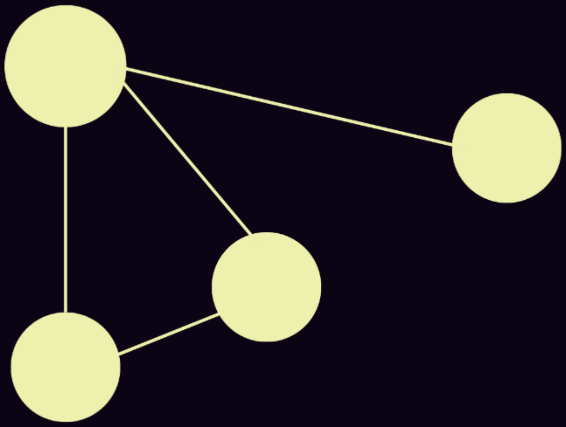
Graph Neural Networks



1 Theoretical Foundations and Core Mechanisms

Era

Graph Neural Networks: Theoretical Foundations and Core Mechanisms, A Review of the Groundbreaking 2009 Paper

Graph Neural Networks (GNNs) represent a significant advancement in machine learning's ability to process and analyze graph-structured data. This document examines the theoretical foundations established in the seminal 2009 paper by Scarselli et al., which introduced GNNs as a novel framework for handling complex network structures. The analysis focuses on three core aspects: the mathematical principles underlying GNNs, their relationship to previous approaches including Recurrent Neural Networks and Markov models, and the innovative concept of information diffusion. Through detailed examination of these foundational elements, this paper demonstrates how GNNs overcame previous limitations in processing graph-structured data, establishing a robust framework for modern machine learning applications.

Graph Neural Networks: Theoretical Foundations and Core Mechanisms, A Review of the Groundbreaking 2009 Paper

Author Analysis & Documentation

By: Hussein Mahdi Fakhry

Master's Student in Software Development

University of Babylon / College of Information Technology

Document Version: 1.0

Published: Feb 3,2025

Last Updated: Feb 4,2025

Documentation Notice:

This document represents an original analysis of the 2009 research paper "The Graph Neural Network Model" by Franco Scarselli et al. While discussing and analyzing the original paper's content, this analysis contains original insights, explanations, and interpretations that are the intellectual property of the author.

Citation Requirements:

For academic or professional reference, please cite this work as:

Fakhry, H. M. (2025). "Graph Neural Networks: Theoretical Foundations and Core Mechanisms: A Review of the Groundbreaking 2009 Paper (Part 1)". Published on Medium and GitHub.

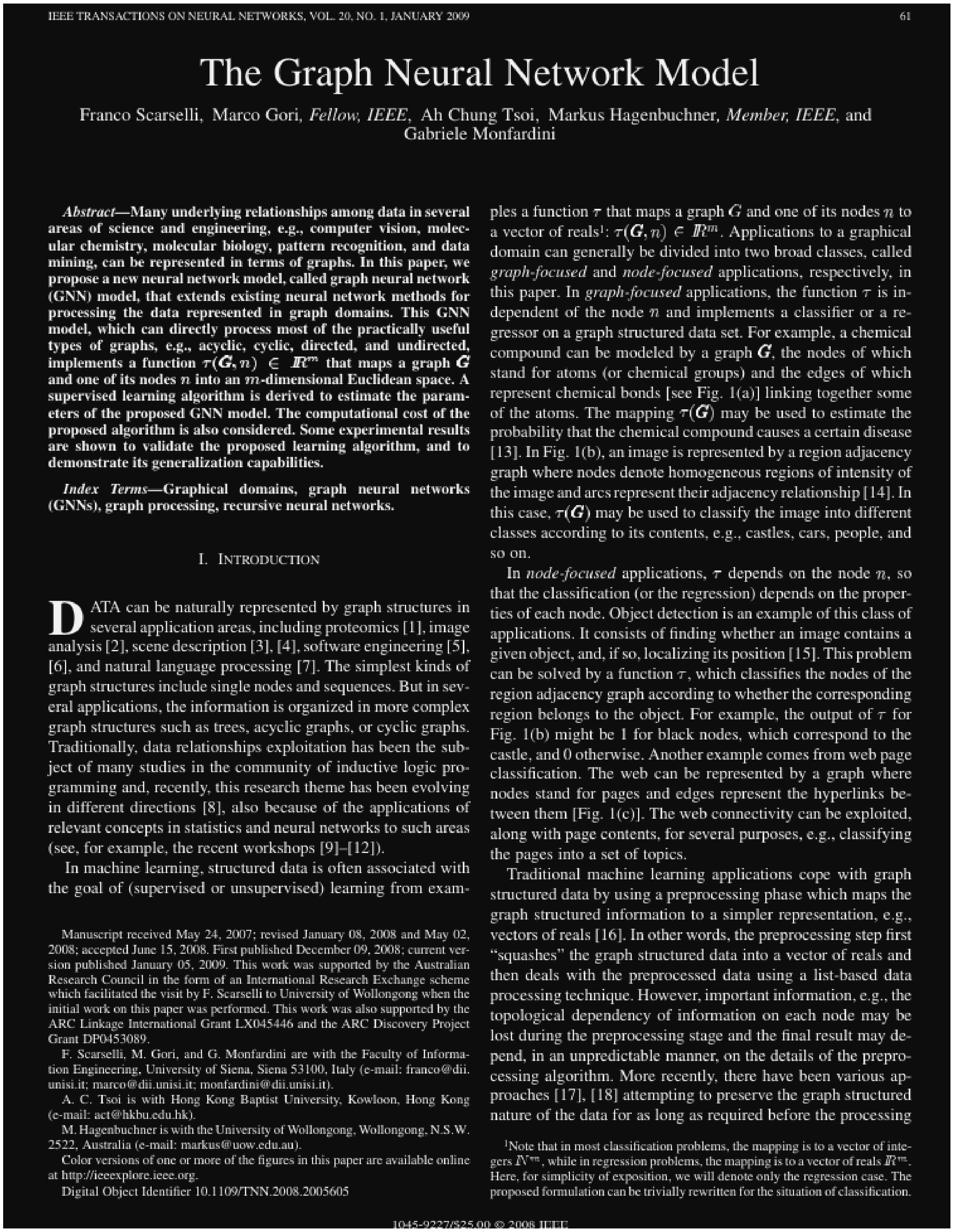
Contact Information:

GitHub: https://github.com/Hu8MA

LinkedIn: https://www.linkedin.com/in/hussein16mahdi/

Professional Email: hussein.mahdifa@gmail.com

Graph Neural Networks: Theoretical Foundations and Core Mechanisms; A Review of the Groundbreaking 2009 Paper (Part 1)



The Graph Neural Network Model

Franco Scarselli, Marco Gori, *Fellow, IEEE*, Ah Chung Tsoi, Markus Hagenbuchner, *Member, IEEE*, and Gabriele Monfardini

***Abstract*—Many underlying relationships among data in several areas of science and engineering, e.g., computer vision, molecular chemistry, molecular biology, pattern recognition, and data mining, can be represented in terms of graphs. In this paper, we propose a new neural network model, called graph neural network (GNN) model, that extends existing neural network methods for processing the data represented in graph domains. This GNN model, which can directly process most of the practically useful types of graphs, e.g., acyclic, cyclic, directed, and undirected, implements a function $\tau(\boldsymbol{G}, n) \in I\!R^m$ that maps a graph $\boldsymbol{G}$ and one of its nodes n into an m-dimensional Euclidean space. A supervised learning algorithm is derived to estimate the parameters of the proposed GNN model. The computational cost of the proposed algorithm is also considered. Some experimental results are shown to validate the proposed learning algorithm, and to demonstrate its generalization capabilities.**

***Index Terms*—Graphical domains, graph neural networks (GNNs), graph processing, recursive neural networks.**

I. Introduction

DATA can be naturally represented by graph structures in several application areas, including proteomics [1], image analysis [2], scene description [3], [4], software engineering [5], [6], and natural language processing [7]. The simplest kinds of graph structures include single nodes and sequences. But in several applications, the information is organized in more complex graph structures such as trees, acyclic graphs, or cyclic graphs. Traditionally, data relationships exploitation has been the subject of many studies in the community of inductive logic programming and, recently, this research theme has been evolving in different directions [8], also because of the applications of relevant concepts in statistics and neural networks to such areas (see, for example, the recent workshops [9]–[12]).

In machine learning, structured data is often associated with the goal of (supervised or unsupervised) learning from examples a function τ that maps a graph G and one of its nodes n to a vector of reals[1]: $\tau(\boldsymbol{G}, n) \in I\!R^m$. Applications to a graphical domain can generally be divided into two broad classes, called *graph-focused* and *node-focused* applications, respectively, in this paper. In *graph-focused* applications, the function τ is independent of the node n and implements a classifier or a regressor on a graph structured data set. For example, a chemical compound can be modeled by a graph $\boldsymbol{G}$, the nodes of which stand for atoms (or chemical groups) and the edges of which represent chemical bonds [see Fig. 1(a)] linking together some of the atoms. The mapping $\tau(\boldsymbol{G})$ may be used to estimate the probability that the chemical compound causes a certain disease [13]. In Fig. 1(b), an image is represented by a region adjacency graph where nodes denote homogeneous regions of intensity of the image and arcs represent their adjacency relationship [14]. In this case, $\tau(\boldsymbol{G})$ may be used to classify the image into different classes according to its contents, e.g., castles, cars, people, and so on.

In *node-focused* applications, τ depends on the node n, so that the classification (or the regression) depends on the properties of each node. Object detection is an example of this class of applications. It consists of finding whether an image contains a given object, and, if so, localizing its position [15]. This problem can be solved by a function τ, which classifies the nodes of the region adjacency graph according to whether the corresponding region belongs to the object. For example, the output of τ for Fig. 1(b) might be 1 for black nodes, which correspond to the castle, and 0 otherwise. Another example comes from web page classification. The web can be represented by a graph where nodes stand for pages and edges represent the hyperlinks between them [Fig. 1(c)]. The web connectivity can be exploited, along with page contents, for several purposes, e.g., classifying the pages into a set of topics.

Traditional machine learning applications cope with graph structured data by using a preprocessing phase which maps the graph structured information to a simpler representation, e.g., vectors of reals [16]. In other words, the preprocessing step first "squashes" the graph structured data into a vector of reals and then deals with the preprocessed data using a list-based data processing technique. However, important information, e.g., the topological dependency of information on each node may be lost during the preprocessing stage and the final result may depend, in an unpredictable manner, on the details of the preprocessing algorithm. More recently, there have been various approaches [17], [18] attempting to preserve the graph structured nature of the data for as long as required before the processing

Manuscript received May 24, 2007; revised January 08, 2008 and May 02, 2008; accepted June 15, 2008. First published December 09, 2008; current version published January 05, 2009. This work was supported by the Australian Research Council in the form of an International Research Exchange scheme which facilitated the visit by F. Scarselli to University of Wollongong when the initial work on this paper was performed. This work was also supported by the ARC Linkage International Grant LX045446 and the ARC Discovery Project Grant DP0453089.

F. Scarselli, M. Gori, and G. Monfardini are with the Faculty of Information Engineering, University of Siena, Siena 53100, Italy (e-mail: franco@dii.unisi.it; marco@dii.unisi.it; monfardini@dii.unisi.it).

A. C. Tsoi is with Hong Kong Baptist University, Kowloon, Hong Kong (e-mail: act@hkbu.edu.hk).

M. Hagenbuchner is with the University of Wollongong, Wollongong, N.S.W. 2522, Australia (e-mail: markus@uow.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2008.2005605

[1]Note that in most classification problems, the mapping is to a vector of integers $I\!N^m$, while in regression problems, the mapping is to a vector of reals $I\!R^m$. Here, for simplicity of exposition, we will denote only the regression case. The proposed formulation can be trivially rewritten for the situation of classification.

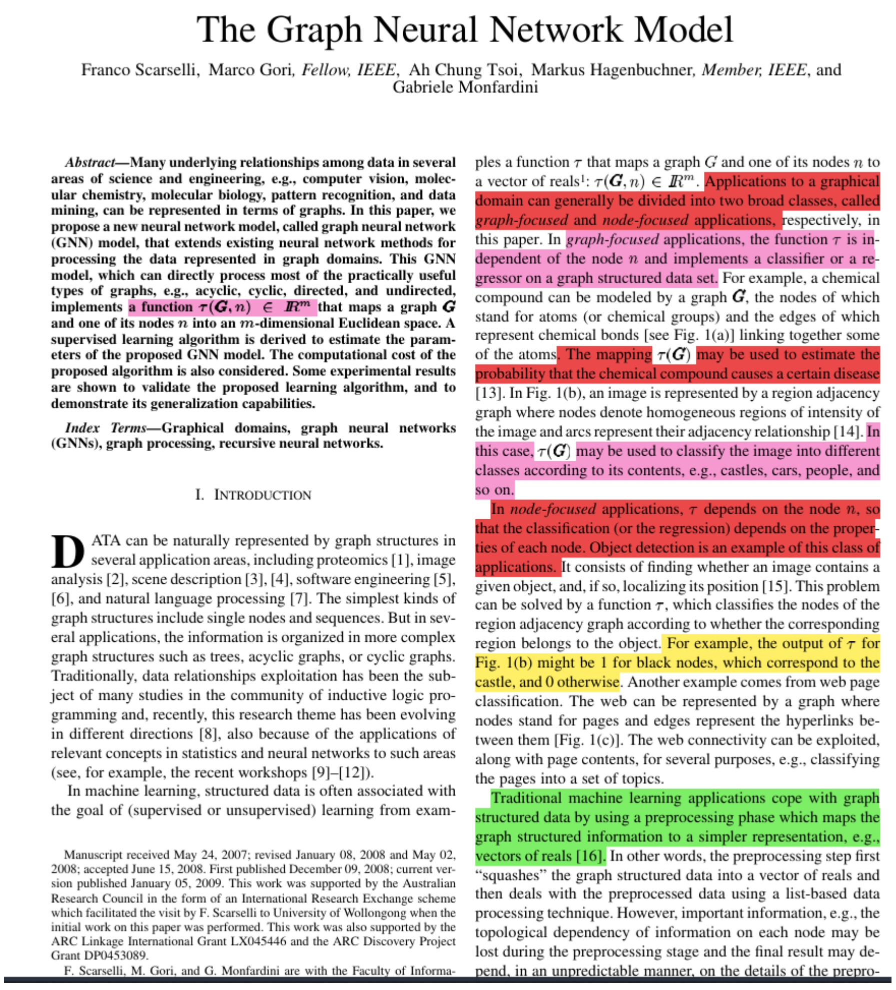
Have you ever asked how computers make sense of complex relationships in data? Back in 2008, a groundbreaking innovation called the Graph Neural Network (GNN) emerged to tackle this very challenge. When it was formally published in 2009, it revolutionized how we approach artificial intelligence, especially in the realm of deep learning.

Before GNNs came along, we faced a significant hurdle in data science. Traditional machine learning and data mining algorithms struggled to handle graph-structured data – think social networks, molecular structures, or web pages with all their interconnections. It was like trying to solve a puzzle without being able to see how the pieces fit together. These algorithms simply couldn't grasp the intricate patterns and relationships hidden within such complex data structures. That's what makes GNNs so remarkable. They didn't just offer a new solution; they opened up entirely new possibilities in artificial intelligence. Today, this field continues to grow and evolve in ways that even its creators might not have imagined.

In this two-part article, we'll take a deep dive into the groundbreaking research paper by Scarselli and his colleagues that from it started it all. We'll explore how their innovative framework transformed artificial intelligence, particularly in deep learning and data visualization. **The first part** of our journey focuses on the building blocks – the theoretical foundations that make GNNs work. We'll explore the elegant mathematical principles behind them, including something called the "universal approximation" property. I'll show you how GNNs cleverly solved problems that had stumped earlier approaches like recurrent neural networks and Markov models. Understanding these foundations helps us appreciate why GNNs represented such a leap forward in artificial intelligence. **In the second part**, we'll roll up our sleeves and look at how GNNs actually work in practice. We'll examine the learning algorithms, see how the training process unboxing, and explore different approaches to transfer functions. To wrap things up, we'll look at some fascinating real-world applications that rely on GNN architecture today.

By the end of this exploration, you'll have both a theoretical understanding and practical insights into what makes GNNs one of the most influential developments in deep learning. Whether you're a practitioner in the field or simply curious about artificial intelligence, this comprehensive look at GNNs will give you a deeper appreciation of their revolutionary impact.

Part One Let me take you through how the researchers introduced their groundbreaking work on graph structures. You know those networks made up of nodes and connections between them? They're everywhere around us, but processing them has always been a real challenge in computer science.



The Graph Neural Network Model

Franco Scarselli, Marco Gori, *Fellow, IEEE*, Ah Chung Tsoi, Markus Hagenbuchner, *Member, IEEE*, and Gabriele Monfardini

***Abstract*—Many underlying relationships among data in several areas of science and engineering, e.g., computer vision, molecular chemistry, molecular biology, pattern recognition, and data mining, can be represented in terms of graphs. In this paper, we propose a new neural network model, called graph neural network (GNN) model, that extends existing neural network methods for processing the data represented in graph domains. This GNN model, which can directly process most of the practically useful types of graphs, e.g., acyclic, cyclic, directed, and undirected, implements a function $\tau(\boldsymbol{G}, n) \in I\!R^m$ that maps a graph $\boldsymbol{G}$ and one of its nodes n into an m-dimensional Euclidean space. A supervised learning algorithm is derived to estimate the parameters of the proposed GNN model. The computational cost of the proposed algorithm is also considered. Some experimental results are shown to validate the proposed learning algorithm, and to demonstrate its generalization capabilities.**

***Index Terms*—Graphical domains, graph neural networks (GNNs), graph processing, recursive neural networks.**

I. Introduction

DATA can be naturally represented by graph structures in several application areas, including proteomics [1], image analysis [2], scene description [3], [4], software engineering [5], [6], and natural language processing [7]. The simplest kinds of graph structures include single nodes and sequences. But in several applications, the information is organized in more complex graph structures such as trees, acyclic graphs, or cyclic graphs. Traditionally, data relationships exploitation has been the subject of many studies in the community of inductive logic programming and, recently, this research theme has been evolving in different directions [8], also because of the applications of relevant concepts in statistics and neural networks to such areas (see, for example, the recent workshops [9]–[12]).

In machine learning, structured data is often associated with the goal of (supervised or unsupervised) learning from examples a function τ that maps a graph G and one of its nodes n to a vector of reals[1]: $\tau(\boldsymbol{G}, n) \in I\!R^m$. Applications to a graphical domain can generally be divided into two broad classes, called *graph-focused* and *node-focused* applications, respectively, in this paper. In *graph-focused* applications, the function τ is independent of the node n and implements a classifier or a regressor on a graph structured data set. For example, a chemical compound can be modeled by a graph $\boldsymbol{G}$, the nodes of which stand for atoms (or chemical groups) and the edges of which represent chemical bonds [see Fig. 1(a)] linking together some of the atoms. The mapping $\tau(\boldsymbol{G})$ may be used to estimate the probability that the chemical compound causes a certain disease [13]. In Fig. 1(b), an image is represented by a region adjacency graph where nodes denote homogeneous regions of intensity of the image and arcs represent their adjacency relationship [14]. In this case, $\tau(\boldsymbol{G})$ may be used to classify the image into different classes according to its contents, e.g., castles, cars, people, and so on.

In *node-focused* applications, τ depends on the node n, so that the classification (or the regression) depends on the properties of each node. Object detection is an example of this class of applications. It consists of finding whether an image contains a given object, and, if so, localizing its position [15]. This problem can be solved by a function τ, which classifies the nodes of the region adjacency graph according to whether the corresponding region belongs to the object. For example, the output of τ for Fig. 1(b) might be 1 for black nodes, which correspond to the castle, and 0 otherwise. Another example comes from web page classification. The web can be represented by a graph where nodes stand for pages and edges represent the hyperlinks between them [Fig. 1(c)]. The web connectivity can be exploited, along with page contents, for several purposes, e.g., classifying the pages into a set of topics.

Traditional machine learning applications cope with graph structured data by using a preprocessing phase which maps the graph structured information to a simpler representation, e.g., vectors of reals [16]. In other words, the preprocessing step first "squashes" the graph structured data into a vector of reals and then deals with the preprocessed data using a list-based data processing technique. However, important information, e.g., the topological dependency of information on each node may be lost during the preprocessing stage and the final result may depend, in an unpredictable manner, on the details of the prepro-

Manuscript received May 24, 2007; revised January 08, 2008 and May 02, 2008; accepted June 15, 2008. First published December 09, 2008; current version published January 05, 2009. This work was supported by the Australian Research Council in the form of an International Research Exchange scheme which facilitated the visit by F. Scarselli to University of Wollongong when the initial work on this paper was performed. This work was also supported by the ARC Linkage International Grant LX045446 and the ARC Discovery Project Grant DP0453089.

F. Scarselli, M. Gori, and G. Monfardini are with the Faculty of Informa-

The researchers made a fascinating observation: while these structures might look similar at first glance - you've got your nodes and connections - they actually behave quite differently depending on what they represent. Think about it - a social network and a molecular structure are both graphs, but they need to be handled in completely different ways.

This insight led them to divide graph processing into two main approaches:

1. Graph-Centric Analysis (graph-focused), where we look at the whole picture. Imagine analyzing a chemical compound - each atom is a node, the chemical bonds are connections, and we're trying to understand what properties the entire molecule might have.

2. Node-Centric Analysis (node-focused), where we zoom in on specific points. Take web pages, for instance - each page is a node, links between them are connections, and we might want to figure out what a particular page is about. Here, we're more interested in understanding individual pieces of the bigger picture.

This brings us to the clever solution they proposed - **the tau function (τ)**

$$\tau(\boldsymbol{G}, n) \in \mathbb{R}^m$$

- Maps a graph G and node n to numerical output
- It is a mathematical function developed by researchers
- Takes two inputs: complete graph structure and specific node
- Produces vector output of real numbers

Think of it as a translator that can take both the entire graph structure and any specific node within it, and convert them into meaningful numerical values that computers can work with. It's like having a universal interpreter that can understand both the forest and the trees. This is how the researchers throw the groundwork for tackling one of computer science's most persistent challenges - making sense of interconnected data structures in a way that's both comprehensive and practical.

Where Did GNNs Get Their Spark? A Look at Their Origins

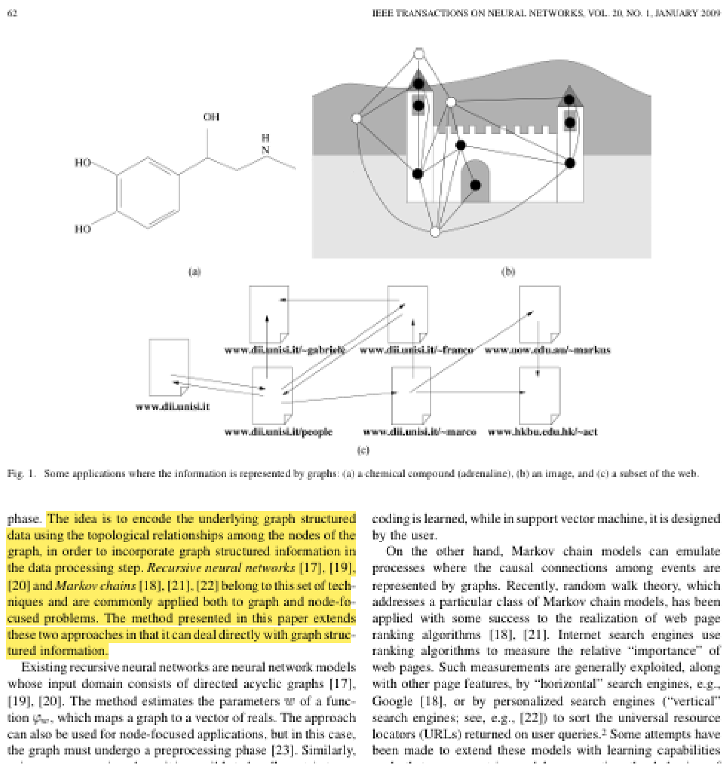

62 IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 20, NO. 1, JANUARY 2009

Fig. 1. Some applications where the information is represented by graphs: (a) a chemical compound (adrenaline), (b) an image, and (c) a subset of the web.

phase. The idea is to encode the underlying graph structured data using the topological relationships among the nodes of the graph, in order to incorporate graph structured information in the data processing step. *Recursive neural networks* [17], [19], [20] and *Markov chains* [18], [21], [22] belong to this set of techniques and are commonly applied both to graph and node-focused problems. The method presented in this paper extends these two approaches in that it can deal directly with graph structured information.

Existing recursive neural networks are neural network models whose input domain consists of directed acyclic graphs [17], [19], [20]. The method estimates the parameters w of a function φ_w, which maps a graph to a vector of reals. The approach can also be used for node-focused applications, but in this case, the graph must undergo a preprocessing phase [23]. Similarly,

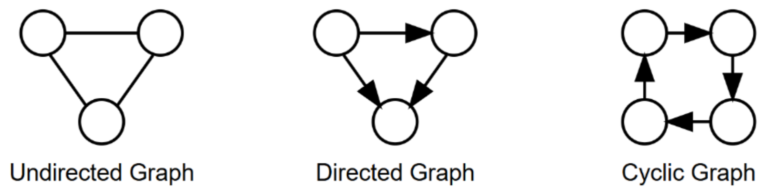
coding is learned, while in support vector machine, it is designed by the user.

On the other hand, Markov chain models can emulate processes where the causal connections among events are represented by graphs. Recently, random walk theory, which addresses a particular class of Markov chain models, has been applied with some success to the realization of web page ranking algorithms [18], [21]. Internet search engines use ranking algorithms to measure the relative "importance" of web pages. Such measurements are generally exploited, along with other page features, by "horizontal" search engines, e.g., Google [18], or by personalized search engines ("vertical" search engines; see, e.g., [22]) to sort the universal resource locators (URLs) returned on user queries.[2] Some attempts have been made to extend these models with learning capabilities

Every great innovation has its inspiration, and GNNs are no exception. On page 62 of their groundbreaking paper, the researchers reveal the fascinating journey that led to their breakthrough. Their story begins with two existing approaches that were already tackling graph processing, each with their own strengths and limitations.

First, there were Recurrent Neural Networks (RNNs). These networks were pretty good at handling straightforward graph structures - imagine following a path that only goes forward, never looping back. They excelled at basic tasks like figuring out what a chemical compound might do. But they had their struggles too. Whenever they encountered a graph with cycles - think of social networks where connections go both ways - they needed extra preprocessing steps. Plus, they weren't great at focusing on individual nodes, and they could only handle one-way relationships.

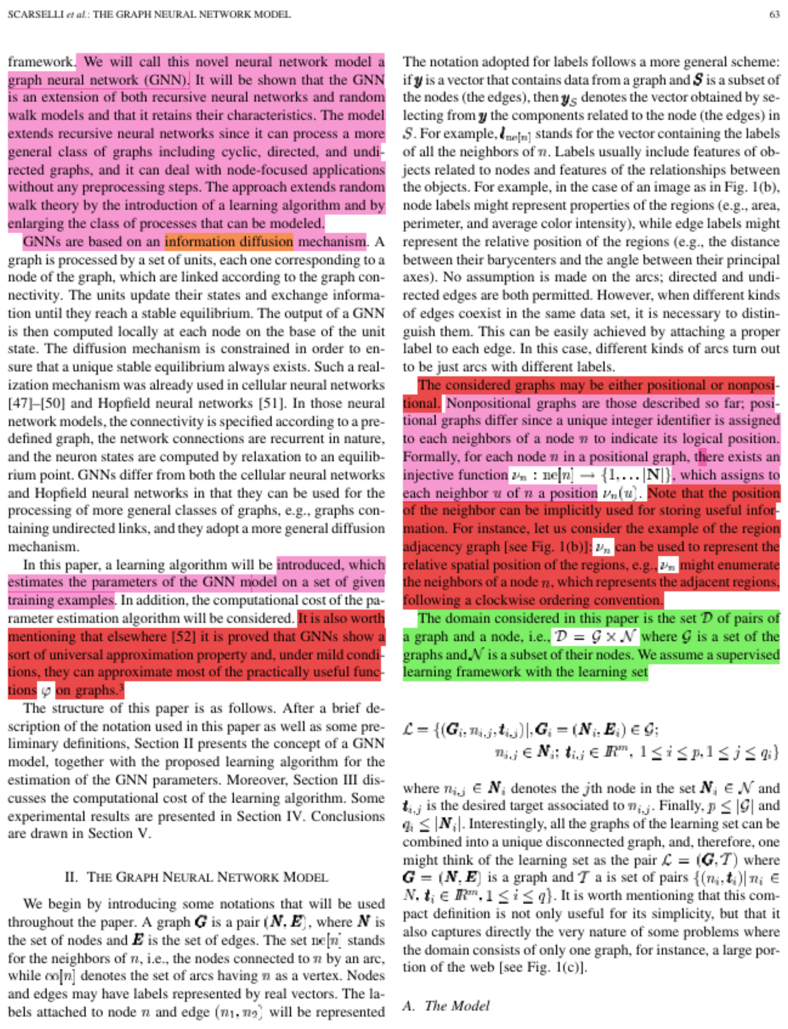
Then there were Markov models, the stars of graph visualization. These models were particularly brilliant at ranking web pages (think early Google) and figuring out which nodes in a network were most important. They could handle those tricky cyclic relationships that gave RNNs such trouble. However, they had their own limitations - they weren't great learners, mostly stuck to probability-based transitions, and struggled when patterns got too complex.



Here's where the researchers had their king moment: Why not take the best of both worlds? They realized they could combine RNNs' learning abilities with Markov models' graph-processing prowess. The result was GNNs - a new kind of neural network that could handle any type of graph structure you threw at it, whether it had cycles or not, whether connections went one way or both ways. No preprocessing required, and it could analyze both entire graphs and individual nodes with equal ease. This elegant solution didn't just overcome the limitations of its predecessors - it opened up entirely new possibilities in graph processing.

What is Information Diffusion?

is stands as the cornerstone of the researchers' work presented on page 63.



SCARSELLI *et al.*: THE GRAPH NEURAL NETWORK MODEL 63

framework. We will call this novel neural network model a graph neural network (GNN). It will be shown that the GNN is an extension of both recursive neural networks and random walk models and that it retains their characteristics. The model extends recursive neural networks since it can process a more general class of graphs including cyclic, directed, and undirected graphs, and it can deal with node-focused applications without any preprocessing steps. The approach extends random walk theory by the introduction of a learning algorithm and by enlarging the class of processes that can be modeled.

GNNs are based on an information diffusion mechanism. A graph is processed by a set of units, each one corresponding to a node of the graph, which are linked according to the graph connectivity. The units update their states and exchange information until they reach a stable equilibrium. The output of a GNN is then computed locally at each node on the base of the unit state. The diffusion mechanism is constrained in order to ensure that a unique stable equilibrium always exists. Such a realization mechanism was already used in cellular neural networks [47]–[50] and Hopfield neural networks [51]. In those neural network models, the connectivity is specified according to a predefined graph, the network connections are recurrent in nature, and the neuron states are computed by relaxation to an equilibrium point. GNNs differ from both the cellular neural networks and Hopfield neural networks in that they can be used for the processing of more general classes of graphs, e.g., graphs containing undirected links, and they adopt a more general diffusion mechanism.

In this paper, a learning algorithm will be introduced, which estimates the parameters of the GNN model on a set of given training examples. In addition, the computational cost of the parameter estimation algorithm will be considered. It is also worth mentioning that elsewhere [52] it is proved that GNNs show a sort of universal approximation property and, under mild conditions, they can approximate most of the practically useful functions φ on graphs.[3]

The structure of this paper is as follows. After a brief description of the notation used in this paper as well as some preliminary definitions, Section II presents the concept of a GNN model, together with the proposed learning algorithm for the estimation of the GNN parameters. Moreover, Section III discusses the computational cost of the learning algorithm. Some experimental results are presented in Section IV. Conclusions are drawn in Section V.

II. THE GRAPH NEURAL NETWORK MODEL

We begin by introducing some notations that will be used throughout the paper. A graph $\boldsymbol{G}$ is a pair $(\boldsymbol{N}, \boldsymbol{E})$, where $\boldsymbol{N}$ is the set of nodes and $\boldsymbol{E}$ is the set of edges. The set $\mathrm{ne}[n]$ stands for the neighbors of n, i.e., the nodes connected to n by an arc, while $\mathrm{co}[n]$ denotes the set of arcs having n as a vertex. Nodes and edges may have labels represented by real vectors. The labels attached to node n and edge (n_1, n_2) will be represented The notation adopted for labels follows a more general scheme: if $\boldsymbol{y}$ is a vector that contains data from a graph and $\boldsymbol{S}$ is a subset of the nodes (the edges), then $\boldsymbol{y}_S$ denotes the vector obtained by selecting from $\boldsymbol{y}$ the components related to the node (the edges) in S. For example, $\boldsymbol{l}_{\mathrm{ne}[n]}$ stands for the vector containing the labels of all the neighbors of n. Labels usually include features of objects related to nodes and features of the relationships between the objects. For example, in the case of an image as in Fig. 1(b), node labels might represent properties of the regions (e.g., area, perimeter, and average color intensity), while edge labels might represent the relative position of the regions (e.g., the distance between their barycenters and the angle between their principal axes). No assumption is made on the arcs; directed and undirected edges are both permitted. However, when different kinds of edges coexist in the same data set, it is necessary to distinguish them. This can be easily achieved by attaching a proper label to each edge. In this case, different kinds of arcs turn out to be just arcs with different labels.

The considered graphs may be either positional or nonpositional. Nonpositional graphs are those described so far; positional graphs differ since a unique integer identifier is assigned to each neighbors of a node n to indicate its logical position. Formally, for each node n in a positional graph, there exists an injective function $\nu_n : \mathrm{ne}[n] \rightarrow \{1, \ldots |\mathrm{N}|\}$, which assigns to each neighbor u of n a position $\nu_n(u)$. Note that the position of the neighbor can be implicitly used for storing useful information. For instance, let us consider the example of the region adjacency graph [see Fig. 1(b)]: ν_n can be used to represent the relative spatial position of the regions, e.g., ν_n might enumerate the neighbors of a node n, which represents the adjacent regions, following a clockwise ordering convention.

The domain considered in this paper is the set $\mathcal{D}$ of pairs of a graph and a node, i.e., $\mathcal{D} = \mathcal{G} \times \mathcal{N}$ where $\mathcal{G}$ is a set of the graphs and $\mathcal{N}$ is a subset of their nodes. We assume a supervised learning framework with the learning set

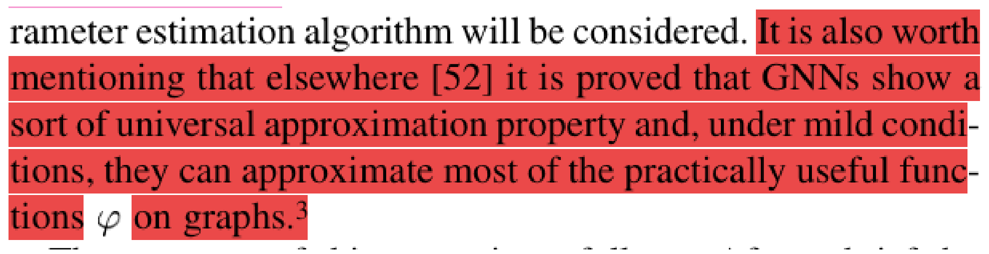
$$\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{i,j}) | , \boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i) \in \mathcal{G}; \quad n_{i,j} \in \boldsymbol{N}_i; \ \boldsymbol{t}_{i,j} \in \mathbb{R}^m, \ 1 \le i \le p, 1 \le j \le q_i\}$$

where $n_{i,j} \in \boldsymbol{N}_i$ denotes the jth node in the set $\boldsymbol{N}_i \in \mathcal{N}$ and $\boldsymbol{t}_{i,j}$ is the desired target associated to $n_{i,j}$. Finally, $p \le |\mathcal{G}|$ and $q_i \le |\boldsymbol{N}_i|$. Interestingly, all the graphs of the learning set can be combined into a unique disconnected graph, and, therefore, one might think of the learning set as the pair $\mathcal{L} = (\boldsymbol{G}, \mathcal{T})$ where $\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{E})$ is a graph and $\mathcal{T}$ a is set of pairs $\{(n_i, \boldsymbol{t}_i) | n_i \in N, \boldsymbol{t}_i \in \mathbb{R}^m, 1 \le i \le q\}$. It is worth mentioning that this compact definition is not only useful for its simplicity, but that it also captures directly the very nature of some problems where the domain consists of only one graph, for instance, a large portion of the web [see Fig. 1(c)].

A. *The Model*

This fundamental concept introduces a method of communication between nodes, where neighboring nodes share features and properties until the graph structure reaches a stable state. The researchers termed this process Information Diffusion, which later evolved into what we now know as "message passing." This innovative approach establishes the foundation for how the network operates, enabling nodes to both share and receive information from their neighbors, helping them understand their nature, location, and importance within the broader structure. Significantly, this approach laid the groundwork for the theory of universal approximation.

In the same section, the researchers present a crucial claim about the network's capabilities. They state:


rameter estimation algorithm will be considered. It is also worth mentioning that elsewhere [52] it is proved that GNNs show a sort of universal approximation property and, under mild conditions, they can approximate most of the practically useful functions φ on graphs.[3]

This leads us to examine the concept of universal approximation in depth. At its core, Graph Neural Networks (GNNs) possess the ability to approximate any continuous function on graphs. To understand this property fully, we must first grasp what a continuous function means in this context.

A continuous function is one where small changes in the input result in correspondingly small changes in the output, without abrupt jumps or interruptions. In graph structures, this continuity show in how information flows between nodes, where similar nodes (in terms of features or connectivity) should produce similar outputs. Such as social network: users with similar friend groups tend to share similar interests.

This universal approximation capability means GNNs can learn and represent complex relationships in graph-structured data, provided these relationships can be described by a continuous function. However, the researchers identified several key requirements for this approximation to work effectively:

- Sufficient network capacity
- Continuous target function
- Relevant information in graph structure
- Effective parameter optimization

The approximation process operates through two essential mechanisms:

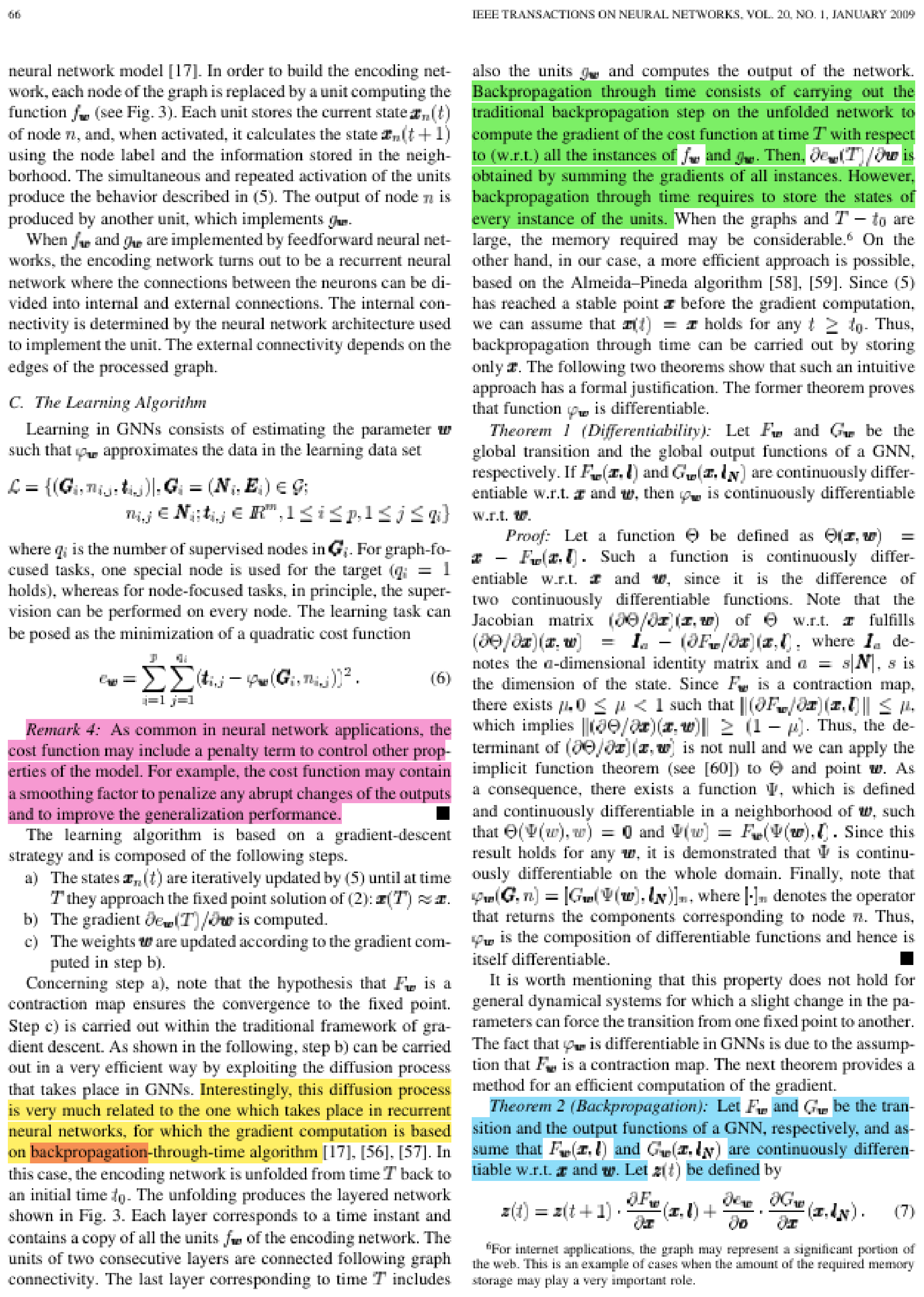
- State transition function (fw) for graph structure
- Output function (gw) for final transformation

This framework's versatility becomes particularly important when we consider the diverse nature of graph structures that GNNs must process. The researchers recognized that different applications require different approaches to handling graph relationships. To address this, they introduced a fundamental distinction in how graphs can be structured and processed.

The first category, non-positional graphs, represents the simplest form of graph relationships. In these structures, the only relevant information is whether a connection exists between nodes. Social networks exemplify this type, where the fundamental concern is whether two users are connected, regardless of any ordering or hierarchical relationship between them. The connections simply indicate a binary state - either nodes are connected or they're not. **In contrast, positional graphs** introduce an additional layer of complexity by incorporating specific ordering in node relationships. These graphs use a position function fn[i] that assigns each neighbor a specific position within the range {1,...,|N|}. This ordering proves crucial in applications like molecular chemistry, where the precise arrangement of atoms (nodes) fundamentally affects the properties of the entire molecule. The position information preserves essential structural characteristics that would be lost in a purely connection-based representation.

This distinction between positional and non-positional graphs reflects a deeper understanding of how different real-world systems organize and utilize their relational information, allowing GNNs to adapt their processing approach based on the specific requirements of each application domain.

Having explored the core mechanisms of information diffusion and universal approximation, the researchers turned their attention to a crucial aspect of GNNs - their learning process. On page 66,



neural network model [17]. In order to build the encoding network, each node of the graph is replaced by a unit computing the function $f_{\boldsymbol{w}}$ (see Fig. 3). Each unit stores the current state $\boldsymbol{x}_n(t)$ of node n, and, when activated, it calculates the state $\boldsymbol{x}_n(t+1)$ using the node label and the information stored in the neighborhood. The simultaneous and repeated activation of the units produce the behavior described in (5). The output of node n is produced by another unit, which implements $g_{\boldsymbol{w}}$.

When $f_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$ are implemented by feedforward neural networks, the encoding network turns out to be a recurrent neural network where the connections between the neurons can be divided into internal and external connections. The internal connectivity is determined by the neural network architecture used to implement the unit. The external connectivity depends on the edges of the processed graph.

C. The Learning Algorithm

Learning in GNNs consists of estimating the parameter $\boldsymbol{w}$ such that $\varphi_{\boldsymbol{w}}$ approximates the data in the learning data set

$$\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{i,j}) | \, \boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i) \in \mathcal{G}; \; n_{i,j} \in \boldsymbol{N}_i; \boldsymbol{t}_{i,j} \in \mathbb{R}^m, 1 \le i \le p, 1 \le j \le q_i\}$$

where q_i is the number of supervised nodes in $\boldsymbol{G}_i$. For graph-focused tasks, one special node is used for the target ($q_i = 1$ holds), whereas for node-focused tasks, in principle, the supervision can be performed on every node. The learning task can be posed as the minimization of a quadratic cost function

$$e_{\boldsymbol{w}} = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (\boldsymbol{t}_{i,j} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))^2 . \tag{6}$$

Remark 4: As common in neural network applications, the cost function may include a penalty term to control other properties of the model. For example, the cost function may contain a smoothing factor to penalize any abrupt changes of the outputs and to improve the generalization performance. ■

The learning algorithm is based on a gradient-descent strategy and is composed of the following steps.

a) The states $\boldsymbol{x}_n(t)$ are iteratively updated by (5) until at time T they approach the fixed point solution of (2): $\boldsymbol{x}(T) \approx \boldsymbol{x}$.
b) The gradient $\partial e_{\boldsymbol{w}}(T)/\partial \boldsymbol{w}$ is computed.
c) The weights $\boldsymbol{w}$ are updated according to the gradient computed in step b).

Concerning step a), note that the hypothesis that $F_{\boldsymbol{w}}$ is a contraction map ensures the convergence to the fixed point. Step c) is carried out within the traditional framework of gradient descent. As shown in the following, step b) can be carried out in a very efficient way by exploiting the diffusion process that takes place in GNNs. Interestingly, this diffusion process is very much related to the one which takes place in recurrent neural networks, for which the gradient computation is based on backpropagation-through-time algorithm [17], [56], [57]. In this case, the encoding network is unfolded from time T back to an initial time t_0. The unfolding produces the layered network shown in Fig. 3. Each layer corresponds to a time instant and contains a copy of all the units $f_{\boldsymbol{w}}$ of the encoding network. The units of two consecutive layers are connected following graph connectivity. The last layer corresponding to time T includes also the units $g_{\boldsymbol{w}}$ and computes the output of the network. Backpropagation through time consists of carrying out the traditional backpropagation step on the unfolded network to compute the gradient of the cost function at time T with respect to (w.r.t.) all the instances of $f_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$. Then, $\partial e_{\boldsymbol{w}}(T)/\partial \boldsymbol{w}$ is obtained by summing the gradients of all instances. However, backpropagation through time requires to store the states of every instance of the units. When the graphs and $T - t_0$ are large, the memory required may be considerable.[6] On the other hand, in our case, a more efficient approach is possible, based on the Almeida–Pineda algorithm [58], [59]. Since (5) has reached a stable point $\boldsymbol{x}$ before the gradient computation, we can assume that $\boldsymbol{x}(t) = \boldsymbol{x}$ holds for any $t \ge t_0$. Thus, backpropagation through time can be carried out by storing only $\boldsymbol{x}$. The following two theorems show that such an intuitive approach has a formal justification. The former theorem proves that function $\varphi_{\boldsymbol{w}}$ is differentiable.

Theorem 1 (Differentiability): Let $F_{\boldsymbol{w}}$ and $G_{\boldsymbol{w}}$ be the global transition and the global output functions of a GNN, respectively. If $F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$ and $G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$ are continuously differentiable w.r.t. $\boldsymbol{x}$ and $\boldsymbol{w}$, then $\varphi_{\boldsymbol{w}}$ is continuously differentiable w.r.t. $\boldsymbol{w}$.

Proof: Let a function Θ be defined as $\Theta(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x} - F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$. Such a function is continuously differentiable w.r.t. $\boldsymbol{x}$ and $\boldsymbol{w}$, since it is the difference of two continuously differentiable functions. Note that the Jacobian matrix $(\partial\Theta/\partial\boldsymbol{x})(\boldsymbol{x}, \boldsymbol{w})$ of Θ w.r.t. $\boldsymbol{x}$ fulfills $(\partial\Theta/\partial\boldsymbol{x})(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{I}_a - (\partial F_{\boldsymbol{w}}/\partial\boldsymbol{x})(\boldsymbol{x}, \boldsymbol{l})$, where $\boldsymbol{I}_a$ denotes the a-dimensional identity matrix and $a = s|\boldsymbol{N}|$, s is the dimension of the state. Since $F_{\boldsymbol{w}}$ is a contraction map, there exists $\mu, 0 \le \mu < 1$ such that $\|(\partial F_{\boldsymbol{w}}/\partial\boldsymbol{x})(\boldsymbol{x}, \boldsymbol{l})\| \le \mu$, which implies $\|(\partial\Theta/\partial\boldsymbol{x})(\boldsymbol{x}, \boldsymbol{w})\| \ge (1 - \mu)$. Thus, the determinant of $(\partial\Theta/\partial\boldsymbol{x})(\boldsymbol{x}, \boldsymbol{w})$ is not null and we can apply the implicit function theorem (see [60]) to Θ and point $\boldsymbol{w}$. As a consequence, there exists a function Ψ, which is defined and continuously differentiable in a neighborhood of $\boldsymbol{w}$, such that $\Theta(\Psi(w), w) = \boldsymbol{0}$ and $\Psi(w) = F_{\boldsymbol{w}}(\Psi(\boldsymbol{w}), \boldsymbol{l})$. Since this result holds for any $\boldsymbol{w}$, it is demonstrated that Ψ is continuously differentiable on the whole domain. Finally, note that $\varphi_{\boldsymbol{w}}(\boldsymbol{G}, n) = [G_{\boldsymbol{w}}(\Psi(\boldsymbol{w}), \boldsymbol{l}_{\boldsymbol{N}})]_n$, where $[\cdot]_n$ denotes the operator that returns the components corresponding to node n. Thus, $\varphi_{\boldsymbol{w}}$ is the composition of differentiable functions and hence is itself differentiable. ■

It is worth mentioning that this property does not hold for general dynamical systems for which a slight change in the parameters can force the transition from one fixed point to another. The fact that $\varphi_{\boldsymbol{w}}$ is differentiable in GNNs is due to the assumption that $F_{\boldsymbol{w}}$ is a contraction map. The next theorem provides a method for an efficient computation of the gradient.

Theorem 2 (Backpropagation): Let $F_{\boldsymbol{w}}$ and $G_{\boldsymbol{w}}$ be the transition and the output functions of a GNN, respectively, and assume that $F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$ and $G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}})$ are continuously differentiable w.r.t. $\boldsymbol{x}$ and $\boldsymbol{w}$. Let $\boldsymbol{z}(t)$ be defined by

$$\boldsymbol{z}(t) = \boldsymbol{z}(t+1) \cdot \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l}) + \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l}_{\boldsymbol{N}}) . \tag{7}$$

[6]For internet applications, the graph may represent a significant portion of the web. This is an example of cases when the amount of the required memory storage may play a very important role.

they presented a detailed comparison between traditional neural networks and GNNs, focusing specifically on how these architectures handle backpropagation. This comparison proved essential in understanding why GNNs required a different approach to learning from their traditional counterparts.

Aspect	Traditional Neural Networks	Graph Neural Networks
Layer Structure	Fixed network layers with static depth	Dynamic layers representing time steps
Connections	Predefined layer-to-layer connections	Determined by graph topology
Memory Requirements	Depends on network depth	Depends on graph size and processing time T
Gradient Flow	Direct layer-to-layer propagation	Follows graph edges after reaching stable state
Convergence	Standard backpropagation	Requires stable state before backpropagation

This understanding comes specifically from the discussion in Section II.C of the paper, where they describe the learning algorithm and its theoretical foundations.

Throughout their groundbreaking paper, particularly in Section II, the researchers demonstrated how GNNs represent a significant leap forward in processing graph-structured data. As they concluded their theoretical framework, they highlighted both the practical advantages and real-world applications that make GNNs particularly valuable.

The framework's strengths emerge from its unified approach to graph processing. As follow:

- Direct processing of all practical graph types (cyclic, directed, undirected)
- Unified framework for both global and local analysis
- Built-in learning capabilities without preprocessing
- Preservation of graph structural information

These theoretical advantages translate into practical applications across diverse fields. Into:

- Chemical compound analysis
- Web page classification and ranking
- Pattern recognition in structured data
- Social network analysis

This combination of theoretical elegance and practical applicability demonstrates why GNNs represent such a significant advancement in machine learning approaches to graph-structured data.

Conclusion, In this groundbreaking exploration of Graph Neural Networks (GNNs), we delved into the theoretical foundations that have revolutionized artificial intelligence's ability to process complex graph-structured data. By was examining the seminal research paper by Scarselli and his colleagues, we uncover the elegant mathematical principles and innovative approaches that set GNNs apart from their predecessors. From the tau function's role as a universal interpreter to the significance of information diffusion and universal approximation, we lay the groundwork for understanding GNNs' transformative potential. As we conclude this first installment, We find ourselves at the next installment: how to translate these theoretical concepts into practical applications? The answer awaits us in Part Two, where we will explore the intricacies of GNN learning algorithms, training processes, and real-world implementations that promise to reshape our understanding of artificial intelligence's capabilities.