}$ encodes the number of neighbors of the i-th node.

A matrix that plays an important role in GSP is the Laplacian matrix $L = D - A$. One of its interesting properties is that there are as many null eigenvalues associated with L as there are connected components in the graph. In particular, for any graph G, 0 is always an eigenvalue of L associated with the eigenvector $\mathbf{1} = (1, 1, \ldots, 1) \in \mathbb{R}^N$ (STANKOVIC *et al.*, 2020).

Another interesting property of the laplacian is that L is a positive semidefinite symmetric matrix. Besides, its eigenvectors' smoothness depends on the associated eigenvalues, i.e., the eigenvector's entries, viewed as a graph signal, tend to change more quickly their value in adjacent nodes if the corresponding eigenvalue is larger (this property is illustrated in Figure 1). In particular, recall that the (non-oscillatory) $\mathbf{1}$ eigenvector is associated with the null eigenvalue.

Figure 1 – Four distinct eigenvectors of the graph laplacian matrix and its associated eigenvalues. Eigenvectors' smoothness is given by its associated eigenvalues.



(a) Eigenvector associated with the eigenvalue $\lambda = 0$

(b) Eigenvector associated with the eigenvalue $\lambda = 0.27$

(c) Eigenvector associated with the eigenvalue $\lambda = 0.75$

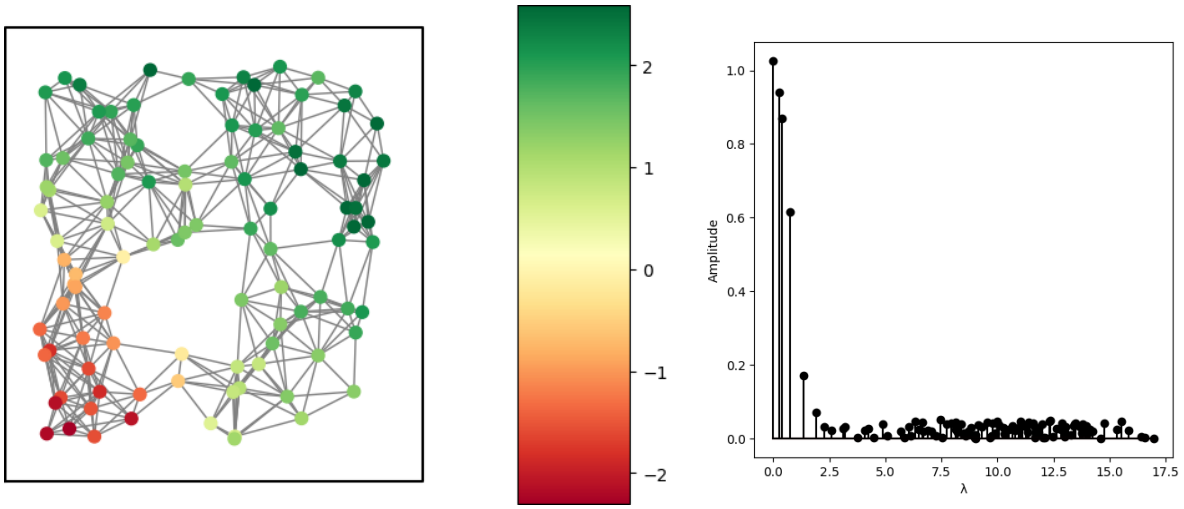
(d) Eigenvector associated with the eigenvalue $\lambda = 1.89$

Now consider the spectral decomposition of the laplacian matrix $L = U\Lambda U^T$, where $U, \Lambda \in \mathbb{R}^{N\times N}$ are its (orthonormal) matrix with column eigenvectors and diagonal matrix with real non-negative eigenvalues, respectively. Since the set of eigenvectors U of L forms an orthonormal basis of the space of graph signals and each eigenvector oscillates according to the corresponding eigenvalue (which can be seen as frequencies), it is natural to interpret the eigenvectors as the Fourier basis in graph context. That is, inner products between some graph signal x and each of the eigenvectors of the Laplacian matrix should result in the Fourier coefficients associated with frequencies given by the respective eigenvalues.

Figure 2 depicts an arbitrary graph signal in the graph of Figure 1. Each Graph Fourier coefficient (Figure 2b) is associated with an eigenvalue (x axis in Figure 2b) of the laplacian. Therefore, one can write the Graph Fourier Transform (GFT) of a graph signal x as $\hat{x} = U^T x$ and the Inverse Graph Fourier Transform (IGFT) accordingly as $x = U\hat{x}$ (STANKOVIC *et al.*, 2020), where $x, \hat{x}$ are vectors from $\mathbb{R}^N$ stated as column matrices.

Another important notion in GSP is the convolution operation between two graph signals x, y, which in the non-graph continuous case is defined as $(x * y)(t) = \int_{-\infty}^{\infty} x(t-\tau)y(\tau)d\tau$. However, it is not possible to directly extend such a definition to the graph context as the shift operation $x(t-\tau)$ is not defined in this scenario. Thus, this operation is defined in terms of the Convolution Theorem, which states that the convolution operation in the spatial domain matches the product operation in the frequency domain, as precisely defined as

Figure 2 – Graph signal represented both in graph and spectral domains. The Fourier coefficients can be used to reconstruct the original graph signal with respect to the Fourier basis (the Laplacian eigenvectors).



(a) Graph signal (graph domain)

(b) Graph signal (spectral domain)

$$\begin{aligned} x * y &= \text{IGFT}\Big(\text{GFT}(x) \odot \text{GFT}(y)\Big) \\ &= \text{IGFT}\Big(\hat{x} \odot \hat{y}\Big) \\ &= U\,\text{diag}(\hat{x})\,\hat{y}\,, \end{aligned} \tag{2.1}$$

where: $\hat{x}, \hat{y}$ are the GFT of x, y respectively, seen as column matrices; and $\odot$ is the element-wise product.

Now suppose we want to filter a graph signal x by amplifying or attenuating each of its Fourier modes $\hat{x}$ given by a spectral kernel $g : \mathbb{R}^+ \to \mathbb{R}^+$ defined in the frequency domain as

$$\hat{y}(\lambda_i) = g(\lambda_i)\hat{x}(\lambda_i)\,, \tag{2.2}$$

where $\hat{y}$ is the GFT of the filtered signal y and each λ_i is one of the N eigenvalues of the laplacian matrix L. Note that since the kernel is sampled only in Λ, one can define $g(\Lambda)$ parametrized by $\theta \in \mathbb{R}^N$ and so Equation 2.2 can be written as

$$\hat{y}(\lambda_i) = \theta_i\,\hat{x}(\lambda_i)\,, \tag{2.3}$$

being θ_i the i-th coordinate of θ. This equation can further be compactly written for all eigenvalues as

$$\hat{y} = g_\theta(\Lambda)\hat{x}\,, \tag{2.4}$$

where $g_\theta(\Lambda)$ is the diagonal matrix $g_\theta(\Lambda) = \text{diag}(\theta)$.

Finally, recalling the GFT and IGFT definitions $\hat{x} = U^T x$ and $y = U\hat{y}$, Equation 2.4 becomes

$$y = U g_\theta(\Lambda) U^T x\,. \tag{2.5}$$

Equation 2.5 can be understood by a sequence of operations at the graph signal x to obtain the filtered graph signal y by the spectral kernel g parameterized by θ. The operations are as follows:

- Compute $\hat{x}$, the GFT of x (by pre-multiplying x by U^T);
- Make a pointwise product between $\hat{x}$ and θ (by multiplying the diagonal matrix $\text{diag}(\theta) = g_\theta(\Lambda)$ by the column matrix $\hat{x}$);
- Compute the IGFT to return to the graph domain (by pre-multiplying the filtered signal $\hat{y}$ by U)

Although fairly simple, Equation 2.5 requires the computation of eigenvectors and eigenvalues of the laplacian matrix L, which may be unfeasible for large graphs. Therefore one could approximate $g_\theta(\Lambda)$ by a truncated expansion in terms of the Chebyshev polynomials (HAMMOND; VANDERGHEYNST; GRIBONVAL, 2011). This expression can be written as

$$g_\theta(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}), \tag{2.6}$$

where: K is the maximum order of the Chebyshev polynomials (also the number of terms used in the truncation); $\theta' \in \mathbb{R}^K$ is a vector of Chebyshev coefficients, being θ'_k is the k-th coordinate of θ'; T_k is the k-th order Chebyshev polynomial; and $\tilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I$ (with λ_{max} being the maximum eigenvalue of Λ) is a linear mapping of the graph spectrum to be contained in the interval $[-1, 1]$, which is the domain of the Chebyshev polynomials. Note that θ are the filter coefficients in the spectral domain while θ' are the Chebyshev coefficients. Remind that Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ (for $k \geqslant 2$), with $T_0(x) = 1$ and $T_1(x) = x$.

Plugging Equation 2.6 in Equation 2.5, we obtain

$$\begin{aligned} y &= U g_\theta(\Lambda) U^T x \\ &\approx U \left(\sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \right) U^T x \\ &= \sum_{k=0}^{K} \theta'_k U T_k(\tilde{\Lambda}) U^T x \\ &= \sum_{k=0}^{K} \theta'_k T_k(U \tilde{\Lambda} U^T) x \\ &= \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x, \end{aligned} \tag{2.7}$$

where $\tilde{L} = U\tilde{\Lambda}U^T = \frac{2}{\lambda_{max}}L - I$.

For more detailed readings and deeper GSP theory construction, we refer to Sandryhaila and Moura (2013a), Sandryhaila and Moura (2013b), Sandryhaila and Moura (2014), Hammond, Vandergheynst and Gribonval (2011), Stankovic *et al.* (2020) and Col *et al.* (2017).

2.2 Graph Convolutional Network (GCN)

Convolutional Neural Networks (CNN) marked a new era in Machine Learning since its proposal (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Nevertheless, they only apply to regular grid domains like time series, images, and videos. GCN is a natural extension of this network paradigm to graphs.

Let $\bar{L} = I - D^{-1/2}AD^{-1/2}$ be the normalized laplacian matrix of an undirected graph G, where I is the identity matrix and, as stated in Section 2.1, A and D are the adjacency and degree matrices of G, respectively. Suppose a graph signal $x \in \mathbb{R}^N$ associated with the N nodes of G and consider the spectral decomposition $\bar{L} = U\Lambda U^T$ of $\bar{L}$, where U is the matrix whose columns correspond to the eigenvectors and Λ is a diagonal matrix with the corresponding eigenvalues of $\bar{L}$.

Kipf and Welling (2016a) consider the operation of a convolution layer as filtering of a graph signal x by a spectral kernel using the truncated expansion of Chebyshev polynomials as given by Equation 2.7. In terms of Neural Networks terminology, x, y can be seen respectively as the input and output, and θ'_k as the layer parameters. Moreover, the authors employ a sequence of simplifications to state the convolution layer precisely.

First, they note that a K-th order truncation of the Chebyshev polynomials expansion aggregate information about the up to K-th order neighborhood of each node, since it is a weighted sum of powers up to K of the normalized laplacian matrix. Hence, the authors consider a linear model ($K = 1$) for a single layer. In this way, they expect that overfitting problems can be alleviated on local neighborhood structures of the graph. Moreover, the information of higher order neighborhoods of each node can still be accessed by just stacking multiple layers, with also the advantage of not being limited to the explicit parameterization given by the Chebyshev polynomials. Thus, setting $K = 1$ in Equation 2.7 gives

$$\begin{aligned} y &\approx \big(\theta'_0 T_0(\tilde{L}) + \theta'_1 T_1(\tilde{L})\big)x \\ &= \big(\theta'_0 + \theta'_1 \tilde{L}\big)x\,. \end{aligned} \tag{2.8}$$

Considering that the larger eigenvalue of the normalized laplacian matrix is $\lambda_{max} \approx 2$ (all eigenvalues of $\bar{L}$ are in the interval $[0, 2]$) we have that

$$\begin{aligned} y &\approx \left(\theta'_0 + \theta'_1 \tilde{L}\right)x \\ &= \left(\theta'_0 + \theta'_1 \left(\frac{2}{\lambda_{max}}\bar{L} - I\right)\right)x \\ &\approx \left(\theta'_0 + \theta'_1(\bar{L} - I)\right)x \\ &= \left(\theta'_0 - \theta'_1 D^{-1/2} A D^{-1/2}\right)x. \end{aligned} \tag{2.9}$$

Then, aiming to reduce the number of learnable parameters, Kipf and Welling (2016a) define $\theta := \theta'_0 = -\theta'_1$. Thus, Equation 2.9 becomes

$$y \approx \theta\left(I + D^{-1/2} A D^{-1/2}\right)x. \tag{2.10}$$

Finally, the authors introduce the so-called renormalization trick: $I + D^{-1/2} A D^{-1/2} \rightarrow$ $\rightarrow \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ (KIPF; WELLING, 2016a), where $\tilde{A} = A + I$ and $\tilde{D} = \text{diag}(\sum_j \tilde{A}_{i,j})$. Note that by adding the identity matrix to A they virtually change the graph structure by adding self-loops in each node, which means that a vertex aggregates the information from its neighbors with their own. Therefore the convolution operation in a layer is considered as

$$y \approx \theta\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}\right)x, \tag{2.11}$$

which can be generalized to a F-dimensional graph signal $X \in \mathbb{R}^{N \times F}$ (i.e., X associates a vector of $\mathbb{R}^F$ to each of the N nodes) and, after applying the convolved signal to an activation function σ, we have that

$$\begin{aligned} Y = f(X,A) &= \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X\Theta\right) \\ &= \sigma\left(\hat{A}X\Theta\right), \end{aligned} \tag{2.12}$$

where $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$.

In Equation 2.12, it is considered that the layer gets as input a F-dimensional graph signal $X \in \mathbb{R}^{N \times F}$ and produces as output a C-dimensional graph signal $Y \in \mathbb{R}^{N \times C}$. Equivalently, one could understand X and Y as graph signals with F and C channels or feature maps, respectively. Matrix $\hat{A} \in \mathbb{R}^{N \times N}$ is responsible for the neighbor aggregation process and it encodes the structure of the considered graph, while $\Theta \in \mathbb{R}^{F \times C}$ is a matrix of learnable parameters which must be adjusted with a training algorithm.

Equation 2.12 defines a single convolution layer. A Graph Convolutional Network (GCN) is taken as a stack of convolution layers. In this way, a 2-layered GCN can be represented as

$$Y = \sigma_2\left(\hat{A}\ \sigma_1\left(\hat{A}X\Theta^{(1)}\right)\ \Theta^{(2)}\right), \tag{2.13}$$

where: σ_1, σ_2 are the activation functions of the first and the second layer, respectively; and $\Theta^{(1)} \in \mathbb{R}^{F\times H}, \Theta^{(2)} \in \mathbb{R}^{H\times C}$ are the matrices of learnable parameters of each layer. As in Multi-Layer Perceptron (MLP), the sizes of the input (F) and output (C) are defined by the problem, while the hidden dimension (H) remains as a hyperparameter.

In order to validate each step in their approximation scheme, Kipf and Welling (2016a) analyze the accuracy of each propagation model given by the intermediate steps in their simplifications, as can be seen in the first column of Table 1. The propagation models described in Table 1 are the ones defined by the respective equations, in particular, the sub-indices of Θ are the ones presented by θ'_k in Equations 2.7, and 2.9 with the only difference that while θ_k is a vector, Θ_k is a matrix of parameters. The authors apply the different propagation models in three different real-world datasets: Citeseer, Cora, and Pubmed (SEN *et al.*, 2008). We give more details about these datasets in Section 3.3. They consider a two-layered GCN with $H = 16$ hidden units and activation functions $\sigma_1(x) = \mathrm{ReLU}(x) = \max(0,x)$ and $\sigma_2(x_i) = \mathrm{softmax}(x_i) = e^{x_i}/\sum_j e^{x_j}$.

Table 1 – Comparison between different propagation models (KIPF; WELLING, 2016a). The convolution layer defined after the renormalization trick outperforms the other propagation models tested.

Description	Propagation Model	Citeseer	Cora	Pubmed
Chebyshev filter (Eq. 2.7, K=3)	$\sum_{k=0}^{3} T_k(\tilde{L})X\Theta_k$	69.8	79.5	74.4
Chebyshev filter (Eq. 2.7, K=2)	$\sum_{k=0}^{2} T_k(\tilde{L})X\Theta_k$	69.6	81.2	73.8
First order model (Eq. 2.9)	$X\Theta_0 + D^{-1/2}AD^{-1/2}X\Theta_1$	69.3	79.2	77.4
Single parameter (Eq. 2.10)	$(I + D^{-1/2}AD^{-1/2})X\Theta$	68.3	80.0	77.5
Renormalization trick (Eq. 2.11)	$\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X\Theta$	**70.3**	**81.5**	**79.0**
First-order term only	$D^{-1/2}AD^{-1/2}X\Theta$	68.7	80.5	77.8
Multi-Layer Perceptron	$X\Theta$	46.5	55.1	71.4

Table 1 shows that the approximations do not impact substantially the accuracy, being the model with renormalization the one with the better result. Also, Table 1 shows that the MLP model did not achieve comparable results, meaning that the proposed GCN is using the graph structure topology to enhance the predictions.

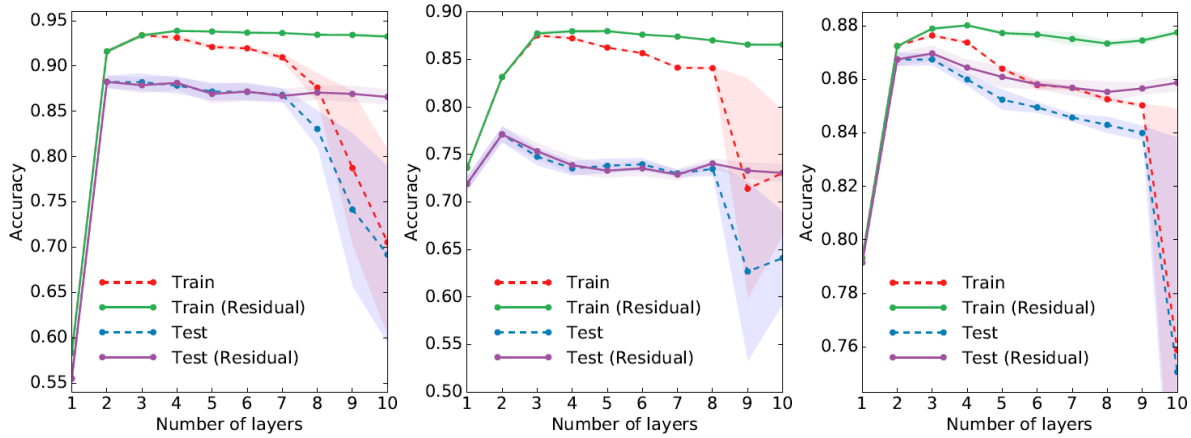
Kipf and Welling (2016a) also studied the impact of the GCN depth. They train networks with 1 to 10 layers, where the GCN layer could be standard (Equation 2.12) or a variant with residual connection (HE *et al.*, 2016), letting the model access information from the previous layer's inputs:

$$Y^{(l+1)} = \sigma_l\left(\hat{A}Y^{(l)}\Theta^{(l)}\right) + Y^{(l)}. \tag{2.14}$$

Figure 3 shows the results they obtained in this experiment. It can be seen that when the model has two layers, it achieves the best test results (or very close to best, in the Pubmed case), regardless of having or not having the variant residual connection. Indeed, the oversmoothing effect (LI; HAN; WU, 2018) is well-known to degrade GCN performance as it gets deeper, as discussed in Section 1.2. That's precisely why they used a two-layered network with no residual connection to obtain the results in Table 1 (see Kipf and Welling (2016a) for more details).

In every experiment of Kipf and Welling (2016a) – and other works (JIN *et al.*, 2020; ZHAO *et al.*, 2019; YAN; XIONG; LIN, 2018) –, trainable parameters are learned via Gradient Descent variants. These training algorithms are iterative, being prone to take several steps to converge to a meaningful solution. Thus, the proposal of a novel training technique that computes network weights analytically (instead of iteratively) with low computational cost turns out an interesting alternative.

Figure 3 – Case study of how the depth of GCN impacts its performance (KIPF; WELLING, 2016a). In general, as the number of layers grows, the test performance of the GCN is degraded regardless of whether residual connections are or are not employed.



(a) Cora (b) Citeseer (c) Pubmed

2.3 Extreme Learning Machine (ELM)

In contrast to Gradient Descent variant methods, Extreme Leaning Machine (ELM) (HUANG; ZHU; SIEW, 2006) is a non-iterative learning algorithm to train 2-layered fully connected networks (i.e., with exactly one hidden layer). Despite being fast, as weights are defined analytically, this training method has been overshadowed in the last years by the increasing interest in Deep Learning (DL), which considers networks with many layers.

Consider a fully connected network with one hidden layer and one output layer, defined by

$$Y = \sigma\left(X\Theta^{(1)}\right)\Theta^{(2)}, \tag{2.15}$$

where: $X \in \mathbb{R}^{N\times F}$ is the input matrix with N samples and F features; $\Theta^{(1)} \in \mathbb{R}^{F\times H}$ is the hidden layer weights; $\Theta^{(2)} \in \mathbb{R}^{H\times C}$ is the output layer weights; and σ is the hidden layer non-linear activation function. Note that the output layer has no bias and its activation is the identity function. Moreover, bias in the hidden layer can be emulated by adding a column of ones in the feature matrix X. Consider also $T \in \mathbb{R}^{N\times C}$ a matrix of desired targets.

The key idea of ELM (HUANG; ZHU; SIEW, 2006) is that even if $\Theta^{(1)}$ is randomly settled (and kept **unchanged**), $\Theta^{(2)}$ can still be analytically computed so as to minimize the error between the predicted Y and the targets T. Precisely, suppose $\Theta^{(1)}$ a $F \times H$ matrix with entries drawn from a probability distribution ψ and consider $Y_h = \sigma(X\Theta^{(1)})$ the (random) $N \times H$ non-linear mapping of X resulting from the hidden layer. Since the output layer is linear, i.e.,

$$Y = Y_h\Theta^{(2)}, \tag{2.16}$$

we can naively ask under which conditions Y_h (a random mapping) is an invertible matrix, such that $\Theta^{(2)}$ can be computed as $\Theta^{(2)} := Y_h^{-1}T$, letting Equation 2.16 become $Y = T$, i.e., the network output matches perfectly the target labels.

A necessary condition for the matrix to be invertible is that it is a square matrix (i.e. the number of rows equals the number of columns). Therefore, since $Y_h \in \mathbb{R}^{N\times H}$, the number of hidden neurons must be selected as the number of data samples ($H = N$). An interesting result proved by Huang, Zhu and Siew (2006) is that if this condition holds, then Y_h is an invertible matrix with probability one, thus $\Theta^{(2)} := Y_h^{-1}T$ gives a network with null training error, as follows by Theorem 2.1.

Theorem 2.1. (ELM – case H=N) (HUANG; ZHU; SIEW, 2006):

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be an infinitely differentiable activation function in any interval. Consider a single hidden layer fully connected network with N hidden neurons and an input matrix $X \in \mathbb{R}^{N \times F}$ with distinct rows, associated with a target matrix $T \in \mathbb{R}^{N \times C}$. If hidden weights $\Theta^{(1)}$ are randomly sampled according to any continuous probability distribution ψ, then the hidden mapping Y_h is, with probability one, invertible, and thus there exists $\Theta^{(2)}$ such that $\|Y_h\Theta^{(2)} - \mathbf{T}\| = 0$. Such $\Theta^{(2)}$ is defined as $\Theta^{(2)} := Y_h^{-1}T$.

Theorem 2.1 states that in certain conditions one can train the output layer of a fully connected network analytically given a random hidden layer weights matrix and obtain a network free of error. Also, note that the random weights should be sampled according to any continuous probability distribution ψ, which includes – but is not limited to – the commonly used uniform distribution (in an arbitrary interval) and Gaussian distribution (with arbitrary mean and standard deviation). Moreover, the error-free property is applied to the data that is presented to the network in the training step, that is, ELM finds weights such that the error is null at the **training** subset of the database. Algorithm 2.1 shows a step-to-step to train the network in this context.

Despite being fast and efficient, this particular case of ELM that considers $H = N$ has a number of drawbacks. First, the number of samples in a dataset is typically large, which means that the network should be configured with a hidden dimension much larger than commonly chosen. Second, a null error in the training subset is often undesired since it means the network is overfitting the data. Therefore, Theorem 2.1 actually shows an upper bound in the number of necessary hidden units and also guarantees maximum learning if that condition is satisfied.

In this context, Huang, Zhu and Siew (2006) also shows that one can upper bound the training error by properly tunning the number of units H in the hidden layer, as stated in Theorem 2.2. In this case, the inverse matrix Y_h^{-1} must be replaced by the Moore–Penrose pseudo-inverse $Y_h^{\dagger}$ (BANERJEE, 1973).

Algorithm 2.1 – Extreme Leaning Machine – case H = N

Input: X, T, σ, ψ
Output: $\Theta^{(1)}$, $\Theta^{(2)}$

$\rightarrow$ $\Theta^{(1)} \sim \psi$

$\rightarrow$ $Y_h = \sigma(X\Theta^{(1)})$

$\rightarrow$ $\Theta^{(2)} = Y_h^{-1}T$

Algorithm 2.2 – Extreme Leaning Machine
Input: X, T, σ, ψ
Output: $\Theta^{(1)}$, $\Theta^{(2)}$

$\rightarrow$ $\Theta^{(1)} \sim \psi$

$\rightarrow$ $Y_h = \sigma(X\Theta^{(1)})$

$\rightarrow$ $\Theta^{(2)} = Y_h^{\dagger} T$

Theorem 2.2. (ELM) (HUANG; ZHU; SIEW, 2006):

Let $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ be an infinitely differentiable activation function in any interval and $\varepsilon > 0$ arbitrary. Then, there exists $H \leqslant N$ such that given a single hidden layer fully connected network with H hidden neurons and an input matrix $X \in \mathbb{R}^{N \times F}$ with distinct rows, associated with a target matrix $T \in \mathbb{R}^{N \times C}$, if hidden weights $\Theta^{(1)}$ are randomly sampled according to any continuous probability distribution ψ, there exists, with probability one, $\Theta^{(2)}$ such that $\|Y_h\Theta^{(2)} - \mathbf{T}\| < \varepsilon$. Such $\Theta^{(2)}$ is defined as $\Theta^{(2)} := Y_h^{\dagger} T$.

We first note that Theorem 2.2 is a generalization of Theorem 2.1, since in the special case that $H = N$, we have Y_h a square matrix and hence its pseudo-inverse $Y_h^{\dagger}$ matches its inverse Y_h^{-1}, as discussed in Appendix A. Moreover, $\Theta^{(2)}$ as given by ELM Theorem has the property of being the weight matrix with minimum Frobenius norm such that the approximation error is also as small as possible (see Appendix A).

Algorithm 2.2 shows how to obtain the weights of the network. Note that Algorithm 2.2 is precisely Algorithm 2.1 changing only the inverse Y_h^{-1} with the pseudo-inverse $Y_h^{\dagger}$ operation in the last instruction. Besides, observe again that in the particular case that $H = N$, we have $Y_h^{\dagger} = Y_h^{-1}$, being Algorithm 2.1 a special case of Algorithm 2.2.

Algorithm 2.3 – Regularized Extreme Leaning Machine
Input: X, T, σ, ψ, γ
Output: $\Theta^{(1)}$, $\Theta^{(2)}$

$\rightarrow$ $\Theta^{(1)} \sim \psi$

$\rightarrow$ $Y_h = \sigma(X\Theta^{(1)})$

$\rightarrow$ $\Theta^{(2)} = \left(\frac{1}{\gamma} I + Y_h^T Y_h\right)^{\dagger} Y_h^T T$

A variant of ELM is the so-called Regularized Extreme Learning Machine (RELM) proposed by Deng, Zheng and Chen (2009). RELM relies on a formulation where the norm of the weight matrix $\Theta^{(2)}$ is penalized during optimization. With some algebraic manipulation, the regularized weight matrix $\Theta^{(2)}$ can be computed as the last instruction of Algorithm 2.3. Deng, Zheng and Chen (2009) also shows that ELM is a special case of RELM when the hyperparameter $\gamma \to \infty$.

CHAPTER

3

EXTREME LEARNING MACHINE TO GRAPH CONVOLUTIONAL NETWORKS (ELM-GCN)

In this chapter, we introduce our first contribution: ELM-GCN, a novel training algorithm for Graph Convolutional Networks (GCN). We first discuss the related works in Section 3.1 and then we formulate the theoretical results of our technique in Section 3.2. Finally, in Section 3.3 we evaluate experimentally our methodology applied to both synthetic and real-world datasets.

3.1 Related Works

A main issue in the context of Graph Neural Networks is the time spent to train the models, especially in problems involving large graphs. A number of different strategies have been proposed to tackle this issue. Hamilton, Ying and Leskovec (2017), for instance, proposes GraphSAGE, a training mechanism that relies on an aggregation function to sample the neighborhood of nodes. Cluster-GCN (CHIANG *et al.*, 2019) is a learning algorithm that applies graph clusters to restrict the neighborhood search to a subgraph identified by a graph cluster algorithm. GraphACT (ZENG; PRASANNA, 2020) builds upon CPU-FPGA heterogeneous systems to boost the training process. L-GCN (YOU *et al.*, 2020) is a layer-wise training algorithm for GCN that learns the weight matrix in each layer in a sequential manner. The Simple Graph Convolution (WU *et al.*, 2019) method reduces the complexity of GCNs by removing nonlinearities, collapsing the result into a single linear transformation. A technique that has gained great attention in recent years is the fastGCN proposed by Chen, Ma and Xiao (2018), which interprets graph convolution as integral transforms of embedding functions under probability measures, evaluating the integrals through the Monte Carlo method.

The methods discussed above rely on iterative gradient descent-based algorithms to update the weights in each layer every epoch during the training. In contrast, the methods proposed

in this Chapter compute the weights of GCNs analytically, exhibiting comparable performance to other GCN training mechanisms while being more efficient in terms of computational times.

Graph Convolutional Extreme Learning Machine (GCELM) (ZHANG *et al.*, 2020) is a training methodology that closely relates to the proposed RELM-GCN. However, our approach, RELM-GCN, differs from GCELM in four main aspects: first, RELM-GCN has a message-passing mechanism in the second layer, which GCELM has not. This leads to the important consequence that each vertex in our model gathers information from both 1- and 2-hop neighbors, while GCELM only considers information from adjacent nodes. Second, Zhang *et al.* (2020) conducts experiments only on non-graph machine learning datasets, i.e., they derive a graph from correlations present on the data used in their experiments. Therefore, the graph structure and the feature vectors associated with the nodes are closely correlated, making it hard to assess whether the good reported results are a consequence of their training mechanism or due to the bias they introduced when generating the graph. Third, we formulate and prove theorems to establish theoretical guarantees to our approach and such theoretical results are not present in GCELM. Lastly, we state a theoretical upper bound in the number of hidden units to guarantee that a GCN trained with the proposed ELM-GCN algorithm results in controlled training error.

3.2 Theoretical Approach

On the one hand, ELM is an efficient learning algorithm to train a fully connected network if its architecture has exactly 2 layers, as shown in Section 2.3. On the other hand, GCN is a model that works on graphs that performs well with exactly 2 layers, as discussed in Section 2.2. In the following, we show how to extend ELM to operate (train) on 2-layered GCNs with a linear output activation function.

First, comparing Equations 2.13 and 2.15, one notes that the difference between the considered architectures is only the graph convolutional filter $\hat{A}$ in each layer and the output activation σ_2.Hence, we consider a 2-layered GCN with linear output activation $\sigma_2(x) = x$, resulting in the mapping

$$Y = \hat{A}\,\sigma\left(\hat{A}X\Theta^{(1)}\right)\,\Theta^{(2)}, \tag{3.1}$$

where: $X \in \mathbb{R}^{N\times F}$ is a F-dimensional graph signal associated with the N nodes of a graph G; $\Theta^{(1)} \in \mathbb{R}^{F\times H}$ is the hidden layer weight matrix; $\Theta^{(2)} \in \mathbb{R}^{H\times C}$ is the output layer weight matrix; and $\hat{A} \in \mathbb{R}^{N\times N}$ is the GCN graph convolution filter, which depends only on the graph structure, as shown in Section 2.2. For convenience, we replicate the definitions of matrices $\hat{A}$, $\tilde{D}$ and $\tilde{A}$ as follows:

$$\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}, \tag{3.2}$$

$$\tilde{D} = \text{diag}(\sum_{j}\tilde{A}_{i,j}), \tag{3.3}$$

$$\tilde{A} = A + I. \tag{3.4}$$

Similar to classical ELM, we assume that $\Theta^{(1)}$ is a matrix with entries randomly sampled from any continuous probability distribution ψ and that $\Theta^{(1)}$ is fixed during the whole training process. We then compute the random mapping of the input graph signal $Y_h = \sigma(\hat{A}X\Theta^{(1)})$. Note that the output of the GCN can now be written as $Y = \hat{A}\, Y_h\, \Theta^{(2)}$. First, consider we configured the number of hidden units H to match the number of nodes N of the graph. Under this assumption, we can prove Theorem 3.1 and guarantee that Y_h is invertible with probability one. Moreover, the output weight matrix $\Theta^{(2)}$ can be analytically computed to ensure no training error. Algorithm 3.1 shows the computational steps.

Theorem 3.1. (ELM-GCN – case H=N):

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be an infinitely differentiable activation function in any interval, G a graph with N nodes, non-negative edge weights and invertible matrix $\tilde{A}$. Let $\hat{X} = \hat{A}X \in \mathbb{R}^{N\times C}$ be a convolved graph signal matrix with distinct rows and $T \in \mathbb{R}^{N\times F}$ be the target labels. Then, if a single hidden layer GCN with N hidden units has its hidden weights $\Theta^{(1)}$ randomly sampled according to any continuous probability distribution ψ, then the convolved hidden mapping $(\hat{A}Y_h)$ is, with probability one, invertible and thus there exists $\Theta^{(2)}$ such that $\|\hat{A}Y_h\Theta^{(2)} - T\| = 0$. Such $\Theta^{(2)}$ is defined as $\Theta^{(2)} := (\hat{A}Y_h)^{-1}T$.

Proof. First, we can see the convolved graph signal matrix $\hat{X}$ as the input matrix X of Theorem 2.1. Since their other hypothesis that rows of $\hat{X}$ are distinct, σ is an infinitely differentiable activation function and $\Theta^{(1)}$ is randomly sampled from a continuous probability distribution are also satisfied, we can conclude from Theorem 2.1 that $\sigma(\hat{X}\Theta^{(1)}) = \sigma(\hat{A}X\Theta^{(1)}) =: Y_h$ is invertible with probability one.

Now note that since edge weights of G are non-negative, all elements of matrix $\tilde{A}$ (Equation 3.4) are non-negative. Thus, all diagonal elements of $\tilde{D}$ (Equation 3.3) are positive and hence $\tilde{D}$ and $\tilde{D}^{-1/2}$ must be invertible matrices. Then, since by hypothesis $\tilde{A}$ is invertible, Equation 3.2 shows that $\hat{A}$ must also be an invertible matrix. Hence, with probability one, $\hat{A}Y_h$ is invertible and defining $\Theta^{(2)} := (\hat{A}Y_h)^{-1}T$ we have $\|\hat{A}Y_h\Theta^{(2)} - \mathbf{T}\| = 0$, concluding the proof of the Theorem. □

Algorithm 3.1 – Extreme Leaning Machine to Graph Convolutional Network – case H = N

Input: $\hat{A}$, X, T, σ, ψ
Output: $\Theta^{(1)}$, $\Theta^{(2)}$

$\rightarrow$ $\Theta^{(1)} \sim \psi$

$\rightarrow$ $Y_h = \sigma(\hat{A}X\Theta^{(1)})$

$\rightarrow$ $\Theta^{(2)} = (\hat{A}Y_h)^{-1}T$

Analogously to what was discussed about Theorem 2.1, Theorem 3.1 ensure null training error if the hidden dimension is set to match the number of graph nodes, one can analytically train the output layer weights of a GCN given a random hidden layer weights matrix whose entries can be drawn from any continuous probability distribution ψ. However, it is desirable to be able to configure the number of hidden units smaller than the number of graph nodes N. This trade-off implies that the training error is not guaranteed to be null, which also alleviates overfitting. Moreover, Theorem 3.1 actually provides an upper bound in the number of hidden units to guarantee zero training error. To the best of our knowledge, our published paper (GONÇALVES; NONATO, 2022) is the first work to provide the maximum number of hidden units in a GCN to ensure zero training error while stating an algorithm to find this model.

In what follows, we present Theorem 3.2 and Algorithm 3.2, which generalize both Theorem 3.1 and Algorithm 3.1 allowing our learning scheme to be applied to any number of hidden units.

Theorem 3.2. (ELM-GCN):

Let $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ be an infinitely differentiable activation function in any interval, $\varepsilon > 0$ an arbitrary scalar, G a graph with N nodes, non-negative edge weights and invertible matrix $\tilde{A}$. Let $\hat{X} = \hat{A}X \in \mathbb{R}^{N \times C}$ be a convolved graph signal matrix with distinct rows and $T \in \mathbb{R}^{N \times F}$ be the target labels. Then, there exists $H \leqslant N$ such that if a single hidden layer GCN with H hidden units has its hidden weights $\Theta^{(1)}$ randomly sampled according to any continuous probability distribution ψ, then, with probability one, there is $\Theta^{(2)}$ such that $\|\hat{A}Y_h\Theta^{(2)} - T\| < \varepsilon$. Such $\Theta^{(2)}$ is defined as $\Theta^{(2)} := (\hat{A}Y_h)^{\dagger}T$.

Proof. Following exactly the same proof as of Theorem 2.2 (HUANG; ZHU; SIEW, 2006), the validity of the theorem is obvious, otherwise, one could simply choose $H = N$ which makes $\|\hat{A}Y_h\Theta^{(2)} - T\| < \varepsilon$ according to Theorem 3.1. □

Algorithm 3.2 – Extreme Learning Machine to Graph Convolutional Network

Input: $\hat{A}$, X, T, σ, ψ
Output: $\Theta^{(1)}$, $\Theta^{(2)}$

$\rightarrow$ $\Theta^{(1)} \sim \psi$

$\rightarrow$ $Y_h = \sigma(\hat{A}X\Theta^{(1)})$

$\rightarrow$ $\Theta^{(2)} = (\hat{A}Y_h)^{\dagger}T$

Algorithm 3.2 is similar to Algorithm 3.1, but replacing the inverse $(\hat{A}Y_h)^{-1}$ by the pseudo-inverse $(\hat{A}Y_h)^{\dagger}$ operator (Appendix A). Therefore, in the special case $H = N$, we have $(\hat{A}Y_h)^{\dagger} = (\hat{A}Y_h)^{-1}$ (see Appendix A), making Algorithm 3.2 to be reduced to Algorithm 3.1.

We have also adapted the RELM algorithm to the context of GCN, resulting in a new expression to compute $\Theta^{(2)}$, as shown in Algorithm 3.3. The regularized version of our algorithm is called Regularized Extreme Learning Machine to Graph Convolutional Network (RELM-GCN) and we now also prove that ELM-GCN is a special case of RELM-GCN when the hyperparameter $\gamma \to \infty$.

Theorem 3.3. If the hyperparameter γ in the last instruction of Algorithm 3.3 is such that $\gamma \to \infty$, then the RELM-GCN technique is the same as the ELM-GCN. In other words, ELM-GCN is a special case of RELM-GCN.

Proof. When $\gamma \to \infty$ we have $\frac{1}{\gamma}I \to 0$. Thus the analytical assignment to $\Theta^{(2)}$ by RELM-GCN (last instruction of Algorithm 3.3) becomes:

$$\begin{aligned}
\Theta^{(2)} &= \left(\frac{1}{\gamma}I + (\hat{A}Y_h)^T(\hat{A}Y_h)\right)^{\dagger}(\hat{A}Y_h)^T T = \left((\hat{A}Y_h)^T(\hat{A}Y_h)\right)^{\dagger}(\hat{A}Y_h)^T T \\
&= (\hat{A}Y_h)^{\dagger}\left((\hat{A}Y_h)^T\right)^{\dagger}(\hat{A}Y_h)^T T = (\hat{A}Y_h)^{\dagger}\left((\hat{A}Y_h)^{\dagger}\right)^T(\hat{A}Y_h)^T T \\
&= (\hat{A}Y_h)^{\dagger}\left((\hat{A}Y_h)(\hat{A}Y_h)^{\dagger}\right)^T T = (\hat{A}Y_h)^{\dagger}(\hat{A}Y_h)(\hat{A}Y_h)^{\dagger}T = (\hat{A}Y_h)^{\dagger}T
\end{aligned}$$

which is the assignment to $\Theta^{(2)}$ given by ELM-GCN (last instruction of Algorithm 3.2). □

Algorithm 3.3 – Regularized Extreme Learning Machine to Graph Convolutional Network

Input: $\hat{A}$, X, T, σ, ψ, γ
Output: $\Theta^{(1)}$, $\Theta^{(2)}$

$\rightarrow\ \Theta^{(1)} \sim \psi$

$\rightarrow\ Y_h = \sigma(\hat{A}X\Theta^{(1)})$

$\rightarrow\ \Theta^{(2)} = \left(\frac{1}{\gamma} I + (\hat{A}Y_h)^T(\hat{A}Y_h)\right)^{\dagger}(\hat{A}Y_h)^T T$

3.3 Experiments

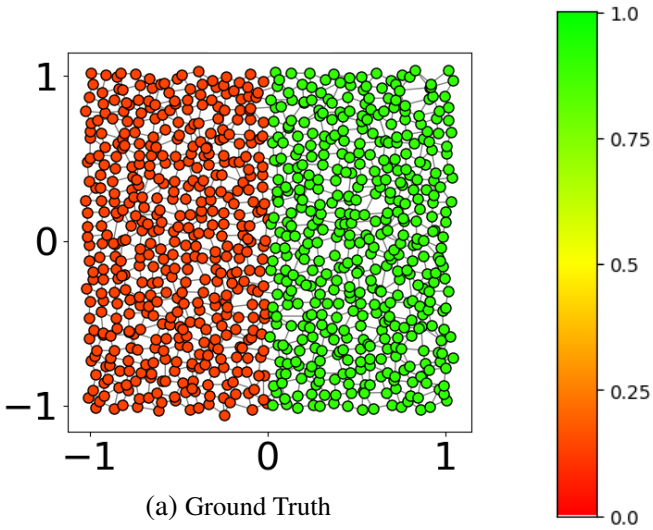
Syntetic Data 1. This first experiment relies on a simple but illustrative synthetic dataset that allows us to assess whether the ELM-GCN and RELM-GCN algorithms behave as expected when training a GCN.

Consider a graph G derived from a 30×30 regular grid, that is, nodes are placed in the center of each grid cell and each node is connected by an edge to its left, right, top, and bottom neighbors. Gaussian noise is added to the x and y coordinates of the nodes and the perturbed coordinates are assigned as a two-dimensional feature vector to the nodes in G (see Figure 4a). A binary label (0 or 1) is associated with each node depending on whether the node comes from the left or right half of the grid, that is, the left half of the nodes are labeled as 0 and the right half as 1, as depicted in Figure 4a. Although this graph dataset gives rise to a simple classification task, it is able to reveal whether the proposed algorithms behave as expected when training 2-layer GCNs with different hidden layer configurations.

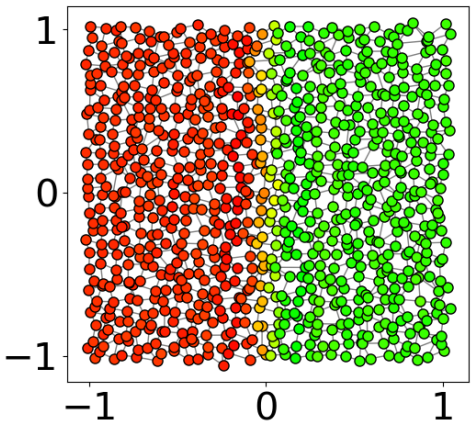
We divide the graph nodes randomly into training (80%), validation (10%), and test (10%) subsets. A GCN is then trained using the transductive paradigm with both ELM-GCN and RELM-GCN strategies keeping the subsets fixed, but varying the number of hidden units H along every integer from 1 to 900, the last being the number of nodes N in the graph. For RELM-GCN, we set the regularization parameter γ as $\{10^{-8},\ 10^{-7},\ \ldots\ ,\ 10^{8}\}$ for each H value. Figure 5 and Figure 6 show the Mean Square Error (MSE) of ELM-GCN and RELM-GCN in terms of the number of hidden units, respectively (in Figure 6, the MSE corresponds to the smallest error for all γ in each H value).

ELM-GCN was trained with 900 hyperparameter configurations ($H \in \{1, 2, \ldots 900\}$). Similarly, RELM-GCN had more than 15,000 distinct setups ($\gamma \in \{10^{-8},\ 10^{-7},\ \ldots\ ,\ 10^{8}\}$ for each $H \in \{1, 2, \ldots 900\}$). Therefore, it would be intractable to employ this amount of trainings in a reasonable time with a standard iterative learning algorithm, clearly showing the advantage of the proposed analytical training solution.

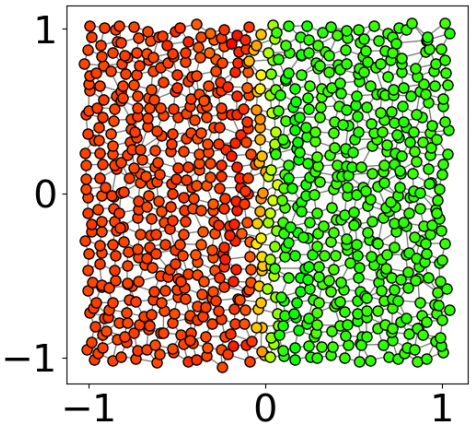
Figure 4 – Result obtained in the experiment with the first synthetic data. The GCN trained with both ELM-GCN and RELM-GCN can reproduce the ground truth after thresholding.



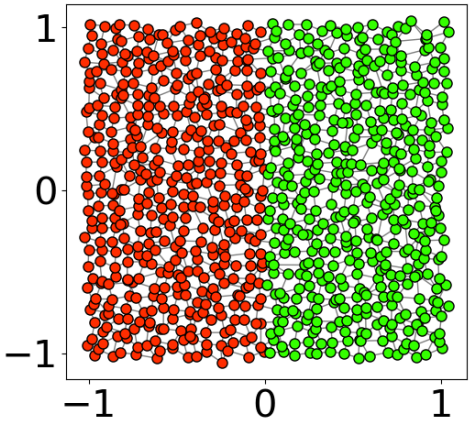
(a) Ground Truth



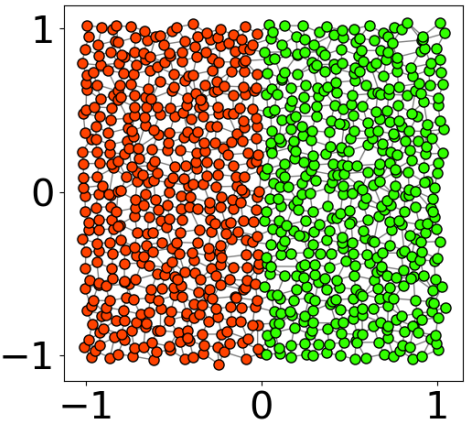
(b) Result obtained with ELM-GCN



(c) Result obtained with RELM-GCN



(d) Result obtained with ELM-GCN (thresholded)



(e) Result obtained with RELM-GCN (thresholded)

Figure 5 – Loss of GCN trained with ELM-GCN for different numbers of hidden units. This is a classic behavior of learning algorithms: as the number of parameters grows, the training error decreases, while the validation drops and then increases (indicating overfitting).

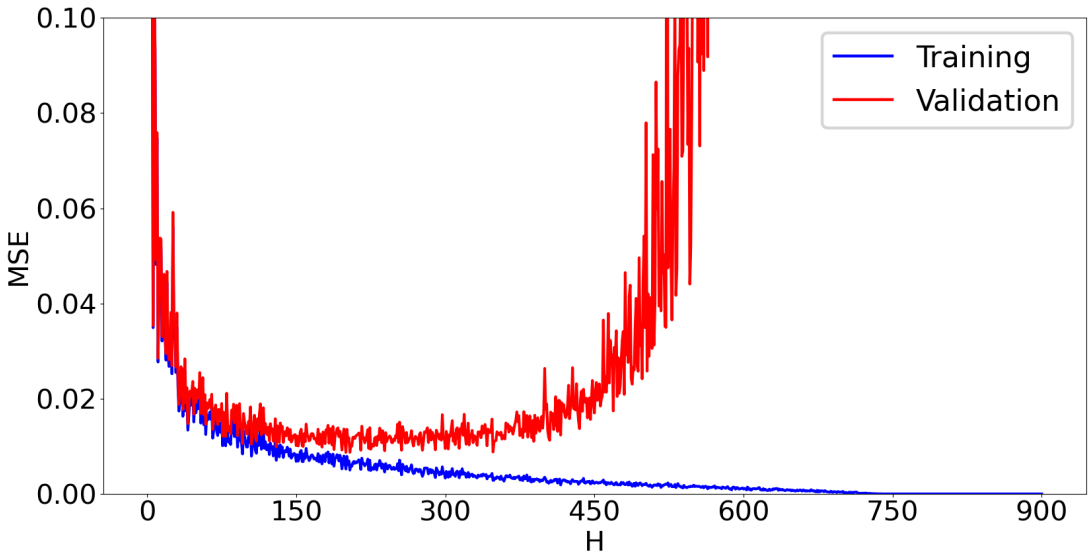
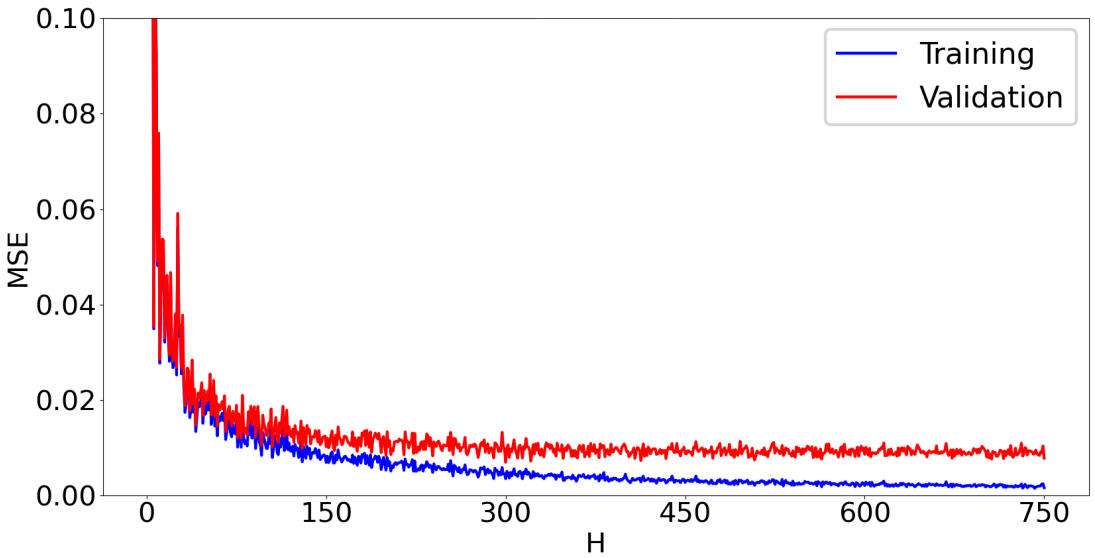


Figure 6 – Loss of GCN trained with RELM-GCN for different numbers of hidden units. In contrast with Figure 5, the regularization mechanism is able to alleviate overfitting as the number of parameters grows.

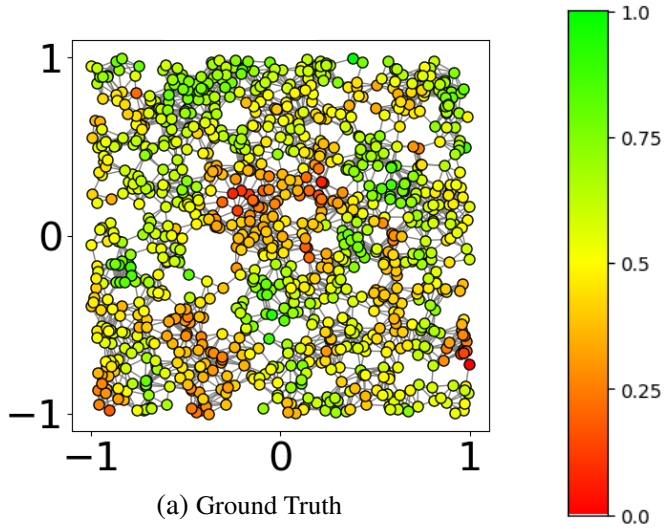


Moreover, notice in Figure 5 that the training error tends to decrease when the number of hidden neurons H increases. Validation error, in contrast, drops quickly until a certain point, when it starts to increase, indicating overfitting. Figure 6 shows that both validation and training errors follow the same decreasing tendency for RELM-GCN, indicating that the regularization mechanism minimizes overfitting. Those classical behaviors show that both ELM-GCN and RELM-GCN perform as expected and are similar to other non-regularized and regularized learning techniques.

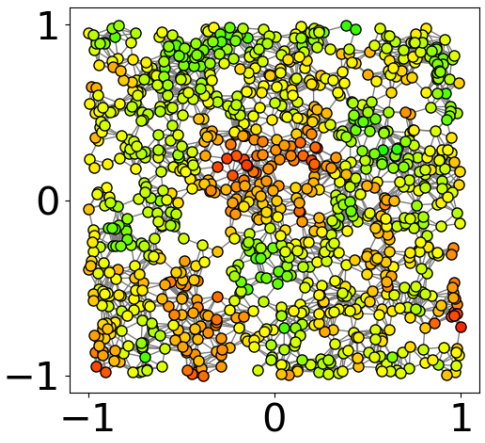
Finally, one can see in Figure 5 that the ELM-GCN training error vanishes when the number of hidden units reaches 720 (which is the number of graph training nodes), corroborating with Theorem 3.1.

By selecting the best hyperparameter combination as to minimize MSE during validation (the chosen hyperparameters were $H^* = 200$ for ELM-GCN and $(H^*,\ \gamma^*) = (300, 10^5)$ for RELM-GCN), we obtain Figure 4b and Figure 4c. Moreover, the GCNs trained with both proposed methods can predict the labels as expected, presenting errors only near the decision boundary between the two classes, which is also a classic model behavior. Thresholding the output of the network in $Y_{thresh} = 0.5$ results in Figure 4d and Figure 4e, implying 100% classification accuracy for this fairly simple synthetic experiment.

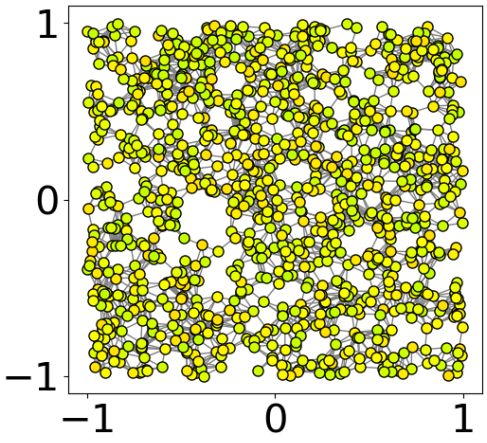
Figure 7 – Result obtained in the second synthetic dataset experiment with RELM-GCN and RELM. RELM-GCN can explore the neighborhood aggregation mechanism present in the GCN to make the predictions.



(a) Ground Truth



(b) Result of RELM-GCN



(c) Result of classic RELM

Synthetic Data 2. In the next experiment, we compare RELM-GCN against classic RELM aiming to check if our method is able to capture graph topological information. We build the synthetic dataset by first uniformly sampling $N = 1000$ random points of the square $[-1,1]^2$ as the graph nodes. Then we defined a complete graph G where each edge is associated to a weight $\omega_{i,j} = \exp(-d(v_i,\ v_j)^2/\sigma_\omega)$, where $d(v_i,\ v_j)$ is the euclidian distance between nodes v_i and v_j and σ_ω is a parameter defined as $\sigma_\omega = 0.1$. Edge weights are bigger for closer nodes than for ones more distant in the square. Then, edges with weight less than a threshold $\tau_\omega = 0.85$ are removed and we associate to each node a random feature $x_i \in [0,1]$ extracted from the uniform distribution. Finally, we define the target of each node to be the average between its feature and the feature of its neighbors. Figure 7 depicts the graph and the random features represented by this synthetic dataset. One can observe that the network must handle neighborhood information to predict the targets correctly in this experiment.

We randomly partition the graph nodes into 80% / 10% / 10% respectively for the training / validation / test subsets. Subsequently, we employ both RELM-GCN and RELM learning algorithms to train the respective models they were designed for, i.e., a Graph Convolutional Network (using transductive paradigm) and a Fully Connected Network, respectively, keeping the subsets constant for both strategies. Moreover, we vary the number of hidden units H from 1 to 50, while the regularization parameter γ ranges from 10^{-8} to 10^{8} in powers of 10 for each number of hidden units. Again, we could only such a large amount of different hyperparameters configurations thanks to the fast training process of both ELM-derived mechanisms. Indeed, it takes less than a second to train the model (in the experiments with real data below we show the precise training times for bigger datasets).

Figure 8 – MSE of the models trained with RELM-GCN and RELM in the graph of Figure 7a. RELM-GCN presents a loss lower than RELM for exploring the capacity of GCN to aggregate data from node neighbors.

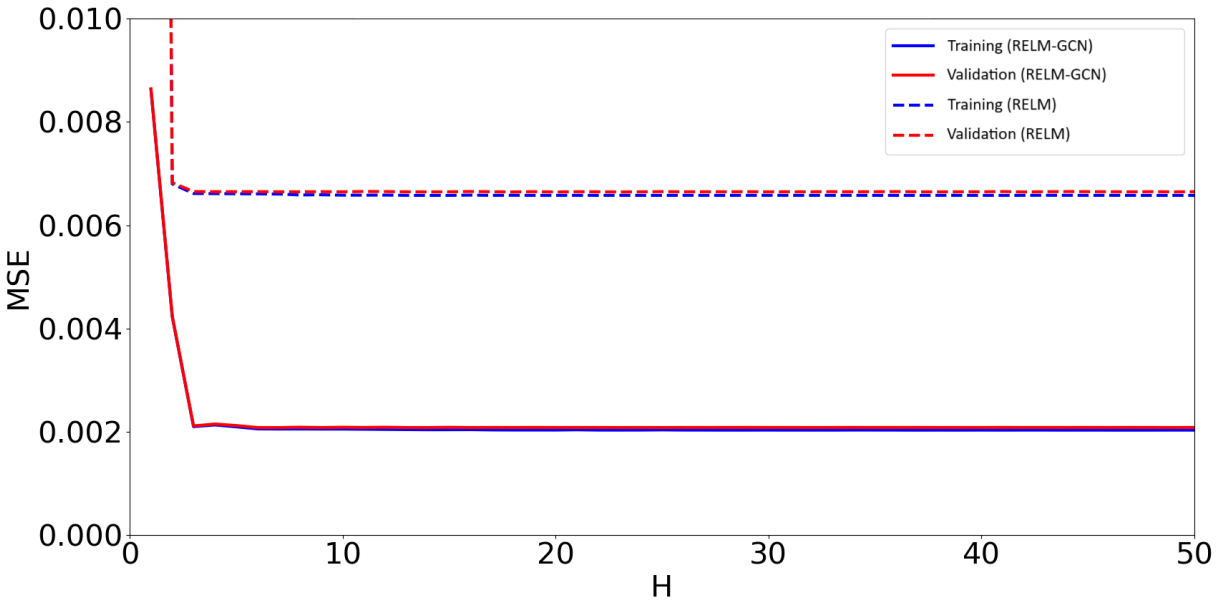


Figure 8, interpreted like Figure 6, shows the result of this experiment to both RELM-GCN and RELM training schemes. Clearly our proposed learning algorithm RELM-GCN is able to exploit GCN's capacity to deal with neighborhood data to enhance the model's predictions. Moreover, the best hyperparameters configurations are $(H^*,\ \gamma^*) = (7, 10^6)$ for RELM-GCN and $(H^*,\ \gamma^*) = (20, 10^{-1})$ for RELM and we note a quick stabilization of both networks with very few hidden units, due to the low complexity of the task. However, one can clearly see in Figures 7b and 7c that RELM-GCN trained the model that correctly captures graph features in contrast with RELM, which provides almost the same result for all nodes. Those figures depict the outputs of the networks with their best hyperparameters configuration.

Real Data. Finally, in our last experiment, we assess the performance of ELM-GCN and RELM-GCN when training GCNs on four well-known graph datasets: Cora, Citeseer, Pubmed, and Reddit. The first three datasets are citation networks and the goal is to categorize documents, whereas in the fourth dataset, the model should predict post categories from the Reddit community, a social network. Cora, Citeseer, and Pubmed are available in <https://github.com/tkipf/gcn> and the Reddit dataset can be obtained from <http://snap.stanford.edu/graphsage/>. Similarly to Chen, Ma and Xiao (2018), for the citation networks we build the validation and test subsets as in Kipf and Welling (2016a), employing the remaining nodes for training. In the case of Reddit, we split the dataset as in Chen, Ma and Xiao (2018). Table 2 shows the dataset statistics and how the splits have been done.

We compare GCNs (KIPF; WELLING, 2016a) trained with the proposed ELM-GCN / RELM-GCN against the classical gradient descent-based method, which we call Backpropagation (BP), and the fastGCN learning algorithm proposed by Chen, Ma and Xiao (2018). Baseline implementations were downloaded from <https://github.com/tkipf/gcn> and <https://github.com/matenure/FastGCN>, respectively.

The hyperparameters for BP and fastGCN are settled as proposed in the baseline implementation. Specifically, in the citation networks, BP (KIPF; WELLING, 2016a) is configured with 16 hidden units, $5 \cdot 10^{-4}$ L_2 as regularization parameter, 0.01 as learning rate, dropout equal to 0.5, 200 epochs and 10 patience iteration (i.e., optimization is early stopped if accuracy

Table 2 – Statistics and split of each dataset for the real data experiment.

Dataset	Nodes	Edges	Features	Classes	Split (Train./Val./Test)
Cora	2,708	5,429	1,433	7	1,208 / 500 / 1,000
Citeseer	3,327	4,732	3,703	6	1,827 / 500 / 1,000
Pubmed	19,717	44,338	500	3	18,217 / 500 / 1,000
Reddit	232,965	11,606,919	602	41	153,932 / 23,699 / 55,334

in the validation does not increase over 10 consecutive iterations). GCN is not trained with BP for the Reddit dataset, as it runs out of memory for large graphs. For fastGCN (CHEN; MA; XIAO, 2018), the number of hidden units is 16 for Cora and Pubmed and 128 for Reddit, L_2 regularization parameter is $5 \cdot 10^{-4}$ for Cora and Pubmed and $1 \cdot 10^{-4}$ for Reddit, learning rate ranges in $\{0.01, 0.001, 0.0001\}$, no dropout is used and 200 epochs is employed for all datasets, setting 10 patience iteration for Cora and Pubmed and 30 for Reddit. The sample size is 400 for Cora and Reddit and 100 for Pubmed, batch size is 256 for Cora and Reddit and 1024 for Pubmed. We use importance sampling for fastGCN, since Chen, Ma and Xiao (2018) report better accuracies with this configuration. Since Chen, Ma and Xiao (2018) do not conduct experiments on the Citeseer dataset, we configured all hyperparameters for this dataset to be the same as in Cora.

It is well known that classical ELM / RELM requires more hidden units when compared to Backpropagation (HUANG; ZHU; SIEW, 2006). Therefore, for our techniques, we have run experiments with the number of hidden units $H \in \{128, 256, 512, 750, 1000, 1250, 1500\}$. The regularization parameter γ ranges in $\{10^{-8}, 10^{-7}, \ldots, 10^{8}\}$ and no dropout was used. We select hyperparameters that lead to the best accuracy during validation.

Table 3 – Accuracy (rate of corrected predictions) and training time (in seconds) with transductive paradigm. The numbers of this table are pictured in Figure 9a and Figure 9c.

Dataset	Accuracy (%)				Time (seconds)			
	BP	fastGCN	ELM-GCN	RELM-GCN	BP	fastGCN	ELM-GCN	RELM-GCN
Cora	86.33	83.86	82.83	**86.87**	15.33	16.77	**0.15**	0.42
Citeseer	**77.72**	75.29	74.05	77.34	17.43	49.42	**0.51**	0.66
Pubmed	86.85	86.28	87.13	**87.74**	152.64	4.85	4.48	**4.20**
Reddit	NA	**93.83**	91.94	91.96	NA	911.38	136.64	**65.41**

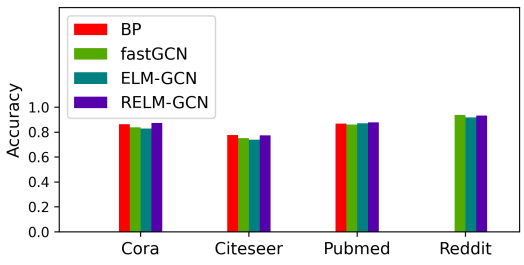
Table 4 – Accuracy (rate of corrected predictions) and training time (in seconds) with inductive paradigm. The numbers of this table are pictured in Figure 9b and Figure 9d.

Dataset	Accuracy (%)				Time (seconds)			
	BP	fastGCN	ELM-GCN	RELM-GCN	BP	fastGCN	ELM-GCN	RELM-GCN
Cora	83.17	84.62	78.58	**87.18**	7.20	11.23	**0.10**	0.46
Citeseer	77.82	**78.49**	70.10	77.49	13.28	42.81	**0.17**	0.91
Pubmed	86.64	86.65	87.30	**87.33**	147.78	6.22	11.65	**5.15**
Reddit	NA	**93.36**	91.02	92.50	NA	831.01	90.95	**47.61**

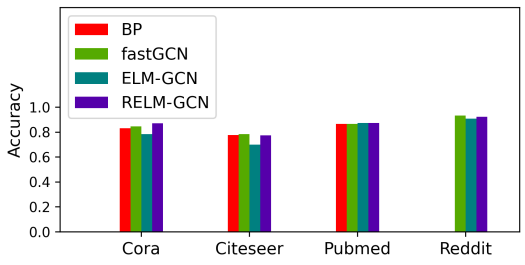
For each learning configuration, we train the networks based on both transductive and inductive paradigms, as proposed by Yang, Cohen and Salakhudinov (2016). In the transductive paradigm, the whole graph is assumed to be known for the training step and the task is to predict labels from the features associated with the nodes of the graph, training labels, and the graph topology. In the inductive paradigm, the model learns from a subgraph made up of training nodes only. After training, the full graph is presented to the network and the task becomes to predict labels from the whole structure. For each dataset, training paradigm, and learning algorithm, 10 executions are performed. The results are shown in Tables 3 and 4 are the average of these executions. We ran experiments in a single machine with 4-core 1.99GHz Intel Core i7 and 16G RAM.

Figure 9 depicts graphically the content of Tables 3 and 4. From Figure 9 it is possible to observe that GCN trained with the classical BP, fastGCN, ELM-GCN, and RELM-GCN present similar performance in terms of accuracy for all datasets in both paradigms. However, regarding training time, which is represented in the logarithm scale, both ELM-GCN and its regularized version turn out to be at least one order of magnitude faster than BP and fastGCN in three out of the four datasets (Cora, Citesser, and Reddit). In the Pubmed dataset, RELM-GCN is also one order of magnitude faster than BP, while being 15% faster than fastGCN. In fact, the fastest learning methods were ELM-GCN or RELM-GCN for every dataset in both training paradigms. Therefore, we can see that our proposed schemes are competitive training alternatives for 2-layer GCNs in terms of accuracy while presenting a considerable gain in training time.

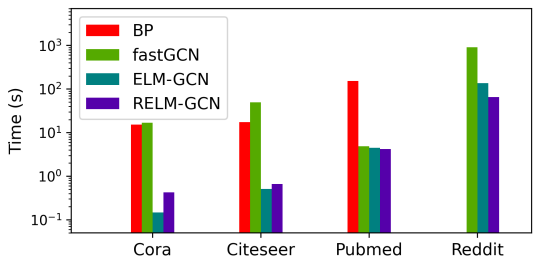
Figure 9 – Accuracy (rate of corrected predictions) and training times (in seconds) for both transductive and inductive paradigms. ELM-GCN and RELM-GCN are learning alternatives that maintain accuracy but train much faster than the baseline algorithms.



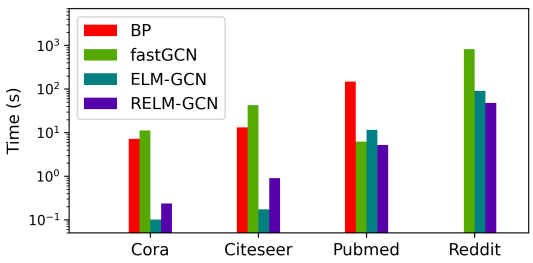
(a) Accuracy (transductive)



(b) Accuracy (inductive)



(c) Time (transductive)



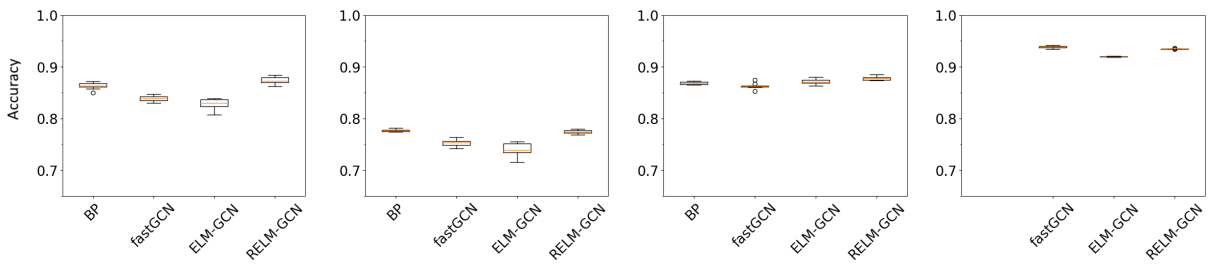
(d) Time (inductive)

In the following, we further validate the results obtained in the experiments involving real data. Specifically, Figures 10 and 11 show respectively the accuracy and training time boxplots obtained among different executions that produced Figure 9. Moreover, we employ the Wilcoxon Signed-Rank Test (DEMŠAR, 2006) which is a non-parametric hypothesis test that compares two samples that are paired. Precisely, the test compares accuracy and training time produced by either ELM-GCN or RELM-GCN against the ones resulting from the other two algorithms. First, we consider the null hypothesis that one of our approaches and competing algorithms generate results according to the same distribution. If the null hypothesis is rejected, we proceed to the next step which considers another null hypothesis that ELM-GCN or its regularized version performs worse (i.e. lower accuracy or higher training time) than the other learning techniques. In all tests, we use a significance of 99.9%.

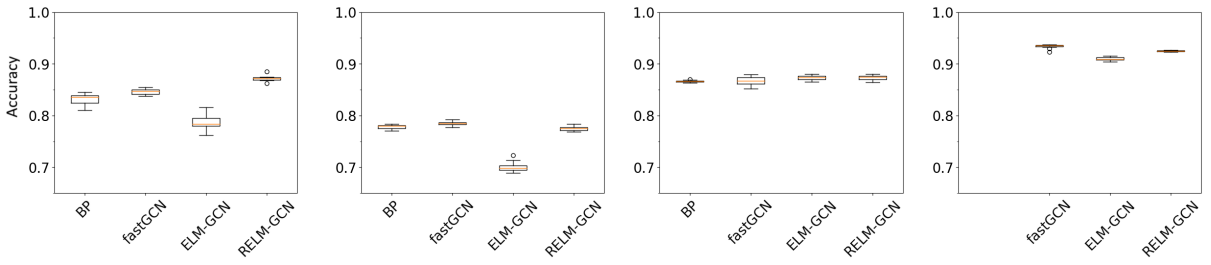
Regarding ELM-GCN, the Wilcoxon test rejected the hypothesis that this technique produces output at least as accurate as BP or fastGCN. In terms of training time, the null hypothesis that ELM-GCN comes from the same distribution as its competitors is rejected. Moreover, the Wilcoxon test also rejects the hypothesis that ELM-GCN has a higher training time than BP or fastGCN.

Comparing RELM-GCN with BP, the Wilcoxon test could not reject the null hypothesis that both techniques produce the same accuracy. However, when RELM-GCN is compared against fastGCN, we get an interesting outcome. The null hypothesis can not be rejected when both learning algorithms are compared in the inductive paradigm, but the hypothesis is rejected

Figure 10 – Accuracy (rate of corrected predictions) boxplot obtained for different training iterations. The Wilcoxon test concluded that RELM-GCN produces output at least as accurate as the baselines.

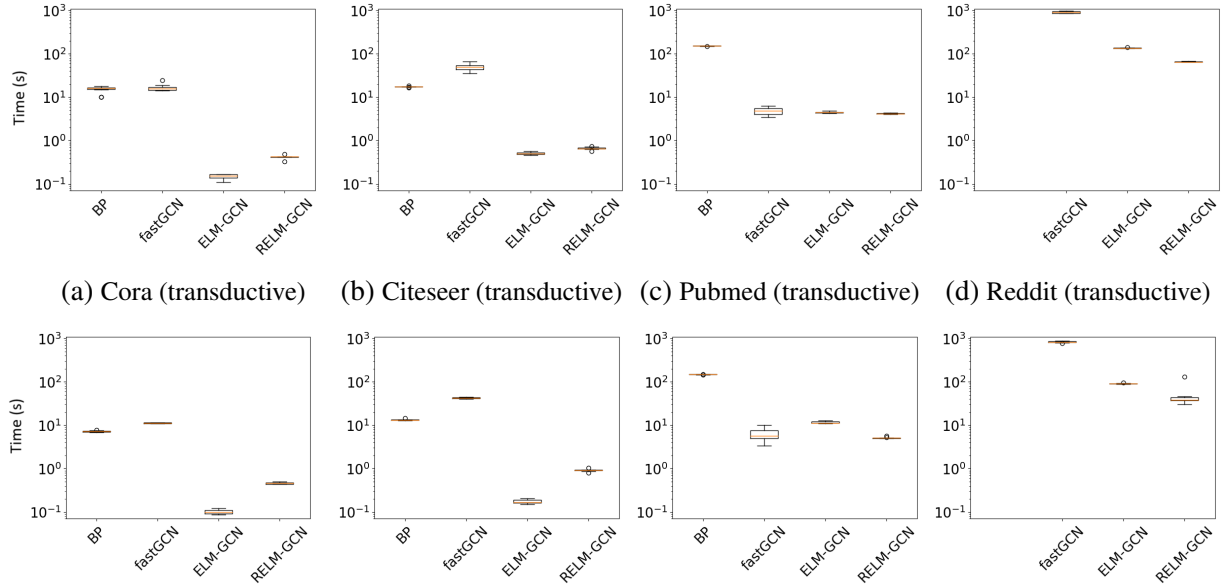


(a) Cora (transductive) (b) Citeseer (transductive) (c) Pubmed (transductive) (d) Reddit (transductive)



(e) Cora (inductive) (f) Citeseer (inductive) (g) Pubmed (inductive) (h) Reddit (inductive)

Figure 11 – Time (in seconds) boxplot obtained for different trainings. The Wilcoxon test rejected the hypothesis that RELM-GCN is slower than the baselines.



(a) Cora (transductive) (b) Citeseer (transductive) (c) Pubmed (transductive) (d) Reddit (transductive)

(e) Cora (inductive) (f) Citeseer (inductive) (g) Pubmed (inductive) (h) Reddit (inductive)

in the transductive scenario. Moreover, the hypothesis that RELM-GCN is less accurate than fastGCN in the same paradigm is rejected. Indeed, a careful analysis of Figure 9a shows that RELM-GCN consistently outperforms fastGCN on the first 3 datasets while performing comparably on Reddit.

Furthermore, the Wilcoxon test rejected the null hypothesis that RELM-GCN is as fast as competing techniques, regardless of the algorithm and paradigm chosen to compare. Moreover, the second step of the test showed that we should also reject the hypothesis that RELM-GCN is slower than the other algorithms in any learning paradigm. Conclusions provided by the Wilcoxon test are consistent with training times shown in Figure 9, since RELM-GCN outperforms the competing algorithms in most datasets, being only comparable with fastGCN in Pubmed.

CHAPTER

4

GRAPH NEURAL NETWORK FOR VALUING SOCCER PLAYERS (GNN-VSP)

One of the challenges in the Graph Machine Learning field is dealing with the task of predicting time series associated with the elements of a dynamic graph. In this chapter we consider the problem of forecasting shots to goal in a soccer match, which is modeled as a pair (one for each team) of time-evolving graphs endowed with time-dependent node features. We propose a Graph Neural Network architecture to handle temporal sequences of graph pairs. Moreover, we employ an explainability algorithm to assess player importances based on the proposed GNN. In Section 4.1 we first discuss related work, in Section 4.2 we describe the dataset and the data engineering we developed aiming to build the pair of graph sequences. The proposed GNN model is introduced in Section 4.3 and in Section 4.4 we describe how to derive player's importance. Finally, we analyze a number of different case studies and compare our methodology with baselines in Section 4.5.

4.1 Related Works

Sports analytics is a field of growing interest, mainly due to the increasing amount of data and financial resources involved in this market. In fact, in the last few years, many analytical methods have been developed for different sports, such as American football (SABIN, 2021), baseball (SUN; LIN; TSAI, 2023; ONO; DIETRICH; SILVA, 2018), basketball (BORNN *et al.*, 2017), ice hokey (LIU *et al.*, 2020), water polo (ARGUDO *et al.*, 2021), rugby (ZHANG; GAO; XIANG, 2020) and soccer (SINGH, 2019; DECROOS *et al.*, 2019; YAM, 2019; MACKAY, 2017). In order to better contextualize our approach, we focus the following related work discussion on techniques designed specifically for analyzing soccer players and matches. A more comprehensive overview can be found in recent sports analytics surveys (GHOSH *et al.*, 2023; SINGH, 2020).

Extracting meaningful insights from soccer data is particularly challenging due to the low-scoring nature of this sport and the complex interaction between the players. Pass networks have been one of the main resources used to model the relationship between team members. A pass network is a graph where nodes and edges correspond to players and passes, respectively. Pass networks are built by fixing a time window and linking players (nodes) who have exchanged passes within the time window interval. The time window can comprehend the entire match time (KAWASAKI *et al.*, 2019; AHMADI; NOORI; TEIMOURPOUR, 2020; MEDINA *et al.*, 2021) or it may be more granular, comprising intervals of 2- to 5-minutes to enable the analysis of the dynamics of players over the match (CAO, 2023).

Building a pass network requires previously recorded data and recent works have relied on either on-the-ball events or player-tracking records. The former describes the time, location, and action type (e.g. passes, shots, dribbles) of players when in contact with the ball (SINGH, 2019; DECROOS *et al.*, 2019; PAPPALARDO *et al.*, 2019; LIU *et al.*, 2022), i.e., only actions of the players with the ball are recorded over time. The latter handles information about the position of all the players and the ball during the whole match (SPEARMAN, 2018; FERNÁNDEZ; BORNN; CERVONE, 2019; DICK; TAVAKOL; BREFELD, 2021), with no information about the action accomplished by the player. The possibility of analyzing player actions and the greater availability of data associated with leagues and clubs have leveraged the development of on-the-ball methods.

Different tasks can be accomplished by exploiting a pass network built based on soccer data, including the estimation of match results (MEDINA *et al.*, 2021), finding a correlation between match events (CAO, 2023), and mining subgraphs for tactical analysis (AHMADI; NOORI; TEIMOURPOUR, 2020). Most of those analyses derive from conventional statistical and graph-based measures applied on pass networks. However, recent works in different areas have shown that GNNs tend to outperform such classical approaches (JIN *et al.*, 2020; ZHANG *et al.*, 2022; YANG *et al.*, 2023; JIANG *et al.*, 2023).

The work by Anzer *et al.* (2022) is a good example of the use of GNN to analyze soccer data, which relies on Graph Attention Networks (GAT) (VELIČKOVIĆ *et al.*, 2017) to detect tactical soccer patterns. Yılmaz and Öğüdücü (2022) propose a graph embedding technique to learn soccer player features in order to build a recommendation mechanism for in-game substitutions. Dick, Tavakol and Brefeld (2021) propose one of the few GNN-based approaches for evaluating soccer players, making use, however, of player-tracking data.

In the context of on-the-ball events for deriving player's importance, Pappalardo *et al.* (2019) propose a linear model to learn the weight of each type of action (e.g. pass, shot, dribble), scoring players according to the weighted average of his/her actions over the match. Liu *et al.* (2022) introduce the Risk-sensitive Game Impact Metric (RiGIM), a measurement mechanism that can be parameterized to weight offensive and defensive players. Other two well-

established techniques that rely on on-the-ball events for scoring soccer players are Expected Threat (xT) (SINGH, 2019) and Valuing Actions by Estimating Probabilities (VAEP) (DECROOS *et al.*, 2019). xT computes players' scores from a Markov chain built from players' actions, while VAEP relies on feature vectors derived from the players' actions and a predictive model to estimate the ratings. Important to notice that none of the mentioned techniques consider the collective action of the players in their evaluation, aggregating only individual actions. Not considering the interaction between players can result in scoring an athlete who only receives and passes the ball the same as another player who works with several teammates to set up plays. Therefore, it is fundamental for assessing players' performance that the interplay between teammates is taken into account.

4.2 Dataset and Data Engineering

4.2.1 Wyscout Dataset

The dataset we make use is the on-the-ball publicly available Wyscout[1] soccer data, obtained from the *socceraction*[2] python library. This dataset contains 1786 soccer matches from 5 different competitions (English, Spanish, French, German, and Italian 2017 - 2018 national leagues). Each game is described as a sequence of events, which are actions (e.g. passes, shots, dribbles) of players when in contact with the ball, as Figure 12 illustrates.

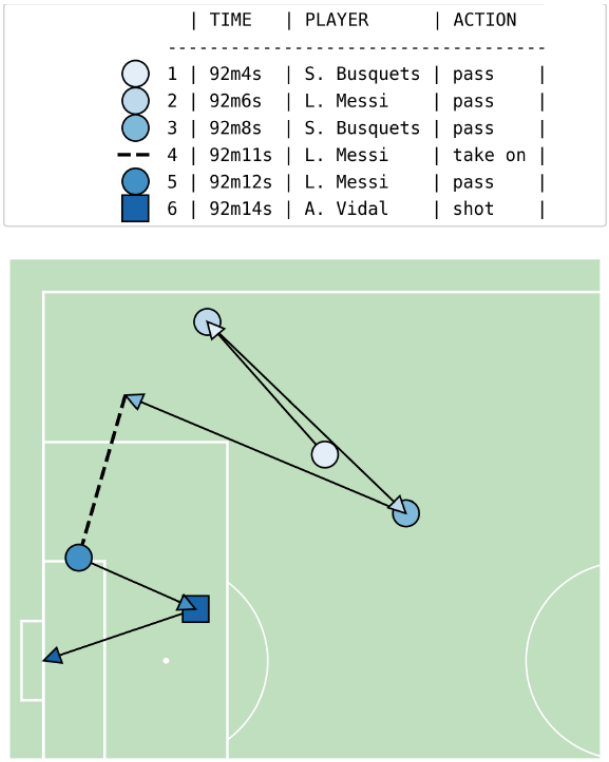
In this dataset, an average of 1200 player actions are registered for each match, totaling more than 2 million actions for the entire database. Moreover, the actions are represented in the SPADL (Soccer Player Action Description Language) format (DECROOS *et al.*, 2019), i.e. each action record contains information about: game ID, period ID (1^{st} or 2^{nd} half of the game), action time, player ID, team ID, start and end (x, y) position, action type (e.g. pass, dribble, shot), action result (success or fail), and bodypart (e.g feet, head).

4.2.2 Dynamic Graph Construction

The dynamic pass network is built by setting non-overlapping one-minute time windows and associating a pair of graphs (one for each team) to each time window. The nodes of each graph correspond to the players of the team and the edges represent passes between teammates that took place within the time window. The edges are weighted according to the number of passes, that is, edge weights are the number of passes between two players in the time window. Therefore, each match gives rise to two dynamic pass networks with at least 90 graphs each (a

[1] <https://wyscout.com/>
[2] <https://socceraction.readthedocs.io/en/latest/index.html>

Figure 12 – A sequence of actions that led to a goal for Barcelona. Only actions of the players with the ball are recorded over time.



Source: (DECROOS *et al.*, 2019), modified by the author of this dissertation.

match usually takes longer than 90 minutes), resulting in a total of 167,781 pairs of graphs for the whole dataset.

Each node (player) in the one-minute graph is associated with a 4-dimensional feature vector containing: the (x, y) averaged position of the player in the corresponding minute, the ball possession time of the player within the time window, and the number of times the player shot to goal in the time window. To obtain the last feature we count the number of actions of type "shot"; as for the player possession time, first we compute the time elapsed between all consecutive actions, and then we aggregate those duration for every player. The location feature is trickier to define since we only have access to on-the-ball event data. We proceed as follows: first, we estimate player position by averaging the (x, y) data of every action done by that player in the referred time step. In the absence of actions done by some player in some time step, we reassign for them the same position as estimated in the previous time window, which is the most updated location of that athlete we have access to. For the first minute of the game, if some player does not take any action, we regard his position as the average location of all their actions from the last game he played along the championship. Finally, for the first minute of the first game of the competition, if some player did not take any action, then we randomly assign a position for him by sampling from a uniform distribution supported in their team's half of the pitch.

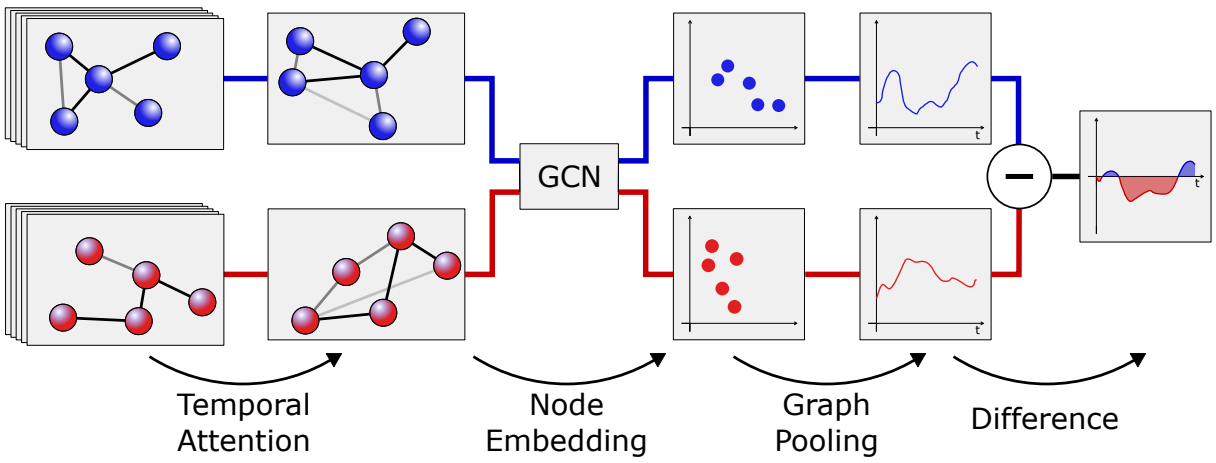
From the Graph Neural Networks perspective, the proposed graph modeling scheme is an interesting problem to tackle.First, the graph is dynamic at the edge level since the interplay between players is likely to change over the game. Second, the graph nodes usually also change along the soccer match due to substitutions or red cards (in which case also changes the number of graph nodes). Third, the features associated with the nodes are also time-dependent, and finally, the model entails concurrently handling two dynamic graphs, a practice not commonly found in current modeling approaches.

4.3 Proposed GNN Architecture

The idea is to build a machine learning model that takes as input a pair of graphs in a given time step and predicts the number of shots on goal one team will make more than the other in the next time step. More specifically, let $\Gamma^{(Z)} = \{\mathcal{G}_1^{(Z)}, \ldots, \mathcal{G}_n^{(Z)}\}$ be a dynamic pass network where $\mathcal{G}_t^{(Z)}$ is the graph in the time window t with node feature vectors from the team $Z \in \{A, B\}$. In order to predict the difference Δs_{t+1} of the number of shots on goal between teams A and B in the instant $t+1$, the model takes as input two sequences of graphs $\mathcal{G}_{t-T+1}^{(Z)}, \ldots, \mathcal{G}_t^{(Z)}$ from Γ_Z, $Z \in \{A, B\}$, where $T \neq 0$ defines the number of graphs in the input sequence. Such a model can mathematically be stated as

$$\Delta s_{t+1} = f\left(\left(\mathcal{G}_{t-\Delta t}^{(A)}\right)_{\Delta t=0}^{T-1}, \left(\mathcal{G}_{t-\Delta t}^{(B)}\right)_{\Delta t=0}^{T-1} \right), \tag{4.1}$$

Figure 13 – The proposed Graph Neural Network architecture to handle dynamic graph pairs. The input is a pair of graph sequences that represent the game dynamics for each team, and the output is a prediction of the difference between shots to goals in the next minute.



where each graph $\mathscr{G}_t^{(Z)} \in \Gamma^{(Z)}$ is given by the pair $\mathscr{G}_t^{(Z)} = (G_t^{(Z)}, X_t^{(Z)})$ where $G_t^{(Z)}$ is the graph of team Z at time step t and X is the $N \times F$ matrix encoding the F-dimensional feature vectors associated with the N nodes of $G_t^{(Z)}$.

In our context, the model f is a GNN whose architecture is depicted in Figure 13. The proposed GNN has four sequential layers: Temporal Attention, Node Embedding, Graph Pooling, and Difference, which we detail in the following.

Temporal Attention. The first layer aims to summarize the temporal sequence of T graphs and feature vectors $\left(\mathscr{G}_{t-\Delta t}^{(Z)}\right)_{\Delta t=0}^{T-1}$ into a single graph with feature vectors $\mathscr{G}^{(Z)} = (G^{(Z)}, X^{(Z)})$ by learning a weighted aggregation induced by the learnable attention coefficients $a_{\Delta t}$. First, we show how the summarized graph $G^{(Z)}$ is built and then we proceed to the matrix of feature vectors $X^{(Z)}$. The summarized graph $G^{(Z)}$ has the same nodes (soccer players) as vertices in the last graph $G_t^{(Z)}$ of the input sequence. The edge weights of the summarized graph are given by the attention mechanism applied to the sequence of edge weights after a $softmax$ activation in $a_{\Delta t}$. Specifically, the edge weight $e^{(i,j)}$ in $G^{(Z)}$ is given by

$$e^{(i,j)} = \sum_{\Delta t=0}^{T-1} softmax(a_{\Delta t})\ e_{\Delta t}^{(i,j)}, \tag{4.2}$$

where $softmax(a_{\Delta t}) = \exp(a_{\Delta t}) / \sum_{\Delta t'=0}^{T-1} \exp(a_{\Delta t'})$, and $e_{\Delta t}^{(i,j)}$ is the edge weight between nodes $i,\ j$ in graph $G_{t-\Delta t}^{(Z)}$. Notice that if two nodes $i,\ j$ are never connected in the graphs sequence, then $e_{\Delta t}^{(i,j)} = 0$ for every Δt and thus $e^{(i,j)} = 0$ in the summarized graph $G^{(Z)}$, i.e., there will be no edge between nodes i and j.

Analogously, the summarized feature matrix $X^{(Z)}$ is a weighted aggregation given by the same attention coefficients $a_{\Delta t}$ after a *softmax* activation function, i.e. the i-th dimension of the feature vector associated to a node k is given by

$$x^{(k,i)} = \sum_{\Delta t=0}^{T-1} softmax(a_{\Delta t})\ x_{\Delta t}^{(k,i)}. \tag{4.3}$$

Alternatively, we can fix the attention coefficients $a_{\Delta t} = 1/T$ instead of learning them, in which case the summarized graph and feature matrix are simply the arithmetic average of the input temporal sequence. Moreover, we only apply the temporal attention coefficients to time windows where they make sense, i.e. if some player is in the game for less than T time steps (either at the first minutes of the match or after some substitution), we restrict the attention mechanism for only the known periods for that player.

Node Embedding. Given the pair of summarized graphs and feature matrices as input, the node embedding is accomplished by a sequence of graph convolution layers. The goal is to learn a

mapping of the vertices in the summarized graph $G^{(Z)}$ (players in the match) to a latent space. We employ a 2-layered Graph Convolutional Network (GCN) (KIPF; WELLING, 2016a) with input, hidden, and output dimensions F, $F-1$, $F-2$, respectively, i.e., we linearly decay the dimension from $F=4$ to 2 to generate compact latent representation, which, consequently, facilitate to visualize the embeddings.

Graph Pooling. The third layer is a graph pooling layer that aggregates the feature vectors of all nodes in a single number. Mathematically, if $X^{(Z)}_{emb}(n,h)$ is the h^{th} dimension of the n^{th} player embedded representation of team Z, then the pooling operation results in

$$P^{(Z)} = \sum_{n,h} X^{(Z)}_{emb}(n,h)\,, \tag{4.4}$$

where n,h ranges over N,H, the total number of players and the latent space dimension (two-dimensional in our context), respectively. This is a common operation for Graph Machine Learning tasks when the target regards the whole graph instead of node- or edge-level predictions (e.g. team shots to goal, instead of player shots to goal). Therefore, the layer output is a single number for each team. As discussed in one of the provided case studies, the number resulting from the pooling layer can be interpreted as the team's offensive and defensive strength in the given time step (see Section 4.5).

Difference. The previous layers apply the same computation for both teams since all the learned attention coefficients and GCN weights are precisely the same. Moreover, the Graph Pooling layer is a parameter-free operation. Therefore, the last layer makes the difference between the graph pooling output values for each team, being this difference, the output of the model, which is trained to match Δs_{t+1}, the amount of shots team A will make more than team B in the next time interval.

The proposed architecture has a number of interesting properties. First, it is robust to changes in the number of graph nodes since no assumption is made in terms of keeping the number of nodes fixed when defining the layers (i.e. the model output is still well-defined even if a node vanishes in the graph sequence due to a red card). Moreover, the temporal attention mechanism is applied only for time windows in which the node is presented in the graph sequence and both node embedding and the graph pooling steps are able to be applied for graphs of variable sizes. Second, the node embedding and graph pooling operations do not depend on the order in the teams A and B are presented to the network, i.e. both $f\big((\mathscr{G}^{(A)}_{t-\Delta t})^{T-1}_{\Delta t=0}, (\mathscr{G}^{(B)}_{t-\Delta t})^{T-1}_{\Delta t=0}\big)$ and $f\big((\mathscr{G}^{(B)}_{t-\Delta t})^{T-1}_{\Delta t=0}, (\mathscr{G}^{(A)}_{t-\Delta t})^{T-1}_{\Delta t=0}\big)$ generates the same embeddings $X^{(Z)}_{emb}$ and graph pooling $P^{(Z)}$ for the teams. Finally, an immediate consequence of the previous observation is that $f\big((\mathscr{G}^{(A)}_{t-\Delta t})^{T-1}_{\Delta t=0}, (\mathscr{G}^{(B)}_{t-\Delta t})^{T-1}_{\Delta t=0}\big) = -f\big((\mathscr{G}^{(B)}_{t-\Delta t})^{T-1}_{\Delta t=0}, (\mathscr{G}^{(A)}_{t-\Delta t})^{T-1}_{\Delta t=0}\big)$. The content of this mathematical identity is that if A is predicted to shoot to goal Δs more than B, then B is

predicted to shoot to goal $-\Delta s$ more than A.For instance, in a Barcelona against Real Madrid match, there is no natural ordering of the competing teams. Therefore, if a user provides the Barcelona-Real Madrid dynamic graphs to the model and another user inputs the data in the other ordering, then the model is guaranteed to output the opposite number to both users.

4.4 GNN-VSP: Assessing Player's Performance

As discussed in Section 4.3, the proposed GNN architecture learns to predict, for each time step, the number of shots on goal one team makes more than the other in the next time window. The proposed architecture in Figure 13 shows that if the result of the graph pooling goes up for one team, the number of shots to the goal also increases for that team with a decrease for the opponent. Therefore, the pooling output encodes both the offensive and defensive performance of each team.

In order to understand the contribution of each player to the team performance, we employ GNNExplainer (YING *et al.*, 2019) (see Appendix B), a graph explainability algorithm. Specifically, let $\mathscr{G}_t^{(Z)} = (G_t^{(Z)}, X_t^{(Z)})$ be the summarized graph $G_t^{(Z)}$ and the feature matrix $X_t^{(Z)}$ resulting from the temporal attention layer for a fixed time step t. Let $P_t^{(Z)}$ be the "performance" of team Z resulting from the graph pooling operation in the same instant t. GNNExplainer assigns an importance $\mathscr{I}(p,i)$ for each feature i of player p in $\mathscr{G}_t^{(Z)}$ when predicting $P^{(Z)}$. Therefore, the "relevance/importance" score of a player p at time t of the game can be computed by simply aggregating $\mathscr{I}(p,i)$ over the features. In mathematical terms:

$$\text{GNN-VSP}_t(p) = \sum_i \mathscr{I}(p,i)\,. \tag{4.5}$$

We evaluate the relevance of each player over the entire competition by aggregating his/her score as given in Equation 4.5 over every time step of all matches s/he played. To make the comparison fair we normalize the aggregated scores per 90 minutes played, as typically done in the literature (SINGH, 2019; DECROOS *et al.*, 2019; ROY *et al.*, 2020). Such a normalization reduces the bias in favor of players who played longer throughout the championship. In mathematical terms, the GNN-VSP worth of a player p is given by

$$\text{GNN-VSP}(p) = \frac{90}{\mathbb{T}} \sum_t \text{GNN-VSP}_t(p)\,, \tag{4.6}$$

where $\mathbb{T}$ is the total time steps that player p played over the whole competition.

We can analogously assess the importance of each feature by aggregating $\mathscr{I}(p,i)$ over the players, i.e., the importance of a feature i is $\sum_p \mathscr{I}(p,i)$.

Equation 4.6 normalizes the relevance of a player per 90 minutes. For example, if a player p_1 plays only for a few minutes, then the normalization factor ${}^{90}/_{\mathbb{T}}$ contributes to increasing the score of p_1, but the summation in Equation 4.6 aggregates only for a few time windows. In contrast, another player p_2 that played for almost the whole competition has more time steps to sum his scores, but also a larger value of $\mathbb{T}$. Therefore, this normalization makes the comparison among players fair.

4.5 Experiments and Comparisons

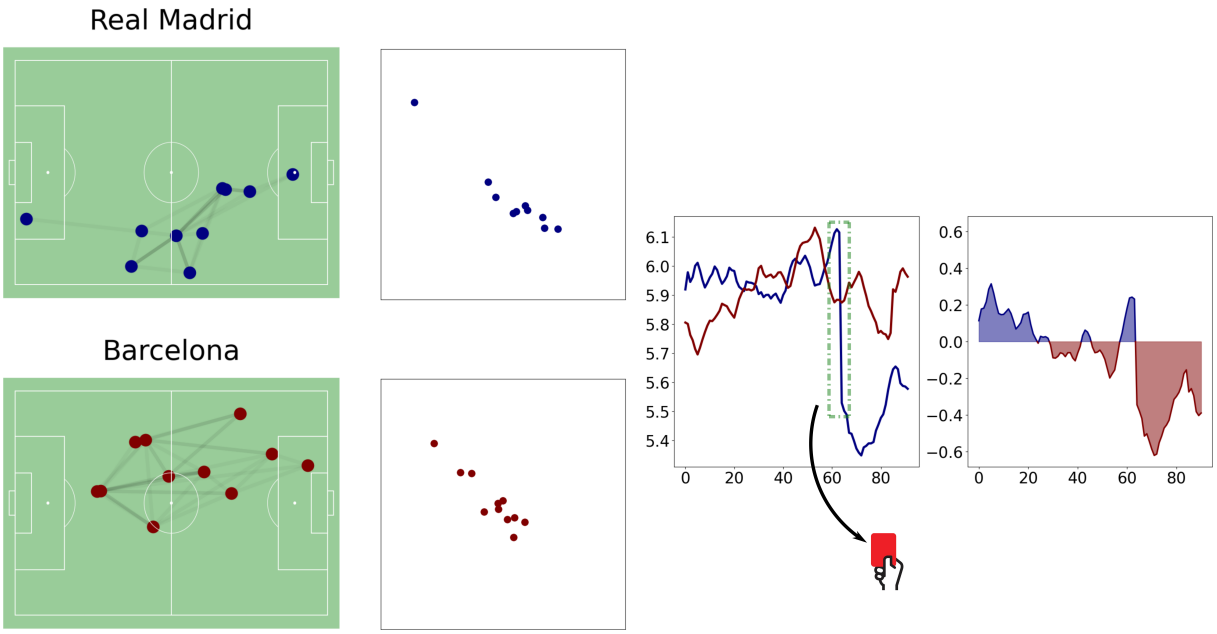
4.5.1 Experiment Setup

For the experiments, we trained the proposed GNN architecture by randomly splitting the 167,781 samples into training / validation / test subsets by the proportion of 70% / 15% / 15% respectively. We tried batch sample sizes of $\{64, 128, 256\}$ and optimize the parameters employing Adam (KINGMA; BA, 2014) for 100 epochs, 5 maximum patience, learning rate in $\{0.001, 0.01, 0.1\}$ and MSE loss function. The GCN activation function is set as $\text{LeakyReLU}(x) = max(0,x) + \alpha\ min(0,x)$ ($\alpha = 0.01$) and we select the hyperparameter configuration that led to the best performance in the validation subset.

It is not possible to use ELM-GCN introduced in Chapter 3 to train the proposed architecture in this context. We remind the reader that ELM-GCN is a training mechanism suitable for 2-layered GCNs employed in a single static graph with node feature vectors and desired predictions at the node level. In the context of this Chapter, this approach is different in several aspects. First, the model is not purely a 2-layered GCN, but a more sophisticated architecture with also temporal attention mechanism, graph pooling, and difference layers. Second, the graph is not static, nor is it a single graph. Instead, the input is a temporal sequence of dynamic graphs. Third, the input in the context of this Chapter also comes in pairs, which is not accounted for in the ELM-GCN. Finally, the target of the architecture of Figure 13 is a single number for every pair of graph sequences, not matching the node-level prediction of the ELM-GCN case.

Regarding the training of the explainability algorithm, GNNExplainer is applied to generate attribute explanations for 300 epochs and 0.01 learning rate. This type of explanation (attribute explanations) provides the importance $\mathscr{I}(p,i)$ for each feature i of each node (i.e. player) p.

Figure 14 – GNN architecture predictions for Real Madrid vs Barcelona in 23/Dec/2017 for the Spanish championship. Real Madrid (in blue) had a red card for a player at minute 63 and the model started predicting that Barcelona would shoot to goal considerably more.



4.5.2 GNN Qualitative Evaluation

In this Section, we provide a case study analyzing how the model captures the impact of red cards on the teams' performance. We also analyze feature importance in the prediction process.

The Impact of Red Cards. We analyze the impact of red cards taking as a case study a match (see Figure 14) between Real Madrid (in blue) and Barcelona (in red) on December 23, 2017, for the Spanish championship. We used *matplotsoccer*[3] python library for the soccer pitch visualization. In this analysis, we trained the GNN model keeping the attention coefficients fixed at $1/T$ to compute the graph sequence summarization. At minute 63 the GNN architecture predicts Real Madrid's offensive and defensive power drops suddenly, remaining below Barcelona for the rest of the match. At 63 minutes of the match, Real Madrid's player Dani Carvajal got a red card, which impacted substantially Real Madrid's performance. The proposed GNN model was capable of capturing this game-changing event, predicting a difference in the number of shots on goal much more favorable to Barcelona Analyzing the original data, we indeed see a game change in favor of Barcelona, since before the red card both teams had shot to goal 7 times (and the game score was Real Madrid 0 - 1 Barcelona); whereas after the expulsion, Barcelona team shot 9 more times, against only 4 from Real Madrid (Barcelona converted 2 more of them into goals and ending the game Real Madrid 0 - 3 Barcelona). In other words, even though red cards

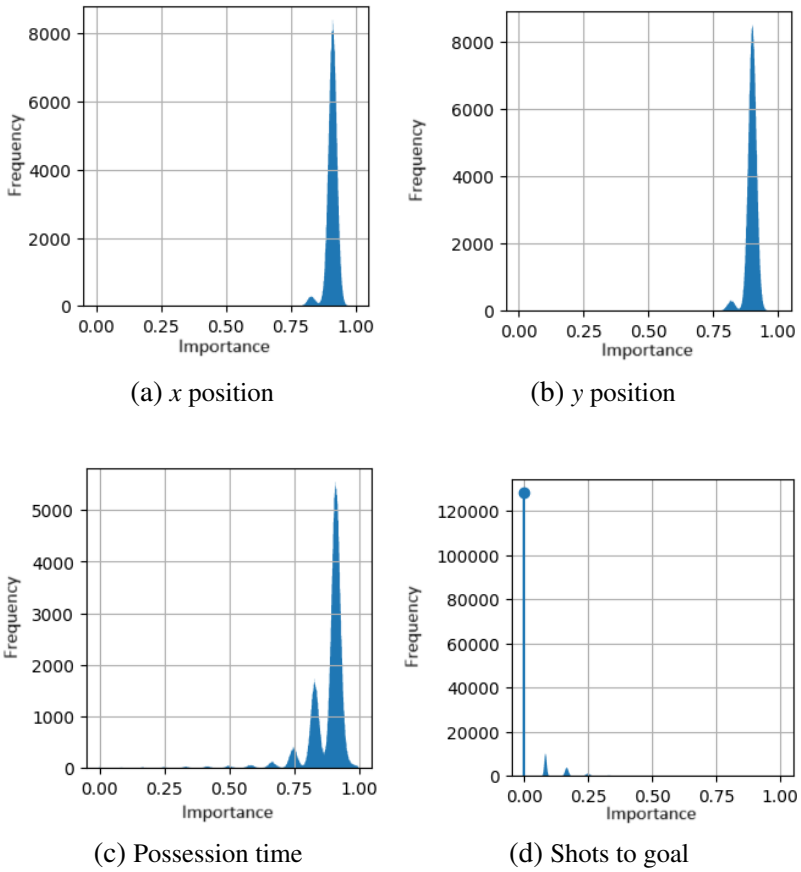
[3] <https://pypi.org/project/matplotsoccer/>

are not seen by the model, the change in the team behavior after the red card was captured by the proposed GNN, shows its capability in representing the dynamics of the teams.

Feature Importance As discussed in Section 4.4, given a fixed time step of some game we can assess the importance of feature i by aggregating over the players the importance $\mathscr{I}(p,i)$ given by the explainability algorithm. We have computed the feature importances in each minute of all matches over the five different competitions in the dataset. For this analysis, the attention coefficients of the proposed GNN architecture were set as learnable parameters.

Figure 15 depicts the distribution of each feature's importance for the whole dataset. Note that all features present multi-modal distributions, due to the different temporal attention coefficients learned by the model. From the distributions, one can clearly notice that players' position and ball possession time are more relevant to forecast shots to goal than previous shootings. In fact, in most cases, previous shootings have received no importance by the explainability algorithm. We believe this is a consequence of the sparsity of this feature.

Figure 15 – Feature importance distribution. Players' positioning and possession time are more important to predicting next-minute shots to goal than previous shootings.



(a) x position

(b) y position

(c) Possession time

(d) Shots to goal

4.5.3 Players' Performance Assessment

In this Section, we compare the proposed player scoring methodology GNN-VSP against xT (SINGH, 2019) and VAEP (DECROOS *et al.*, 2019), two well-known techniques for evaluating soccer players. We perform the comparisons in terms of: (1) the score distributions, (2) a correlation between the scores provided by the three methods, and (3) a case study involving awarded soccer players.

For the experiments, we configured the proposed GNN architecture with learneable attention coefficients and we used xT and VAEP implementations from socceraction[4]. Moreover, we considered the whole dataset for the experiments, i.e. all the 1786 soccer matches along the five available competitions. We rate only athletes who played more than 360 minutes over the season and we normalize the scores given by xT, VAEP, and GNN-VSP per 90 minutes of game played.

xT (Expected Threat). This approach discretizes the soccer field using a $M \times N$ regular grid, where each of the MN cells is called a zone. A score associated with threatening to shoot on goal is assigned to each zone and players are ranked based on whether s/he move the ball to higher-scored zones over the matches. In this context, the threatening of a zone z is derived from the chance of a player scoring a goal by shooting from zone z, plus the threatening of the zone z' to which the player moves the ball from z. This recursive definition induces a stochastic matrix that is built from the on-the-ball data, and the scores are given by the steady state of the corresponding Markov chain, as

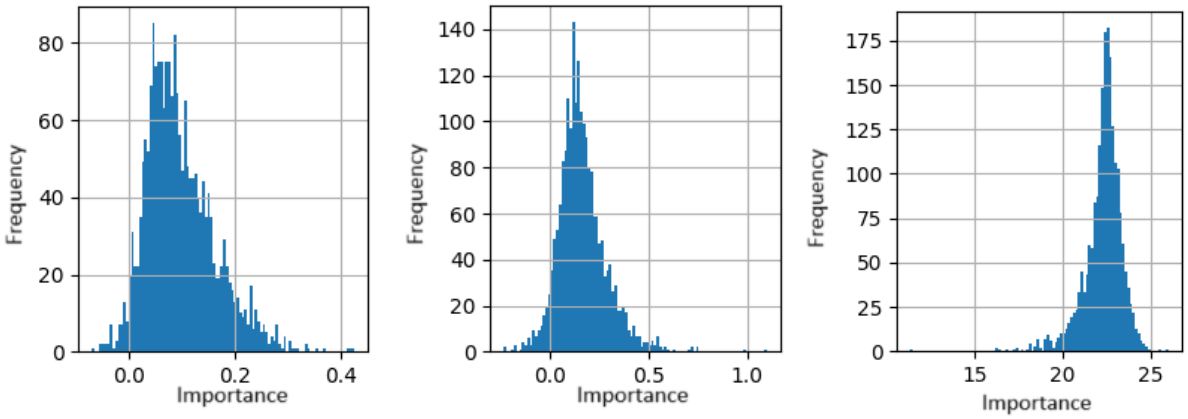
$$\mathrm{xT}(z) = s_z \cdot \mathrm{xG}(z) + m_z \cdot \sum_{z'} T_{z \to z'} \cdot \mathrm{xT}(z')\,, \tag{4.7}$$

where: s_z is probability of shooting to goal from zone z; $\mathrm{xG}(z)$ is the expected goals of zone z, i.e. the probability that a shooting from zone z is converted into a goal; m_z is the probability of moving the ball from zone z; and $T_{z \to z'}$ is the transitioning probability from zone z to zone z'. For more details about the xT method, we refer to Singh (2019).

VAEP (Valuing Actions by Estimating Probabilities). VAEP employs machine learning models to estimate the probability of a team score or conceding a goal given the three last actions performed over a match. A set of features is derived from the three actions, including: information about the actions themselves (location and type of the action, distance, and angle to goal, etc.), the relation between actions (e.g. distance and time between consecutive actions), and the overall context of the match (e.g. time duration of the match and score difference). Those

[4] https://socceraction.readthedocs.io/en/latest/

Figure 16 – Players rating distribution per 90 minutes for each valuing mechanism. The best players are assessed by GNN-VSP only a fraction more than the average athlete, and GNN-VSP never values players with negative numbers.



(a) xT (b) VAEP (c) GNN-VSP

features are fed into two different models to learn the score and concede probability within the next actions. Thus, a match state (a sequence of three actions) is rated according to the difference between scoring and conceding probabilities estimated by the models. An action is valued as the difference between the match states before and after the action is made, and the value of a player is the aggregation of his/her action values. Intuitively, the higher (lower) a player is scored, the more his/her actions contributed to increase (decrease) the team's probability of scoring or decrease (increase) the team's probability of conceding. A more detailed description can be found in the paper by Decroos *et al.* (2019).

We used the default hyperparameters implemented in the socceraction Python library for both xT and VAEP. Specifically, xT overlays a 16×12 grid in the pitch and VAEP employs an XGBoost classifier. In order to train the models, VAEP generates features and labels based on the last 3 and next 10 actions, respectively (i.e. the classifiers learn to estimate the probabilities that a score or concede will happen in the next ten actions, given the features engineered by the last three actions).

Score Distributions. Figure 16 depicts players' score distribution derived from xT, VAEP, and GNN-VSP. We first observe that the magnitude of the values resulting from xT and VAEP are smaller than the ones provided by GNN-VSP. Moreover, in contrast to xT and VAEP, GNN-VSP concentrates the players' scores on the right of the histogram, meaning that, on average, the players are highly scored, which is expected in competitions involving high-level players, as the ones involved in the used dataset.

Table 5 – Correlation between valuing mechanisms. GNN-VSP has a positive but weak correlation with both xT and VAEP, thus it can be used along with techniques from literature to enhance high-level insights.

	xT (p90)	VAEP (p90)	GNN-VSP (p90)
xT (p90)	1.00	0.42	0.15
VAEP (p90)	0.42	1.00	0.22
GNN-VSP (p90)	0.15	0.22	1.00

Moreover, xT and VAEP score players with negative values. Negative values are somewhat misleading, as they can lead to the conclusion those specific players are hindering the team's performance, which is difficult to agree on, as the hiring process in the European leagues is very strict and competitive. GNN-VSP, in contrast, does not assign negative scores to players, as the importance provided by the explanation method is always non-negative.

Correlation with xT and VAEP. Table 5 shows the correlations between the three pairs of methods. Notice that all correlation values in Table 5 are positive, meaning that the three techniques agree when scoring the players. Nonetheless, GNN-VSP has a weaker correlation with both xT and VAEP, indicating that GNN-VSP is assessing players in a different manner, which is expected, as it also takes into account the interaction between team members. This suggests that the proposed approach is a useful tool to be employed along with xT or VAEP to enhance the analytical process.

Best Soccer Players Annual Awards. The most prestigious prizes for male soccer players are the Ballon d'Or (Golden Ball) [5] and The Best FIFA Men's Player [6], hosted by the French magazine France Football and FIFA [7], respectively. In 2017, the first, second, and third positions of both prizes were held by Cristiano Ronaldo, Messi, and Neymar, respectively [8] [9].

[5] https://www.francefootball.fr/ballon-d-or/

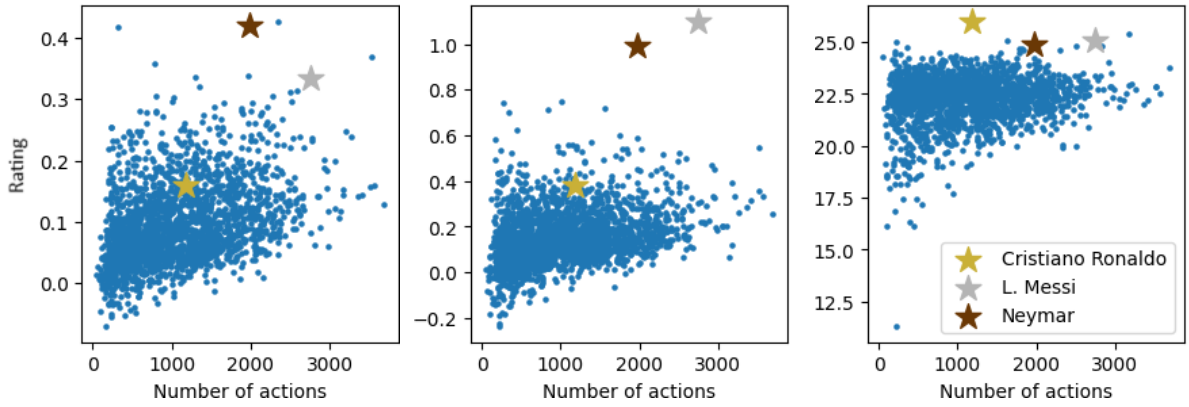
[6] https://www.fifa.com/fifaplus/en/the-best-fifa-football-awards

[7] The Ballon d'Or and The Best FIFA Men's Player happens yearly since 1956 and 1991, respectively. However, from 1991 to 2009 FIFA's award was formally known as FIFA World Player of the Year. From 2010 to 2015 both prizes merged into the FIFA Ballon d'Or and from 2016 they are split into The Best FIFA Men's Player and Ballon d'Or.

[8] Ballon d'Or: <https://www.francefootball.fr/news/Le-classement-complet-du-ballon-d-or-2017/857283>

[9] The Best FIFA Men's Player: <https://web.archive.org/web/20171024095638/http://resources.fifa.com/mm/document/the-best/general/02/91/68/84/fullresults-tbffa_award_rankingpresslist2017_neutral.pdf>

Figure 17 – Players from the English, Spanish, French, German, and Italian championships were assessed via a number of actions and ratings given by xT, VAEP, and GNN-VSP. Cristiano Ronaldo, L. Messi, and Neymar are the top 3 players awarded respectively first, second, and third positions at both the Ballon d'Or (Golden Ball) and The Best FIFA Men's Player in 2017. GNN-VSP is able to clearly distinguish them from other players.



(a) xT (b) VAEP (c) GNN-VSP

Figure 17 shows scatter plots of players with respect to their number of actions along the championship and scores provided by xT, VAEP, and GNN-VSP. The three stars highlight the top three players Cristiano Ronaldo (gold), Messi (silver), and Neymar (bronze), respectively. Notice that GNN-VSP is clearly distinguishing the three best players from the remaining ones, giving them higher scores, while the distinction given by xT and VAEP is not so evident, mainly to Cristiano Ronaldo. In fact, Cristiano Ronaldo, the best 2017 player by both France Football and FIFA, is ranked by xT, VAEP, and GNN-VSP in the 323^{th}, 93^{th} and 1^{st} positions, respectively, showing that the proposed GNN-VSP properly captured the performance of that player.

CHAPTER

5

CONCLUSION AND FUTURE RESEARCH LINES

In this dissertation, we presented the author's contributions to the development of Graph Neural Networks, a family of Machine Learning methods suitable to be employed in problems that can be modeled as graphs endowed with data in its elements. More specifically, we proposed two training algorithms (ELM-GCN / RELM-GCN) and GNN-VSP, a soccer valuing mechanism based on a novel GNN model for dynamic graphs.

ELM-GCN and RELM-GCN are two analytical learning algorithms designed to train 2-layer Graph Convolutional Networks. The proposed extreme learning machine techniques, which are extended for graph-structured domains, come with solid theoretical foundations, providing an upper bound in the number of hidden units needed to reach a desired training error. The experiments in Chapter 3 involving both synthetic and real graph datasets show that RELM-GCN is able to reach an accuracy similar to competing methods, but reducing the training time considerably. In fact, the proposed approach turned out to be at least one order of magnitude faster in most of our experiments.

One of the limitations of ELM-GCN and its regularized version is that it is constrained to 2-layer GCNs. However, this architecture has been a popular choice in many recent works (KIPF; WELLING, 2016a; KIPF; WELLING, 2016b; HAMILTON; YING; LESKOVEC, 2017; CHEN; MA; XIAO, 2018; SHCHUR *et al.*, 2018; ZHAO *et al.*, 2019; JIN *et al.*, 2020), mainly due to the well-known oversmoothing effect (LI; HAN; WU, 2018), as discussed in Section 1.2. Thus ELM-GCN and RELM-GCN turn out to be useful in many application scenarios.

In Chapter 4, we focused on developing a GNN able to handle dynamic graphs, i.e., graphs in which its elements (nodes, edges, and feature vectors) change over time. As a result, we proposed GNN-VSP, a GNN designed to predict goal shots and the importance of players. The scoring mechanism is able to account for the athletes' interplay and it models each match as a pair of dynamic networks in which players are nodes and edges are passes. The proposed GNN architecture predicts the next-minute difference of shots to goal between teams based on time-evolving graphs, relying on an explainability technique to rate the importance of each node (player) as to offensive and defensive actions.

The conducted experiments show that our GNN architecture is able to capture game-changing events and that GNN-VSP is an effective tool for scoring and ranking soccer players. In fact, in one of the case studies our scheme was the only valuing approach to distinguish the top players, given by FIFA and France Football, from the remaining athletes playing in the European leagues.

As future research lines, we highlight a number of possibilities:

ELM-GCN Related. Research papers focusing on other of the classical ELM are still booming (MARTÍNEZ-MARTÍNEZ *et al.*, 2011; ZHANG; LUO, 2015; INABA *et al.*, 2018; SILVA *et al.*, 2020). Generalizing the new achievements while proposing new ones for graph-structured data is a research path we plan to follow. Moreover, since ELM-GCN and RELM-GCN are methods designed for node-level predictions, we also plan to extend them to be employed in other Graph Machine Learning tasks, like graph-level and link predictions.

GNN-VSP Related. We plan to employ our proposed model for dealing with dynamic graphs in other graph-structured domains (KUMAR; ZHANG; LESKOVEC, 2019; KAZEMI *et al.*, 2020; CHEN *et al.*, 2022), e.g. social networks and recommending systems that are changing over time. The author has a large interest in this path because of its robustness in dealing with real-world scenarios (e.g. understanding how blocking street access impacts the urban planning flow dynamics).

Graph Embeddings. Some researches focus on generating embeddings for the data points that are structured in a graph manner (JIN *et al.*, 2016; KIPF; WELLING, 2016b; GROVER; LESKOVEC, 2016; XU, 2021). Those approaches aim to find a low-dimensional representation of the graph data while also preserving the maximum of the structural information. Therefore, they can support a wide range of applications from enhancing insights from data visualization and exploration to pre-processing the information to another downstream classic machine learning algorithm.

Explainability. Another interest of the author is to start contributing in the Machine Learning Explainability area (RIBEIRO; SINGH; GUESTRIN, 2016; LUNDBERG; LEE, 2017). Those classes of techniques focus on opening the black box of Machine Learning models and understanding why they are providing such outputs. The author believes this is a field of increasing importance because while machine learning has been a popular approach to many real-world problems, the models that are being deployed are also becoming progressively more sophisticated.

Fairness in Machine Learning. Fairness in machine learning has emerged as a field that investigates methods to prevent biases in data and inaccuracies in models that could result in discriminatory treatment of individuals based on attributes such as race, gender, disabilities, and sexual or political orientation (CHOULDECHOVA; ROTH, 2020; RUGGIERI *et al.*, 2023). This is an interesting domain of study because, at the same time it is beneficial to rely on machine learning models to support practical tasks, it is also desirable to ensure that they make responsible decisions based on ethics.

5.1 Publications and Submitted Papers

During the Ph.D. period, the author of this dissertation published a paper related to the proposed ELM-GCN / RELM-GCN techniques described in Chapter 3 at the 11^{th} Brazilian Conference on Intelligent Systems (BRACIS 2022) entitled "Extreme Learning Machine to Graph Convolutional Networks" (GONÇALVES; NONATO, 2022).

The paper related to the GNN proposed to handle dynamic graphs and players scoring scheme described in Chapter 4 is called "GNN-VSP: A Graph Neural Network Model for Valuing Soccer Players" and is submitted to the journal Expert Systems With Applications.

During the Ph.D. period, the author of this dissertation also collaborated in two more works that resulted in papers as co-author. The first one is published at the 35^{th} Conference on Graphics, Patterns and Images (SIBGRAPI 2022) and is called "CityHub: A Library for Urban Data Integration" (SALINAS *et al.*, 2022). In this work, we introduce CityHub, a Python library developed to integrate distinct urban data types in a shared spatial domain, namely the graph obtained by the city streets. In our case studies, we make use of CityHub and a proposed visualization tool to understand the underlying relationship among several urban variables of different nature, like crime, socio-economic, infrastructure, and weather variables. The second one, entitled "EXplainable Artificial Intelligence (XAI) – From Theory to Methods and Applications", is published in the journal IEEE Access (ORTIGOSSA; GONÇALVES; NONATO, 2024). This work is a review that discusses the theoretical foundations of explainability algorithms, supporting the identification of objectives, challenges, and future research lines of XAI. We show state-of-the-art approaches such as LIME, SHAP, and other explainability algorithms to Graph Neural Networks. Such papers are not described in this dissertation as being a collaboration with members of the research group.

BIBLIOGRAPHY

ACHARYA, D. B.; ZHANG, H. Feature selection and extraction for graph neural networks. In: **Proceedings of the ACM Southeast Conference**. [S.l.: s.n.], 2020. p. 252–255. Citation on page 27.

AHMADI, A. H.; NOORI, A.; TEIMOURPOUR, B. Social network analysis of passes and communication graph in football by mining frequent subgraphs. In: IEEE. **Proceedings of the ICWR International Conference on Web Research**. [S.l.], 2020. p. 1–7. Citation on page 62.

AHMED, N. M.; CHEN, L.; WANG, Y.; LI, B.; LI, Y.; LIU, W. Sampling-based algorithm for link prediction in temporal networks. **Information Sciences**, Elsevier, v. 374, p. 1–14, 2016. Citation on page 29.

ANZER, G.; BAUER, P.; BREFELD, U.; FASSMEYER, D. Detection of tactical patterns using semi-supervised graph neural networks. In: **Proceedings of the MIT Sloan Sports Analytics Conference**. [S.l.: s.n.], 2022. v. 16, p. 1–3. Citation on page 62.

ARAZZI, M.; COTOGNI, M.; NOCERA, A.; VIRGILI, L. Predicting tweet engagement with graph neural networks. In: **Proceedings of the ACM International Conference on Multimedia Retrieval**. [S.l.: s.n.], 2023. p. 172–180. Citation on page 27.

ARGUDO, F. M.; MARIN, P. G.; HERNANDEZ, P. J. B.; RUÍZ-LARA, E. Influence of rule changes on shooting performance in balanced matches between two european water polo championship. **International Journal of Performance Analysis in Sport**, Taylor & Francis, v. 21, n. 1, p. 61–73, 2021. Citation on page 61.

BANERJEE, K. **Generalized inverse of matrices and its applications**. [S.l.]: Taylor & Francis Group, 1973. Citations on pages 41 and 89.

BARANWAL, A.; FOUNTOULAKIS, K.; JAGANNATH, A. Graph convolution for semi-supervised classification: Improved linear separability and out-of-distribution generalization. **arXiv preprint**, 2021. Citation on page 28.

____. Effects of graph convolutions in deep networks. **arXiv preprint**, p. arXiv–2204, 2022. Citation on page 28.

BORNN, L.; CERVONE, D.; FRANKS, A.; MILLER, A. Studying basketball through the lens of player tracking data. In: **Handbook of statistical methods and analyses in sports**. [S.l.]: Chapman and Hall/CRC, 2017. p. 261–286. Citation on page 61.

CAO, S. Study state dynamics of team passing networks in soccer games. **Journal of Sports Sciences**, Taylor & Francis, p. 1–15, 2023. Citation on page 62.

CHANG, X.; LIU, X.; WEN, J.; LI, S.; FANG, Y.; SONG, L.; QI, Y. Continuous-time dynamic graph learning via neural interaction processes. In: **Proceedings of the ACM International Conference on Information and Knowledge Management**. [S.l.: s.n.], 2020. p. 145–154. Citation on page 29.

CHEN, J.; MA, T.; XIAO, C. Fastgcn: fast learning with graph convolutional networks via importance sampling. **arXiv preprint**, 2018. Citations on pages 45, 55, 56, and 77.

CHEN, J.; PAREJA, A.; DOMENICONI, G.; MA, T.; SUZUMURA, T.; KALER, T.; SCHARDL, T. B.; LEISERSON, C. E. **Evolving graph convolutional networks for dynamic graphs**. [S.l.]: Google Patents, 2022. US Patent 11,537,852. Citation on page 78.

CHIANG, W.-L.; LIU, X.; SI, S.; LI, Y.; BENGIO, S.; HSIEH, C.-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2019. p. 257–266. Citation on page 45.

CHOULDECHOVA, A.; ROTH, A. A snapshot of the frontiers of fairness in machine learning. **Communications of the ACM**, ACM New York, NY, USA, v. 63, n. 5, p. 82–89, 2020. Citation on page 79.

COL, A. D.; VALDIVIA, P.; PETRONETTO, F.; DIAS, F.; SILVA, C. T.; NONATO, L. G. Wavelet-based visual analysis of dynamic networks. **IEEE TVCG Transactions on Visualization and Computer Graphics**, IEEE, 2017. Citation on page 35.

DECROOS, T.; BRANSEN, L.; HAAREN, J. V.; DAVIS, J. Actions speak louder than goals: Valuing player actions in soccer. In: **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2019. p. 1851–1861. Citations on pages 61, 62, 63, 64, 68, 72, and 73.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. **The Journal of Machine Learning Research**, JMLR.org, v. 7, p. 1–30, 2006. Citation on page 58.

DENG, W.; ZHENG, Q.; CHEN, L. Regularized extreme learning machine. In: IEEE. **Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining**. [S.l.], 2009. p. 389–395. Citation on page 43.

DICK, U.; TAVAKOL, M.; BREFELD, U. Rating player actions in soccer. **Frontiers in Sports and Active Living**, Frontiers, p. 174, 2021. Citation on page 62.

FERNÁNDEZ, J.; BORNN, L.; CERVONE, D. Decomposing the immeasurable sport: A deep learning expected possession value framework for soccer. In: **Proceedings of the MIT Sloan Sports Analytics Conference**. [S.l.: s.n.], 2019. v. 13. Citation on page 62.

GHOSH, I.; RAMAMURTHY, S. R.; CHAKMA, A.; ROY, N. Sports analytics review: Artificial intelligence applications, emerging technologies, and algorithmic perspective. **Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery**, Wiley Online Library, p. e1496, 2023. Citation on page 61.

GONÇALVES, T.; NONATO, L. G. Extreme learning machine to graph convolutional networks. In: SPRINGER. **Proceedings of the BRACIS Brazilian Conference on Intelligent Systems**. [S.l.], 2022. p. 601–615. Citations on pages 48 and 79.

GROVER, A.; LESKOVEC, J. node2vec: Scalable feature learning for networks. In: **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2016. p. 855–864. Citation on page 78.

HAMILTON, W. L.; YING, R.; LESKOVEC, J. Inductive representation learning on large graphs. **arXiv preprint**, 2017. Citations on pages 45 and 77.

HAMMOND, D. K.; VANDERGHEYNST, P.; GRIBONVAL, R. Wavelets on graphs via spectral graph theory. **Applied and Computational Harmonic Analysis**, Elsevier, v. 30, n. 2, p. 129–150, 2011. Citation on page 35.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: **Proceedings of the IEEE CVPR Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2016. p. 770–778. Citations on pages 28 and 39.

HISANO, R. Semi-supervised graph embedding approach to dynamic link prediction. In: SPRINGER. **Proceedings of the Conference on Complex Networks CompleNet**. [S.l.], 2018. p. 109–121. Citation on page 29.

HUANG, G.-B.; ZHU, Q.-Y.; SIEW, C.-K. Extreme learning machine: theory and applications. **Neurocomputing**, Elsevier, v. 70, n. 1-3, p. 489–501, 2006. Citations on pages 31, 40, 41, 42, 48, and 56.

IBRAHIM, N. M. A.; CHEN, L. Link prediction in dynamic social networks by integrating different types of information. **Applied Intelligence**, Springer, v. 42, p. 738–750, 2015. Citation on page 29.

INABA, F. K.; Teatini Salles, E. O.; PERRON, S.; CAPOROSSI, G. DGR-ELM - Distributed Generalized Regularized ELM for classification. **Neurocomputing**, Elsevier B.V., v. 275, p. 1522–1530, 2018. ISSN 18728286. Available: <https://doi.org/10.1016/j.neucom.2017.09.090>. Citation on page 78.

JIANG, W.; LUO, J.; HE, M.; GU, W. Graph neural network for traffic forecasting: The research progress. **ISPRS International Journal of Geo-Information**, MDPI, v. 12, n. 3, p. 100, 2023. Citation on page 62.

JIN, G.; WANG, Q.; ZHU, C.; FENG, Y.; HUANG, J.; ZHOU, J. Addressing crime situation forecasting task with temporal graph convolutional neural network approach. In: IEEE. **Proceedings of the International Conference on Measuring Technology and Mechatronics Automation**. [S.l.], 2020. p. 474–478. Citations on pages 28, 39, 62, and 77.

JIN, Z.; LIU, R.; LI, Q.; ZENG, D. D.; ZHAN, Y.; WANG, L. Predicting user's multi-interests with network embedding in health-related topics. In: IEEE. **Proceedings of the International Joint Conference on Neural Networks**. [S.l.], 2016. p. 2568–2575. Citation on page 78.

KAWASAKI, T.; SAKAUE, K.; MATSUBARA, R.; ISHIZAKI, S. Football pass network based on the measurement of player position by using network theory and clustering. **International Journal of Performance Analysis in Sport**, Taylor & Francis, v. 19, n. 3, p. 381–392, 2019. Citation on page 62.

KAZEMI, S. M.; GOEL, R.; JAIN, K.; KOBYZEV, I.; SETHI, A.; FORSYTH, P.; POUPART, P. Representation learning for dynamic graphs: A survey. **The Journal of Machine Learning Research**, JMLR.org, v. 21, n. 1, p. 2648–2720, 2020. Citations on pages 29 and 78.

KERIVEN, N. Not too little, not too much: a theoretical analysis of graph (over) smoothing. **Proceedings of the NIPS Advances in Neural Information Processing Systems**, v. 35, p. 2268–2281, 2022. Citation on page 28.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint**, 2014. Citation on page 69.

KIPF, T. N.; WELLING, M. Semi-supervised classification with graph convolutional networks. **arXiv preprint**, 2016. Citations on pages 15, 19, 28, 29, 31, 36, 37, 38, 39, 55, 67, and 77.

____. Variational graph auto-encoders. **arXiv preprint**, 2016. Citations on pages 77 and 78.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Proceedings of the NIPS Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2012. p. 1097–1105. Citations on pages 27 and 36.

KÜCHLER, A.; GOLLER, C. Inductive learning in symbolic domains using structure-driven recurrent neural networks. In: SPRINGER. **Proceedings of the Annual Conference on Artificial Intelligence**. [S.l.], 1996. p. 183–197. Citation on page 27.

KUMAR, S.; ZHANG, X.; LESKOVEC, J. Predicting dynamic embedding trajectory in temporal interaction networks. In: **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2019. p. 1269–1278. Citation on page 78.

LI, M.; ZHU, Z. Spatial-temporal fusion graph neural networks for traffic flow forecasting. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2021. v. 35, n. 5, p. 4189–4196. Citation on page 27.

LI, Q.; HAN, Z.; WU, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2018. Citations on pages 28, 39, and 77.

LIBEN-NOWELL, D.; KLEINBERG, J. The link prediction problem for social networks. In: **Proceedings of the International Conference on Information and Knowledge Management**. [S.l.: s.n.], 2003. p. 556–559. Citation on page 29.

LIU, G.; LUO, Y.; SCHULTE, O.; POUPART, P. Uncertainty-aware reinforcement learning for risk-sensitive player evaluation in sports game. **Proceedings of the NIPS Advances in Neural Information Processing Systems**, v. 35, p. 20218–20231, 2022. Citation on page 62.

LIU, G.; SCHULTE, O.; POUPART, P.; RUDD, M.; JAVAN, M. Learning agent representations for ice hockey. **Proceedings of the NIPS Advances in Neural Information Processing Systems**, v. 33, p. 18704–18715, 2020. Citation on page 61.

LUNDBERG, S. M.; LEE, S.-I. A unified approach to interpreting model predictions. **Proceedings of the NIPS Advances in Neural Information Processing Systems**, v. 30, 2017. Citation on page 79.

MACKAY, N. **Improving my 'xG added' model**. 2017. Accessed: 2023-08-02. Available: <https://mackayanalytics.nl/2017/07/28/improving-my-xg-added-model/>. Citation on page 61.

MARTÍNEZ-MARTÍNEZ, J. M.; ESCANDELL-MONTERO, P.; SORIA-OLIVAS, E.; MARTÍN-GUERRERO, J. D.; MAGDALENA-BENEDITO, R.; GÓMEZ-SANCHIS, J. Regularized extreme learning machine for regression problems. **Neurocomputing**, v. 74, n. 17, p. 3716–3721, oct 2011. ISSN 09252312. Available: <http://linkinghub.elsevier.com/retrieve/pii/S092523121100378X>. Citation on page 78.

MEDINA, P.; CARRASCO, S.; ROGAN, J.; MONTES, F.; MEISEL, J. D.; LEMOINE, P.; PEÑAS, C. L.; VALDIVIA, J. A. Is a social network approach relevant to football results? **Chaos, Solitons & Fractals**, Elsevier, v. 142, p. 110369, 2021. Citation on page 62.

MICHELI, A.; TORTORELLA, D. Discrete-time dynamic graph echo state networks. **Neurocomputing**, Elsevier, v. 496, p. 85–95, 2022. Citation on page 29.

NGUYEN, G. H.; LEE, J. B.; ROSSI, R. A.; AHMED, N. K.; KOH, E.; KIM, S. Continuous-time dynamic network embeddings. In: **Proceedings of the Web Conference**. [S.l.: s.n.], 2018. p. 969–976. Citation on page 29.

ONO, J. P.; DIETRICH, C.; SILVA, C. T. Baseball timeline: Summarizing baseball plays into a static visualization. **Computer Graphics Forum**, v. 37, n. 3, p. 491–501, 2018. Citation on page 61.

OONO, K.; SUZUKI, T. Graph neural networks exponentially lose expressive power for node classification. **Proceedings of the ICLR International Conference on Learning Representation**, v. 8, 2020. Citation on page 28.

ORTIGOSSA, E. S.; GONÇALVES, T.; NONATO, L. G. Explainable artificial intelligence (xai)–from theory to methods and applications. **IEEE Access**, IEEE, 2024. Citation on page 79.

PAPPALARDO, L.; CINTIA, P.; FERRAGINA, P.; MASSUCCO, E.; PEDRESCHI, D.; GIANNOTTI, F. Playerank: data-driven performance evaluation and player ranking in soccer via a machine learning approach. **ACM Transactions on Intelligent Systems and Technology**, ACM New York, NY, USA, v. 10, n. 5, p. 1–27, 2019. Citation on page 62.

RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. " why should i trust you?" explaining the predictions of any classifier. In: **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2016. p. 1135–1144. Citation on page 79.

ROY, M. V.; ROBBERECHTS, P.; DECROOS, T.; DAVIS, J. Valuing on-the-ball actions in soccer: a critical comparison of xt and vaep. In: **Proceedings of the AAAI Workshop on Artifical Intelligence in Team Sports.** [S.l.: s.n.], 2020. Citation on page 68.

RUGGIERI, S.; ALVAREZ, J. M.; PUGNANA, A.; TURINI, F. *et al.* Can we trust fair-ai? In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2023. v. 37, n. 13, p. 15421–15430. Citation on page 79.

SABIN, R. P. Estimating player value in american football using plus–minus models. **Journal of Quantitative Analysis in Sports**, De Gruyter, v. 17, n. 4, p. 313–364, 2021. Citation on page 61.

SALINAS, K.; GONÇALVES, T.; BARELLA, V.; VIEIRA, T.; NONATO, L. G. Cityhub: A library for urban data integration. In: IEEE. **Proceedings of the SIBGRAPI Conference on Graphics, Patterns and Images.** [S.l.], 2022. v. 1, p. 43–48. Citation on page 79.

SANDRYHAILA, A.; MOURA, J. M. Discrete signal processing on graphs. **IEEE Transactions on Signal Processing**, IEEE, v. 61, n. 7, p. 1644–1656, 2013. Citation on page 35.

____. Discrete signal processing on graphs: Graph fourier transform. In: IEEE. **Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing**. [S.l.], 2013. p. 6167–6170. Citation on page 35.

____. Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure. **IEEE Signal Processing Magazine**, IEEE, v. 31, n. 5, p. 80–90, 2014. Citation on page 35.

SCARSELLI, F.; GORI, M.; TSOI, A. C.; HAGENBUCHNER, M.; MONFARDINI, G. The graph neural network model. **IEEE Transactions on Neural Networks**, IEEE, v. 20, n. 1, p. 61–80, 2008. Citation on page 27.

SEN, P.; NAMATA, G.; BILGIC, M.; GETOOR, L.; GALLIGHER, B.; ELIASSI-RAD, T. Collective classification in network data. **AI magazine**, v. 29, n. 3, p. 93–93, 2008. Citation on page 38.

SHARMA, K.; TRIVEDI, R.; SRIDHAR, R.; KUMAR, S. Temporal dynamics-aware adversarial attacks on discrete-time dynamic graph models. In: **Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2023. p. 2023–2035. Citation on page 29.

SHCHUR, O.; MUMME, M.; BOJCHEVSKI, A.; GÜNNEMANN, S. Pitfalls of graph neural network evaluation. **arXiv preprint**, 2018. Citation on page 77.

SILVA, B. L. S. da; INABA, F. K.; SALLES, E. O. T.; CIARELLI, P. M. Outlier robust extreme machine learning for multi-target regression. **Expert Systems with Applications**, Elsevier, v. 140, p. 112877, 2020. Citation on page 78.

SINGH, K. **Introducing expected threat**. 2019. Accessed: 2023-07-30. Available: <https://karun.in/blog/expected-threat.html>. Citations on pages 61, 62, 63, 68, and 72.

SINGH, N. Sport analytics: a review. **Learning**, v. 9, p. 11, 2020. Citation on page 61.

SPEARMAN, W. Beyond expected goals. In: **Proceedings of the MIT Sloan Sports Analytics Conference**. [S.l.: s.n.], 2018. v. 12, p. 1–17. Citation on page 62.

STANKOVIC, L.; MANDIC, D.; DAKOVIC, M.; BRAJOVIC, M.; SCALZO, B.; LI, S.; CONSTANTINIDES, A. G. Graph signal processing – part iii: Machine learning on graphs, from graph topology to applications. **arXiv preprint**, 2020. Citations on pages 32, 33, and 35.

SUN, H.-C.; LIN, T.-Y.; TSAI, Y.-L. Performance prediction in major league baseball by long short-term memory networks. **International Journal of Data Science and Analytics**, Springer, v. 15, n. 1, p. 93–104, 2023. Citation on page 61.

VELIČKOVIĆ, P.; CUCURULL, G.; CASANOVA, A.; ROMERO, A.; LIO, P.; BENGIO, Y. Graph attention networks. **arXiv preprint**, 2017. Citations on pages 29 and 62.

WANG, Y.; LI, Z.; FARIMANI, A. B. Graph neural networks for molecules. In: **Machine Learning in Molecular Sciences**. [S.l.]: Springer, 2023. p. 21–66. Citation on page 27.

WATKINS, D. S. **Fundamentals of Matrix Computations**. [S.l.]: John Wiley & Sons, 2004. Citation on page 89.

WU, F.; SOUZA, A.; ZHANG, T.; FIFTY, C.; YU, T.; WEINBERGER, K. Simplifying graph convolutional networks. In: PMLR. **Proceedings of the ICML International Conference on Machine Learning**. [S.l.], 2019. p. 6861–6871. Citation on page 45.

WU, X.; CHEN, Z.; WANG, W.; JADBABAIE, A. A non-asymptotic analysis of oversmoothing in graph neural networks. **arXiv preprint**, 2022. Citation on page 28.

XENOPOULOS, P.; SILVA, C. Graph neural networks to predict sports outcomes. In: IEEE. **Proceedings of the IEEE International Conference on Big Data**. [S.l.], 2021. p. 1757–1763. Citation on page 27.

XU, M. Understanding graph embedding methods and their applications. **SIAM Review**, SIAM, v. 63, n. 4, p. 825–853, 2021. Citation on page 78.

YAM, D. **Attacking Contributions: Markov Models for Football**. 2019. Accessed: 2023-08-02. Available: <https://statsbomb.com/articles/soccer/attacking-contributions-markov-models-for-football>. Citation on page 61.

YAN, S.; XIONG, Y.; LIN, D. Spatial temporal graph convolutional networks for skeleton-based action recognition. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2018. Citations on pages 28 and 39.

YANG, L.; WANG, S.; TAO, Y.; SUN, J.; LIU, X.; YU, P. S.; WANG, T. Dgrec: Graph neural network for recommendation with diversified embedding generation. In: **Proceedings of the ACM International Conference on Web Search and Data Mining**. [S.l.: s.n.], 2023. p. 661–669. Citation on page 62.

YANG, Z.; COHEN, W.; SALAKHUDINOV, R. Revisiting semi-supervised learning with graph embeddings. In: PMLR. **Proceedings of the ICML International Conference on Machine Learning**. [S.l.], 2016. p. 40–48. Citation on page 57.

YILMAZ, Ö. İ.; ÖĞÜDÜCÜ, Ş. G. Learning football player features using graph embeddings for player recommendation system. In: **Proceedings of the ACM Symposium on Applied Computing**. [S.l.: s.n.], 2022. p. 577–584. Citation on page 62.

YING, C.; CAI, T.; LUO, S.; ZHENG, S.; KE, G.; HE, D.; SHEN, Y.; LIU, T.-Y. Do transformers really perform badly for graph representation? **Proceedings of the NIPS Advances in Neural Information Processing Systems**, v. 34, p. 28877–28888, 2021. Citation on page 29.

YING, Z.; BOURGEOIS, D.; YOU, J.; ZITNIK, M.; LESKOVEC, J. Gnnexplainer: Generating explanations for graph neural networks. **Proceedings of the NIPS Advances in Neural Information Processing Systems**, v. 32, 2019. Citations on pages 68, 91, and 92.

YOU, Y.; CHEN, T.; WANG, Z.; SHEN, Y. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In: **Proceedings of the IEEE CVPR Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2020. p. 2127–2135. Citation on page 45.

ZENG, H.; PRASANNA, V. Graphact: Accelerating gcn training on cpu-fpga heterogeneous platforms. In: **Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays**. [S.l.: s.n.], 2020. p. 255–265. Citation on page 45.

ZHANG, K.; CAO, Q.; FANG, G.; XU, B.; ZOU, H.; SHEN, H.; CHENG, X. Dyted: Disentangled representation learning for discrete-time dynamic graph. In: **Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2023. p. 3309–3320. Citation on page 29.

ZHANG, K.; LUO, M. Outlier-robust extreme learning machine for regression problems. **Neurocomputing**, Elsevier, v. 151, p. 1519–1527, 2015. Citation on page 78.

ZHANG, S.; TONG, H.; XU, J.; MACIEJEWSKI, R. Graph convolutional networks: a comprehensive review. **Computational Social Networks**, SpringerOpen, v. 6, n. 1, p. 1–23, 2019. Citation on page 27.

ZHANG, Y.; GAO, S.; PEI, J.; HUANG, H. Improving social network embedding via new second-order continuous graph neural networks. In: **Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.: s.n.], 2022. p. 2515–2523. Citation on page 62.

ZHANG, Z.; CAI, Y.; GONG, W.; LIU, X.; CAI, Z. Graph convolutional extreme learning machine. In: IEEE. **Proceedings of the International Joint Conference on Neural Networks**. [S.l.], 2020. p. 1–8. Citation on page 46.

ZHANG, Z.; GAO, L.; XIANG, Y. Application of optimized bp neural network based on genetic algorithm in rugby tackle action recognition. In: IEEE. **IEEE International Conference on Artificial Intelligence and Computer Applications**. [S.l.], 2020. p. 95–99. Citation on page 61.

ZHAO, L.; SONG, Y.; ZHANG, C.; LIU, Y.; WANG, P.; LIN, T.; DENG, M.; LI, H. T-gcn: A temporal graph convolutional network for traffic prediction. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, 2019. Citations on pages 28, 39, and 77.

APPENDIX

A

PSEUDO-INVERSE (*MOORE-PENROSE GENERALIZED INVERSE*)

The pseudo-inverse of a matrix $A \in \mathbb{R}^{N\times M}$ is the generalization of its inverse operation for non-invertible and even non-square matrices (BANERJEE, 1973; WATKINS, 2004). The formal definition is as follows.

Definition A.1. (Pseudo-Inverse)

We say the pseudo-inverse of a matrix $A \in \mathbb{R}^{N\times M}$ is a matrix $B \in \mathbb{R}^{M\times N}$ such that:

i) $ABA = A$

ii) $BAB = B$

iii) $(AB)^T = AB$

iv) $(BA)^T = BA$

It is immediate that if $A \in \mathbb{R}^{N\times N}$ is invertible, then A^{-1} is a pseudo-inverse of A. Moreover, it is the only pseudo-inverse of A, because of the following Theorem.

Theorem A.1. (Existence and Uniqueness of Pseudo-Inverse)

Let A be a $N \times M$ real matrix. Then, the pseudo-inverse $A^\dagger$ of A exists and is unique.

Obs:. $A^\dagger$ can be analytically obtained as $A^\dagger = \lim_{\delta\to 0}(A^TA + \delta I)^{-1}A^T = \lim_{\delta\to 0} A^T(AA^T + \delta I)^{-1}$

Note that $A^\dagger$ can be computed by inverting a $N \times N$ or a $M \times M$ matrix, whichever is computationally less expensive. We also observe that in the particular case that A is full-column rank (which implies that $A^T A$ is invertible), then $A^\dagger = (A^T A)^{-1} A^T$. Analogously, if A is full-row rank (which implies that AA^T is invertible), then $A^\dagger = A^T (AA^T)^{-1}$.

The pseudo-inverse also plays an important role in the solution of general linear systems $AX = Y$, which may not always have an exact solution. Given fixed A and Y, consider the following optimization problem.

$$\underset{X}{\text{argmin}} \quad \|AX - Y\| \tag{A.1}$$

On the one hand, if there exists X that satisfies the linear system, then $\underset{X}{min} \quad \|AX - Y\| = 0$. On the other hand, the system could be impossible, meaning $\underset{X}{min} \quad \|AX - Y\| > 0$. In any case, we want to find which solutions X achieves minimal error $\underset{X}{min} \quad \|AX - Y\|$.

Definition A.2. (Least-Square Solution)

We say X^* is a Least-Square Solution of the optimization problem A.1 if $\|AX^* - Y\| = \underset{X}{min} \quad \|AX - Y\|$.

In the case we have multiple Least-Square Solutions X^* (e.g. an infinite solution linear system) we could be interested in selecting the one with minimal norm.

Definition A.3. (Minimum Norm Least-Square Solution)

We say X_0^* is the Minimum Norm Least-Square Solution of the optimization problem A.1 if X_0^* is a Least-Square Solution of A.1 and $\|X_0^*\| \leqslant \|X^*\|$, $\forall X^*$ Least-Square Solution of A.1.

Moreover, Theorem A.2 connects the definitions A.2, A.3 and the notion of pseudo-inverses.

Theorem A.2. Let there exist a matrix B such that BY is the Minimum Norm Least-Square Solution of A.1. Then, it is necessary and sufficient that $B = A^\dagger$.

That is, suppose we have the linear system $AX = Y$. Computing $X_0^* := A^\dagger Y$ is the (unique) solution that approximates the output Y with minimum error (being null when the system admits solution) and, among all possible solutions which best approximates Y, X_0^* also has the property of being the one with minimum norm.

APPENDIX

B

GNNEXPLAINER

GNNExplainer (YING *et al.*, 2019) is an instance-level graph explainability algorithm that is versatile enough to be used for different graph Machine Learning tasks (e.g. node classification, link prediction) and any message passing GNN model.

Let $G = (V, E)$ be a graph being V, E respectively the sets of nodes and edges. Denote N the number of graph nodes and $X \in \mathbb{R}^{N \times F}$ the matrix of node feature vectors, being F the feature space dimension. Suppose $f : V \to \mathbb{R}$ a GNN that learned a mapping from each vertex to a real number and that we want to explain the prediction $\hat{y} = f(v)$ for some particular node $v \in V$.

The key idea of the GNNExplainer approach is that the output $\hat{y}$ is fully determined by a local neighborhood of v that the GNN aggregates information throughout the message-passing mechanism. For example, if f is a 2-layered GCN, then the 2-hop neighborhood of v is a subgraph of G that contains all the required data to explain $\hat{y}$. Let $G_c(v)$ be this local neighborhood (also called the computation graph of v) and $X_c(v)$ the feature vectors of the nodes in $G_c(v)$.

Suppose we want to find a subgraph $G_s \subset G_c(v)$ and a subset of features X_s given by selecting a number of columns from $X_c(v)$ that are sufficient to explain $\hat{y}$. We discuss further the more general case in which real-valued importance masks in $[0, 1]$ are associated with the graph nodes and feature dimensions for generating the explanation. Ying *et al.* (2019) formalize this notion by the following optimization problem:

$$\max_{G_s, X_s} \quad \mathrm{MI}\big(Y, (G_s, X_s)\big) = \mathrm{H}(Y) - \mathrm{H}\big(Y | G = G_s, X = X_s\big) \tag{B.1}$$

where MI is the mutual information, Y is a random variable denoting the output and H is the

entropy.

Intuitively, the mutual information in Equation B.1 computes the change in the probability of prediction $\hat{y}$ for node v when the computation graph of this node and the feature dimensions are limited to G_s and X_s respectively. In other words, consider that removing node $u \in G_c(v)$ has little impact on the prediction $\hat{y}$ from node v. Thus, pruning u from $G_c(v)$ generates a more compact subgraph G_s for the explanation with little loss of information. In contrast, if the exclusion of feature f_i from $X_c(v)$ strongly changes $\hat{y}$, then this dimension is a good counterfactual explanation for the output of v and therefore must be kept in the explanation.

Since the term $\mathrm{H}(Y)$ in Equation B.1 is constant for different G_s, X_s, maximizing the mutual information $\mathrm{MI}\big(Y,(G_s,X_s)\big)$ is equivalent to minimizing the conditional entropy $\mathrm{H}\big(Y|G=\ G_s, X=\ X_s)\big)$. Hence, Equation B.1 is written as:

$$\min_{G_s,X_s} \mathrm{H}\big(Y|G=G_s,X=X_s)\big) = -\mathbb{E}_{Y|G_s,X_s}\big[logP_f(Y|G=G_s,X=X_s)\big] \tag{B.2}$$

The number of subgraphs and feature dimensions candidates grows exponentially, thus Ying *et al.* (2019) assess Equation B.2 by learning an importance mask for the nodes and features. More specifically, let $N_c(v)$ be the number of nodes in $G_c(v)$ the computation graph of v, $M \in \ \mathbb{R}^{N_c(v)\times F}$ be a matrix of learnable parameters and $\sigma(x) = \ {}^{1}/_{1+e^{-x}}$ the sigmoid function that maps that real numbers into the interval $(0,1)$. Hence, we assign $G_s = G_c(v)$ and $X_s = X_c(v) \odot \sigma(M)$, where $\odot$ denotes the element-wise product and $\sigma(M)$ is the matrix defined by σ applied to each entry of M. Since an arbitrary element $M_{i,j}$ of the mask is the importance of the j-th feature dimension of node i, M encodes both the relevance of the nodes (rows) and features (columns). The mask M is optimized via a gradient descent variant algorithm and the optimization also includes a regularization term in the entries of M to encourage selection only of the most important nodes and features.

Alternatively, if the user wants a subgraph and/or subset of the features for the explanation instead of a real-valued score between 0 and 1, he/she can select the most relevant nodes and/or features by hard thresholding the mask or by keeping its top-k values. Ying *et al.* (2019) also proposes to include a constraint in the optimization scheme to upper bound the number of nodes in the subgraph and/or the number of selected feature dimensions. Another alternative proposed by the authors is to overlay a learnable mask in the adjacency matrix for deriving edge-level importances. However, in the context of GNN-VSP discussed in Chapter 4, since we aim to assess players (nodes) with real-valued scores, we keep the non-thresholding approach and do not make use of masks in the graph edges.


ICMC
USP
SÃO CARLOS