Bocconi

Bachelor of Science in Economics, Management and Computer Science

Graph Neural Networks: From Foundations to Temporal Applications

Bachelor of Science thesis by:
Marco Lomele

Advisor:
Prof. Luca Saglietti

Academic Year 2023-2024

Bocconi
Università Commerciale
Luigi Bocconi

Table of Contents

1 Introduction **3**

2 Graph Representation Learning **5**

2.1 Representation Learning 5

2.2 Graphs 6

2.3 Graph Learning 8

3 Graph Neural Networks **12**

3.1 Background 12

3.2 General Framework 13

3.3 Graph Convolutional Networks 14

3.4 Graph Attention Networks 17

3.5 Message Passing Neural Networks 21

3.6 Depth Concerns 22

4 Dynamic Graphs **26**

4.1 Temporal Dimension 26

4.2 Taxonomy 27

4.3 Dynamic Graph Learning 29

5 Dynamic Graph Neural Networks **31**

5.1 Modelling Approach 31

5.2 Sequence Models 31

5.3 Architectures 34

5.4 Real World Applications 38

6 Conclusions **44**

Table of Notations and Abbreviations

A concise description of the main notations and abbreviations used in this thesis.

Notation	Meaning
G	Graph
$\mathcal{V}$	Node set
N	Number of nodes
$\mathcal{E}$	Edge set
$\mathcal{N}(i)$	Neighbourhood of node i
A	Adjacency matrix
X	Matrix for the attributes of the nodes
E	Tensor for the attributes of the edges
ℓ	Index for layers
$h_i^{(\ell)}$	Vector representation or embedding of node i at layer ℓ
F	Dimension of the embedding space
$W^{(\ell)}$	Matrix of learnable parameters at the ℓ-th layer
b	Bias term, scalar or vector
D	Diagonal degree matrix
t	Index for time
$\mathcal{G}$	Dynamic Graph
G_t	Graph at time t
$\mathcal{O}_t$	Sequence of observations of events up to time t
$z_{i,t}$	Temporal embedding or representation for node i at time t
σ	Rectified Linear Unit (ReLU) function
ϑ	Sigmoid function
$\mathcal{X}$	Input sequence vectors
$T_{\mathcal{X}}$	Number of elements in sequence $\mathcal{X}$

Abbreviation	Meaning
GNN	Graph Neural Networks
DGNN	Dynamic Graph Neural Network
CNN	Convolutional Neural Network
GCN	Graph Convolutional Network
GAT	Graph Attention Network
MPNN	Message Passing Neural Network
DTDG	Discrete Time Dynamic Graph
CTDG	Continuous Time Dynamic Graph
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory
GRU	Gated Recurrent Unit
TGN	Temporal Graph Network
A3T-GCN	Attention Temporal Graph Convolutional Network
MLP	Multi-Layer Perceptron

Introduction

Network systems are ubiquitous in the real world, with applications spanning business, science, and beyond. At the heart of biological systems, communication grids, e-commerce websites, and social media platforms lies a common structure: a network of interacting entities.

To model networks, we use graphs, a high-level mathematical structure that represents entities with nodes and relationships with edges. Graphs are versatile and possess many components that makes them easy to adapt to any network. However, as the size of the network increases, it becomes increasingly inefficient to use the original formulation with nodes and edges.

Graph representation learning deals with converting large-scale graphs into numerical vectors in a latent space known as embeddings. These vectors comprise abstract features that capture the original information contained in the graph. One advantage of this formulation is that it enables Machine Learning tasks on graphs. Over the last decade, Graph Neural Networks (GNNs) have emerged as the leading paradigm for building effective embeddings that enable powerful inference tasks such as node classification or link prediction.

Real-world networks are inherently dynamic, evolving in structure and properties over time. This added dimension is modeled with dynamic graphs, which allow for variable structures and attributes. This implies a new goal for dynamic graph representation learning: to produce node representation embeddings that capture spatial, structural, and temporal dependencies.

How static GNNs should be adapted to dynamic contexts remains an open question. Some distinct architectures are emerging, but the field is far from converging to some general framework, since all the proposed implementations are designed to excel on specific inference task. Concepts and ideas are scattered under different names and notations, information is not easily accessible, and the reader needs significant technical knowledge to fully understand it.

The purpose of the thesis is to contribute a text to the Graph Representation field that consolidates the foundations of Graph Neural Networks and introduces their dynamic extension through consistent notation and straightforward explanations. Therefore, this work is for those with a keen interest in machine learning applied to network systems.

We will begin with representation learning, which contextualises the rest of the thesis. Then, we will dive into GNNs, the general framework, and the three main variations. Next, we will introduce the concept of dynamic graphs and the new inference tasks that they enable. Finally, we will explore Dynamic Graph Neural Networks, review sequence models, explore emerging architectures, and apply our understanding on traffic prediction and agriculture trade forecasting.

The Commodity Trade Problem

I was introduced to GNNs through a project during my professional experience at a Dutch company that offers intelligence tools for commodity markets. The project involved researching a model to address a key challenge in trade markets: the lack of real-time insights.

Since data is sourced solely from government authorities, there is a significant delay in trade information. On average, it takes three months for all countries to report their trade volumes. If we are in June, the latest and complete market data available is from February. This lag is an inefficiency of the market and severely hinders the planning capabilities of buyers, traders, and sellers of commodities.

The goal is to eliminate the reporting delay. We aim to build a model that can accurately predict global trade volumes for the past three months and potentially forecast future values. Additionally, we want to leverage a set variables that are related to trade, such as prices and stocks, and which are updated with a smaller lag.

The commodity trade problem will guide us throughout this thesis. Topic after topic, we will return to this problem to contextualise the discussed theory. In other words, this problem is our guiding North Star, and thus we will annotate all the paragraphs that are related to it with the symbol (★).

Graph Representation Learning

2.1 Representation Learning

2.1.1 Motivation

The effectiveness of machine learning methods relies on both the design of the algorithms and the representation, or feature set, of the data. Different representations incorporate the different explanatory factors that cause variation in the data. For example, we can represent a meteorological event by recording various measures around it like time, location, temperature, and air pressure.

In traditional machine learning, representations are constructed through feature engineering, transforming raw data into more expressive features towards the target variable. In our previous example, a simple feature engineering operation could be computing the ratio between temperature and pressure rather than using the two measures separately.

However, this process is labour-intensive, relies on domain expertise, and is subject to selection bias. Over the last decade, research has focused on developing algorithms that depend less on feature engineering. The goal is to design algorithms that learn features autonomously in latent spaces.

2.1.2 Definition

Y. Bengio et al. (2013) define representation learning as the process of building abstract representations of the data that facilitate the extraction of useful information when building predictors. A strong representation captures the distribution of the explanatory factors from the input data without any noise. [4]

For example, learning the representation of an image means converting its pixel data into a set of features that capture the information contained in that image. We will revisit this concept more in depth for graphs in the coming sections.

2.2 Graphs

2.2.1 Introduction

Representation learning is well-established for popular data types like images, text, and sound because of high availability. However, there is another type of data that is ubiquitous in the world but has received less attention: network data.

Network systems emerge in contexts where many entities are linked together by some relationship. Examples can be found in digital contexts, such as connections on social media or communication grids, and in physical contexts, like protein-to-protein interactions or road systems.

Network data is modelled with graphs. Unlike the standard tabular format that list entities individually, graphs emphasise the relationships among them, preserving the valuable information that is stored in the connections of the network.

2.2.2 Definitions

Graph. A graph is a mathematical structure used to model complex systems with interacting objects. Formally, it is tuple $G = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{v_1, ..., v_N\}$ the set of N nodes and $\mathcal{E} \subseteq \{(v_i, v_j) \mid \forall v_i, v_j \in \mathcal{V}\}$ the set of edges. An edge from node $i \in \mathcal{V}$ to node $j \in \mathcal{V}$ is denoted with $(i, j) \in \mathcal{E}$. Each node in the graph represents an object of the network, and the edges indicate their interactions. [16]

Neighbourhood. The set of nodes that share an edge with node i form the 1-hop neighbourhood of i, denoted as $\mathcal{N}(i) = \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\}$; the set of nodes within two connections form the 2-hop neighbourhood, and so on.

Adjacency Matrix. The adjacency matrix $A \in \mathbb{R}^{N \times N}$ efficiently encodes the structure of the graph. This square matrix uses node indices to label its rows and columns. For each pair of nodes (i, j), $A[i, j] = 1$ if the two nodes share an edge, and $A[i, j] = 0$ otherwise. If an edge starts and ends at the same node, known as a self-loop, then $A[i, i] = 1$. The adjacency matrix enables spectral decomposition of the graph, a powerful concept that we will see again in the next chapter.

Directed Graph. Edges can take different meanings based on the relationships they model. When the direction is meaningful, such that $(i, j) \neq (j, i)$ for $(i, j), (j, i) \in \mathcal{E}$, the graph is said to be directed, and the adjacency matrix is not necessarily symmetric.

If (i, j) is a directed edge, i is the source and j is the destination/target node. Conversely, if direction is meaningless, such that $(i, j) = (j, i)$, then the graph is undirected, and adjacency matrix is symmetric.

Side Information. Extra information about the nodes and the edges can be incorporated into the graph. This data is tied to tangible qualities, allowing the model to capture spatial dependencies, id est (i.e.) the influence that attributes have on each other. With side information is provided the definition of the graph exteneds to $G = \{\mathcal{V}, \mathcal{E}, X, E\}$.

Node information comes as a real-valued matrix $X \in \mathbb{R}^{N \times d}$ and is interpreted as either a label (for $d = 1$) or a set of d features/attributes (for $d > 1$) per node that characterise the entities.

Edge information comes as a tensor $E \in R^{N \times N \times p}$ and is interpreted as either a weight (for $p = 1$) or a set of p features/attributes (for $p > 1$) per edge that characterise the relationships among entities. For $p = 1$, the tensor becomes the adjacency matrix with real-valued entries, and the graph is said to be weighted. [13]

Multirelational Graphs. Modelling complex networks may require to categorise nodes and edges into different types, resulting in multi-relational graphs. If a graph has different node types, which may be indicated explicitly through labels or implicitly by construction, it is called heterogeneous. Alternatively, if a graph has different edge types, where each type indicates a specific kind of relationship, it is called multiplex.

($\bigstar$) We can use a directed and weighted graph to model the commodity trade problem. Nodes represent countries and directed edges indicate trade. Moreover, we append country-specific features on the nodes, such as production, and pair-specific features on the edges, such as the distance and the trade volume, our target variable. Finally, we distinguish between exporting countries, the source nodes, and importing countries, the destination nodes. Figure 2.1 gives an idea of how the graph might look like.

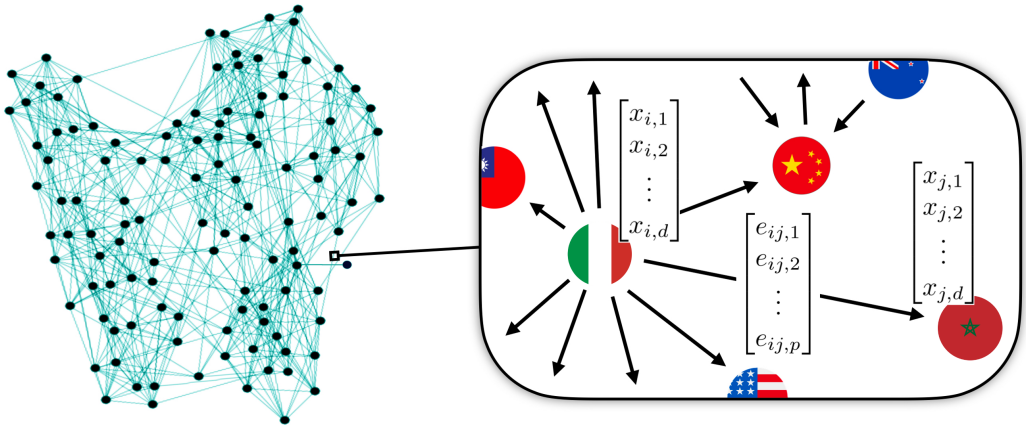

Figure 2.1: Graph modelling the trade network. Node i and j are Italy and Morocco

2.3 Graph Learning

2.3.1 Motivation

As the size of the graph increases, its representation presents several challenges. First, because node relationships are encoded with the edges, graph analysis algorithms require combinatorial amounts of computation steps, and thus scale poorly.

Coupling nodes in pairs also limits the use of parallel and distributed algorithms, since no two parts of a graph are truly independent. Finally, graphs are incompatible with machine learning methods, which require numerical vector data. To overcome these limitations, we need to represent graphs in continuous space.

2.3.2 Node Embedding

Graph representation learning methods aim at systematically learning a mapping of each node to a information-rich vector in a dense and continuous space. The mapping function is called node embedding, and the produced vectors are called embeddings, node representations, or hidden representations. The content of an embedding is a set of scalars referred to as ”features”.

These features have no concrete meaning. Instead, they are abstract dimensions that capture the information contained in the original graph, such as node connectivity, neighbourhood structures, and side information.

Formally, a node embedding is a function $g : \mathcal{V} \rightarrow h \in \mathbb{R}^F$ that maps each node from the graph to a vector h, where $F << |\mathcal{V}|$ is the number of dimensions of the embedding space. By combining all embeddings, we get the graph embedding $H \in \mathbb{R}^{N \times F}$. In practice, H is used for inference tasks by implementing a decoder that turns the abstract representation into predictions. [49]

Intuitively, nodes that are similar to each other will have embeddings with high proximity in the embeddings space. Proximity is a broad concept that encapsulates different metrics of similarity. Nodes can have community-based proximity if they are part of the same cluster, geodesic proximity if they are close in terms of shortest path distance, spatial proximity if they have similar attributes, or other types of proximity. [34]

To summarise, an effective graph embedding should:

1. allow for the reconstruction of the original graph;
2. support inference tasks.

2.3.3 Inference Tasks

There are three categories of inference tasks on graphs, each applicable at different levels. First, **classification**; given a set of embeddings, predict the class of nodes, of edges, or of the entire graph. Second, **regression**; given a set of embeddings, predict one or multiple attributes of the nodes or of the edges. Third, **link prediction**; given a set of embeddings, predict the existence of a link between pairs of nodes.

Another distinction is between transductive and inductive learning. The former involves tasks where the predictions are made on entities that the model already saw during training. The latter relates to tasks where the predictions are made on new entities, which reveals the ability of the model to generalise. [49]

($\bigstar$) Because we appended trade volumes on the edges, we can formulate the commodity trade problem as an edge regression task: predict the edge weights given the structure of the graph, country-specific attributes on the nodes, and pair-specific features on the edges. Additionally, we are in a transductive setting, as countries in the graph remain unchanged between training and testing.

2.3.4 Early Approaches

Early advancements in graph embedding research are based on dimensionality reduction techniques, which map data points from a high-dimensional space into lower-dimensional embeddings. Their primary objective is to learn representations that, upon being projected back to the original space, closely approximate the original points. Early approaches thus focused on the first requirement of effective graph embedding.

An example is Isomap by J. Tenenbaum et al. (2000), a non-linear dimensionality reduction technique that involves modelling a dataset as a graph and calculating shortest paths between point neighbourhoods. [44]

2.3.5 Modern Approaches

Modern approaches incorporate more information about the structure the graph. Additionally, they target downstream inference tasks, satisfying the second requirement of effective graph embedding. [49]

DeepWalk by B. Perozzi et al. (2014) is the first popular approach. It defines the neighbourhood of a node by leveraging random walks and iteratively updating embeddings. The core idea is that nodes appearing frequently in each other's paths are more likely to be in the same neighbourhood. Thus, nodes which have similar neighbourhoods will acquire similar representations. [32]

A. Grover et al. (2016) extended this concept with Node2Vec, a second-order random walk. In particular, they bias transition probabilities between nodes at various shortest path distances to strike a balance between breadth-first and depth-first search. Then, Node2Vec iteratively updates embeddings by maximising the likelihood of seeing a neighbourhood for a node conditioned on the node's representation. [15]

Modern approaches also incorporate additional kinds of information that introduce further proximity measures allowing the representation of the nodes can be learned more comprehensively. This information involves side content from nodes and edges, such as types and attributes, or even the error found during supervised training, which establishes a connection between the representation of the nodes and the target task.

2.3.6 Deep Learning Approaches

Over the last 10 years, deep learning has emerged as the leading paradigm in machine learning, with superior performance in many domains such as computer vision and natural language processing. Y. LeCun et al. (2015) define deep learning as a set of representation learning methods obtained by combining multiple non-linear modules that each transform the representation at one level into a representation at a higher, slightly more abstract level. [26]

However, deep learning methods are inapplicable to graph data. First, graphs are highly irregular structures and have no spatial locality, meaning that nodes can be arranged in any arbitrary shape and without affecting the information they together contain. This makes it difficult to generalise mathematical operations.

Furthermore, the type and properties of graphs depend on the specific problem that is being modelled. Two graphs can be isomorphic, meaning that they have the same structure but different names and properties of the nodes. This further amplifies the issue of generalising operations. In contrast, any image or text can be converted into a standardised matrix or array of numbers.

Additionally, large-scale graphs can have millions of nodes and billions of edges. This poses a significant constraint on the complexity of the models, which ideally have to scale linearly with respect to graph size. Finally, graphs can hold side information, so they require models flexible enough to incorporate this knowledge.

The most effective solution to these challenges is Graph Neural Networks (GNNs). In February 2020, a team of researchers from MIT used GNNs to discover halicin, a novel antibiotic capable of killing the world's worst disease-causing bacteria. [41] This achievement highlights just one of the many potential applications driving research in GNNs, which we will explore in detail in the next chapter.

Graph Neural Networks

3.1 Background

3.1.1 Problem Definition

Let $G = (\mathcal{V}, \mathcal{E})$ be a graph with $\mathcal{V}$ the set of nodes of size $|\mathcal{V}| = N$ and $\mathcal{E}$ the set of edges connecting the nodes. Define also $A \in \mathbb{R}^{N \times N}$ as the adjacency matrix, $X \in \mathbb{R}^{N \times d}$ as the node attribute matrix, and $E \in R^{|\mathcal{V}| \times |\mathcal{V}| \times p}$ as the edge attribute tensor. Here, d and p indicate the number of attributes per node and per edge respectively. A Graph Neural Network (GNN) will learn a representations of the graph in a continuous space while preserving as much information from that graph as possible. The output is a graph embedding $H \in \mathbb{R}^{N \times F}$.

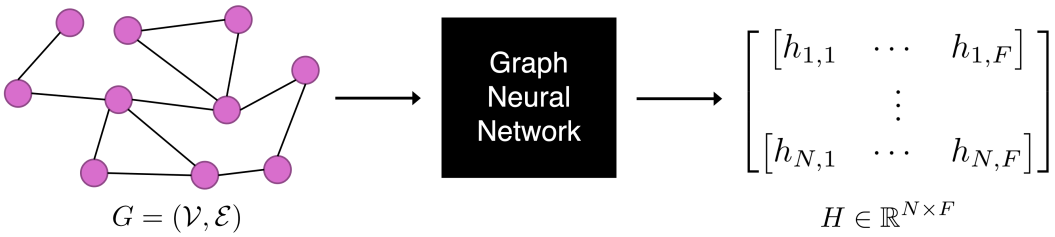

Figure 3.1: The core "black-box" idea of GNNs: input G and receive H.

3.1.2 Learning Approaches

Representation learning with GNNs can be categorized into supervised and unsupervised techniques. Supervised GNNs operate on labelled data, where each input data point is paired with a target label. In this way, the model learns embeddings that achieve high-accuracy predictions on a certain downstream task.

Unsupervised GNNs use unlabelled inputs and instead focus on discovering latent variables of the nodes. These are based on successful unsupervised learning methods, including Variational Autoencoders and Deep Graph Infomax. [22][47] The discussion in this text will focus on supervised GNNs.

3.2 General Framework

At their core, GNNs are a node embedding function that applies neural architectures on graph-structured data using the neighbourhood aggregation strategy. This means that they leverage the local neighbourhoods of the nodes to learn their representations.

Specifically, the embedding for each node i is obtained by aggregating the embeddings of the neighbours $j \in \mathcal{N}(i)$ and combining them with the embedding of i itself from the previous iteration.

Each layer of a GNN equals a level of aggregation. After ℓ aggregations, the embedding of node i, denoted with $h_i^{(\ell)} \in \mathbb{R}^F$, captures the structural information within ℓ-hop neighbourhoods of i. [51] Notably, because we don't have a canonical ordering of the neighbours, the node embedding function must be permutation invariant, meaning that the same representation bust be obtained independently the order with which neighbours are handled.

Given a node i, its embedding at the ℓ-th layer is updated through a propagation rule, composed of two functions:

1. $AGGREGATE$, which merges the information from the ℓ-hop neighbours of i;
2. $COMBINE$, which updates the embedding of i by combining the aggregated information from $\mathcal{N}(i)$ with the embedding of i from the previous layer.

Here is the general framework for GNNs.

Algorithm 1 General Framework for Graph Neural Networks

Initialisation: $H^{(0)} = Q$
for $\ell = 1, 2, \ldots, \mathscr{L}$ **do**
  for each $i \in \mathcal{V}$ **do**
    $agg_i^{(\ell)} = AGGREGATE^{(\ell)}\left(h_j^{(\ell-1)} \mid j \in \mathcal{N}(i)\right)$
    $h_i^{(\ell)} = COMBINE^{(\ell)}\left(h_i^{(\ell-1)}, agg_i^{(\ell)}\right)$
  end for
end for

Consider Q as a matrix for the attributes of the nodes and the edges; $\mathscr{L}$ as the hyper-parameter indicating the number of layers in the GNN; $agg_i^{(\ell)}$ as the aggregated embedding from all neighbours of node i; $\mathcal{N}(i)$ as the set of the 1-hop neighbour nodes of i; $h_i^{(\ell)}$ as the updated embedding for i. The combination of embeddings at the final layer $H^{(\mathscr{L})}$ is treated as the final graph embedding. [49]

The $AGGREGATE$ function generally follows this expression:

$$AGGREGATE(v) = \phi(i, \square_{j \in \mathcal{N}(i)} \psi(j))$$

where $\square$ stands for a permutation invariant operator like sum or maximum, ψ is a learnable function that transforms the neighbours' representations, and ϕ is another learnable function that updates the representation of node i using the aggregated representations of its neighbours $\mathcal{N}(i)$. [6]

This framework applies to all types of GNNs. The differences between each type are the $AGGREGATE$ and $COMBINE$ functions, how many layers of aggregation are used, and what additional information is included from the attributes of the nodes and the edges. We now discuss the three main flavours of graph neural networks.

3.3 Graph Convolutional Networks

3.3.1 Convolutions

A convolution is a mathematical operation between two functions f and g that produces a third function $(f * g)$. It is defined as the set of sums, for discrete cases, or the set of integrals, for continuous cases, of the two functions after one is reversed and shifted.

Convolutions play a central role in Convolutional Neural Networks (CNNs), a deep learning method for image processing. [25] In this context, the two functions to be combined are an image and the kernel.

The kernel, or filter, is a grid of learnable parameters that is slid across the pixels of the image. At each step of the convolution, the kernel computes a dot product between its parameters and the values of the pixels of the image, effectively mapping the result onto a smaller grid called a feature map.

The weights of the kernel are initialised at random and optimised through back-propagation to identify meaningful features of the image. Finally, multiple kernels are trained simultaneously to learn different features.

Kernels are useful because they reduce the size of the representation of the image. The size of the kernel defines the receptive field size of each neuron in the neural network that follows the convolution layers. Loosely speaking, the smaller the kernel, the finer the patterns that the CNN can identify, as each neuron will be connected to a smaller region of the input image. [36]

3.3.2 Convolutions on Graphs

Inspired by CNNs, Kipf et al. (2016) introduced Graph Convolutional Networks (GCN). [23] The fundamental idea remains unchanged; a kernel is slid over the graph, and the embeddings are formed as a weighted sum of the embeddings of the neighbours, but now the size of the kernel varies according to each node's 1-hop neighbourhood.

The propagation rule of GCN at the $\ell + 1$-th layer is:

$$H^{(\ell+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(\ell)}W^{(\ell)}) = \sigma(\hat{A}H^{(\ell)}W^{(\ell)})$$

Here, $\tilde{A} = A + I$, with $I \in \mathbb{R}^{N\times N}$ being the identity matrix. It is added to the adjacency matrix to incorporate self-loops, making a node a neighbour of itself, and thus incorporating its own features during the updating of the representations. $\tilde{D}$ is a diagonal matrix called the degree matrix, with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ representing the number of connections of node i plus 1.

$\sigma(\cdot)$ is some activation function like the Rectified Linear Unit (ReLU) which introduces non-linearity. $H^{(\ell)} \in \mathbb{R}^{N\times F}$ is a matrix of node representation at the ℓ-th layer, with the first layer $H^{(0)} = X$ being the node features matrix. $W^{(\ell)} \in \mathbb{R}^{F\times F'}$ is the collection of kernels, a learnable matrix that is trained during the optimisation.

Integers F and F' indicate the dimensions of the embedding spaces at the ℓ-th and $\ell + 1$-th layers, respectively. To better understand this propagation rule, we unroll its operations for a single node i:

$$h_i^{(\ell)} = \sigma(\sum_{j \in N(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii}\tilde{D}_{jj}}} h_j^{(\ell-1)} W^{(\ell)} + \frac{1}{\tilde{D}_i} h_i^{(\ell-1)} W^{(\ell)})$$

The first part inside σ is the $AGGREGATE$ function. It is a sum of all the node representations from the previous layer $\ell - 1$, weighted by $W^{(\ell)}$ and scaled by entries from the adjacency and degree matrices. The second part is the $COMBINE$ function. It says that the aggregated representations of the neighbours are combined via summation to the weighted and scaled representation of node i from the previous layer. Figure 3.2 depicts the propagation rule for GCN.

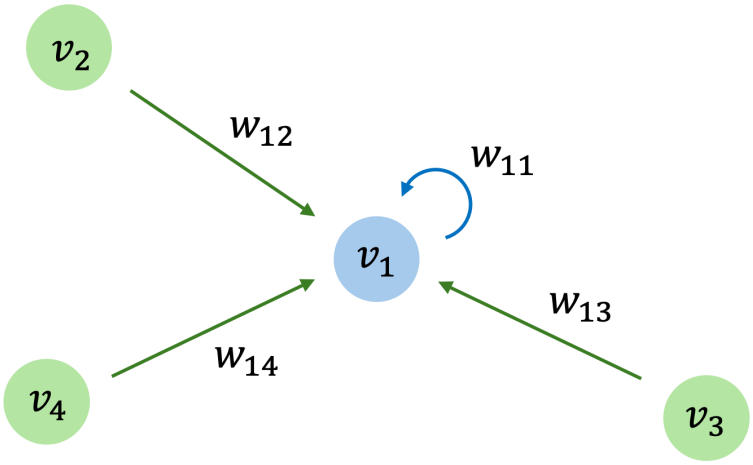

Figure 3.2: Propagation rule for GCN.

The formulation of GCN is quite similar to that of Neural Networks. The main difference is in the way the adjacency matrix is scaled into $\hat{A}$. To understand how the scaling coefficients are obtained, we go to the realm of Spectral Graph Theory.

3.3.3 Spectral Graph Theory Justification

Spectral Graph Theory is the study of graphs through linear algebra. There are two important matrices that we can extract from a graph: the adjacency matrix A and the Laplacian matrix L_G. They are linked by the relationship $L_G = D - A$.

Formally, the Laplacian L_G of a graph $G = (V, E)$ is a matrix whose entries $l_G(i, j)$ are: -1, if $\{i, j\} \in E$; $degree(i)$, if $i = j$; and 0 otherwise. [27] L_G is used to study various properties of graphs, such as connectivity and clustering. In GCN, the Laplacian matrix is used to approximate a spectral convolution on graphs.

A spectral convolution $g_\theta * x := U g_\theta U^T x$ is defined as the product between a convolutional kernel g_θ, parameterised by $\theta \in \mathbb{R}^N$, and the signals from all the nodes $x \in \mathbb{R}^{N \times 1}$. Here, U is obtained via spectral decomposition $\tilde{L_G} = U \Delta U^T$ of the normalised Laplacian matrix $\tilde{L_G} = I - D^{-\frac{1}{2}} L_G D^{-\frac{1}{2}}$. A series of steps to alleviate the computational cost and reduce over-fitting risk that we will not cover lead to the following approximated layer-wise linear model: $g_\theta * x \approx \theta(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})$.

There still persists one issue: the eigenvalues of the normalised Laplacian matrix lie in the interval $[0, 2]$. A GCN with many aggregation layers risks incurring either vanishing gradients, with eigenvalues close to 0, or exploding gradients, with eigenvalues close to 2. Thus, the final re-normalisation operation $\tilde{A} = A + I$ is applied, ensuring that eigenvalues are in the range $[-1, 1]$. Generalising to nodes with d features yields

$$H = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$$

By adding σ and noticing that $\Theta \in \mathbb{R}^{d \times F} = W^{(1)}$, we find the propagation rule of GCN.

3.4 Graph Attention Networks

3.4.1 Attention

Attention has become ubiquitous among machine learning methodologies, including the influential Transformer. [46] The concept was introduced as part of a Neural Machine Translation architecture. [2] Neural Machine Translation aims at outperforming traditional phrase-based translation of natural language using a single neural network.

The authors approach translation probabilistically. Given a source sentence $\mathcal{X} = (x_1, ..., x_{T_\mathcal{X}})$ in one language and the target sentence $\mathcal{Y} = (y_1, ..., y_{T_\mathcal{Y}})$ in another language, both stored as a sequence of different length of vectors for words, translation finds the target word that maximises the conditional probability of $\mathcal{Y}$ given $\mathcal{X}$:

$$P(y_1, ..., y_{T_\mathcal{Y}} \mid x_1, .., x_{T_\mathcal{X}}) = \prod_{t=1}^{T_\mathcal{Y}} P(y_t \mid C, y_1, ..., y_{t-1})$$

where C is a special vector containing information about the context of the word under prediction.

The common way to solve this problem is to parameterise $P(\mathcal{Y}|\mathcal{X})$ and train it on a corpus of sentence pairs. After learning the distribution, the model takes a source sentence and outputs the translation that maximises the conditional probability.

[2] proposes the following architecture. First, a bidirectional Recurrent Neural Network is used to encode the input sequence into embeddings $(h_1, ..., h_{T_x})$ which we will call annotations. [38] Intuitively, annotation h_j for word x_j contains a summary of the words preceding and following x_j. Because RNNs tend to represent better recent inputs, the summaries will focus on the words closer to x_j. We will return to RNNs in Chapter 5.

The concept of attention is introduced in the decoder. The function to model the conditional probability of each word is defined as $g(y_{i-1}, s_i, c_i) = P(y_i|y_1, ..., y_{i-1}, x)$, where $s_i = f(s_{i-1}, y_{i-1}, C_i)$ is the RNN hidden state, for $i = 1, ..., T_{\mathcal{Y}}$ the number of words in the target sentence. C_i is a context vector, defined as a weighted sum of the annotations of the source sentence, where each weight α_{ij} is between 0 and 1.

$$C_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \ , \quad \alpha_{ij} = Softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_{j=1}^{T_x} exp(e_{ij})}$$

In turn, $e_{ij} = a(s_{i-1}, h_j)$. At last, the function $a(\cdot)$ is an alignment mechanism that quantifies how similar or aligned the inputs around position j are to the output at position i. Since e_{ij} is normalised with the Softmax activation function, α_{ij} can be interpreted as the probability that target word y_i is translated from source word x_j. The higher e_{ij} and α_{ij}, the more important h_j will be to the context C_i of word y_i. In other words, through $a(\cdot)$, the decoder now knows to which part of the input it needs to pay more attention.

3.4.2 Attention on Graphs

Graph Convolutional Networks have two limitations. The first is that the importance of each node to its neighbours is solely determined by the weight on the edges, which is stored in the adjacency matrix. The second is that the representation is constructed from the eigenbasis of the Laplacian matrix, which itself depends on the structure of the graph, making the learned kernel ineffective for graphs with different structures.

Graph Attention Networks (GATs) were developed by Veličkovič P. et al. (2017) to address both these problems. [28] GAT leverages self-attention to compute representations of each node by attending the node's neighbours and determining which have the most important features. The term "self" indicates that attention is applied to the input sequence itself rather than to another target sequence, as in the original method.

First, GAT defines a linear transformation on the embeddings from the previous layer $H^{(\ell-1)} \in \mathbb{R}^{N \times F}$ to embeddings of the new layer $H^{(\ell)} \in \mathbb{R}^{F \times F'}$ with a learnable weight matrix $W \in \mathbb{R}^{F \times F'}$. Then, GAT performs masked self-attention, computing the attention coefficient for each pair of nodes using the attention function $a : \mathbb{R}^F \times \mathbb{R}^{F'} \to \mathbb{R}^{F'}$:

$$e_{ij} = a(Wh_i^{(\ell-1)}, Wh_j^{(\ell-1)})$$

Intuitively, the attention coefficient e_{ij} represents the importance of node's j features to node i. Crucially, a mask is applied such that e_{ij} is computed only for nodes $j \in N(i)$, and for $j \notin \mathcal{N}(i)$, $e_{ij} = -\infty$. For the attention function, the authors use a single-layer feed-forward neural network with with a Leaky ReLU activation function. The neural networks is parameterised by a weight vector $a \in \mathbb{R}^{2F'}$ that learned during training together with the other components. Finally, Softmax is applied to get the scaled attention coefficient:

$$\alpha_{ij} = Softmax(e_{ij}) = \frac{exp(LeakyReLU(a^T[Wh_i||Wh_j]))}{\sum_{k \in N(i)} exp(LeakyReLU(a^T[Wh_i||Wh_k]))}$$

where $||$ represents the concatenation operation and layer superscripts have been omitted for clarity. Then, the new node representation for node i is obtained via linear combination of the representations of the neighbours, with the scaled attention coefficients acting as weights.

$$h_i^{(\ell)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j^{(\ell-1)}\right)$$

Through testing, the authors discovered that running multiple attention heads improved the stability of the learning process. An attention head is one instance of the mechanism described above.

The final node representation is defined as either the concatenation or the average of the features found by the K attention heads. Therefore, the propagation rule of GAT for a single node i is:

$$h_i^{(\ell)} = \Big\Vert_{k=1}^{K} \sigma\Big(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(k)} W^{(k)} h_i^{(\ell-1)} \Big) \quad \text{or} \quad h_i^{(\ell)} = \sigma\Big(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(k)} W^{(k)} h_i^{(\ell-1)} \Big)$$

GAT fuses the $AGGREGATE$ and $COMBINE$ parts into one as the weighted sum of the embeddings of the neighbours across all the attention heads. The propagation rule is visualised in Figure 3.3

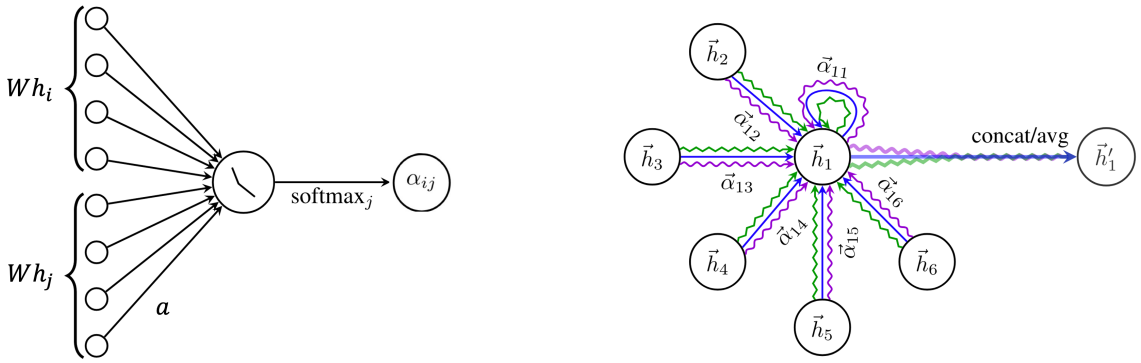

Figure 3.3: Left: GAT's attention mechanism. Right: GAT's multi-head attention, where each colour is a different attention head. [28]

3.4.3 GATv2

The attention mechanism in GAT has a major drawback: it relies on a single ranking of the features of the neighbours. When calculating the attention coefficient e_{ij} for pairs of nodes i, j, for all $j \in \mathcal{N}(i)$, the weights in the attention mechanism a are the same. This means that the importance assignment of the features of node j is independent of node i, limiting the expressive power of the model, i.e. how accurately the model can represent the relationships in the graph.

To address this issue, Brody S. et al. (2021) present GATv2, a modified version of GAT. [5] In particular, they suggest altering the order of internal operations and have first the concatenation of the embeddings, then the multiplication via the learnable weight matrix, then the non linearity, and finally the attention function.

$$e_{ij} = LeakyReLU(a^T(Wh_i||Wh_j)) \quad \text{Attention coefficients in GAT.}$$
$$e_{ij} = a^T LeakyReLU(W(h_i||h_j)) \quad \text{Attention coefficients in GATv2.}$$

In this way, the attention mechanism is augmented from a single array of weights into a multi-layer perceptron that assigns different importance to the features of the neighbours based on the node and the neighbour being considered. For the same computational complexity, the authors found that this alternative attention mechanism achieves higher accuracy compared to GAT.

3.5 Message Passing Neural Networks

The third flavour of GNNs comes from the field of computational chemistry, specifically the problem of learning graph representations of molecules. QM9 is the benchmark dataset in the field, containing data about 130 thousand molecules, each with 13 quantum mechanical properties. [17] Many GNN applications have been developed thanks to QM9, from predicting molecular properties, to modelling interaction networks.

In this regard, Gilmer J. et al. (2017) formulate a single framework for all GNNs called Message Passing Neural Networks (MPNNs). [12] MPNNs introduce the concept of messages, i.e. the information that node j contributes to node i. This notion gives a more intuitive interpretation of how information is propagated during learning.

The forward pass entails two steps: message passing and readout. Message passing is repeated for $\mathscr{L}$ time-steps, or layers, and each layer is composed of a message function M_ℓ and an update function U_ℓ. For each node i, M_ℓ aggregates the messages from the neighbours $j \in \mathcal{N}(i)$, and U_ℓ combines the aggregated message with the current representation of the node to get the representation at the $\ell + 1$-th layer. The propagation rules of MPNN, also visualised in Figure 3.4, are:

$$\text{Message} \quad m_i^{(\ell+1)} = \sum_{j \in \mathcal{N}(i)} M_\ell(h_i^{(\ell)}, h_j^{(\ell)}, e_{ij})$$
$$\text{Update} \quad h_i^{(\ell+1)} = U_\ell(h_i^{(\ell)}, m_i^{(\ell+1)})$$

where $e_{ij} \in \mathbb{R}^p$ is a vector of edge attributes of dimension p. After $\mathscr{L}$ layers, readout computes a single feature vector as a summary for the whole graph.

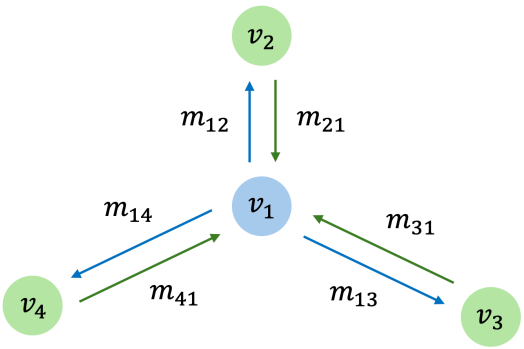

Figure 3.4: Message passing concept: each node sends information to its neighbours.

MPNN defines $AGGREGATE$ in the Message stage and $COMBINE$ in the Update stage. The crucial difference between MPNN and the other two flavours of GNNs is that MPNN generalises the $AGGREGATE$ function to any non-linear learnable function depending on the embeddings of a node and its neighbours.

3.6 Depth Concerns

The GNN flavours encompass many models, each designed for specific task. Give the variety of problems, there is no single optimal GNN design, and decisions must be made on a case-by-case basis.

One of the key design choices is the number of layers, referred to as the model's depth in deep learning. A GNN with k layers will integrate information from k-hop neighbourhoods, the receptive field of its embeddings. However, GNN performance degrades with sub-optimal number of layers. We discuss the main scenarios, summarised in Figure 3.5.

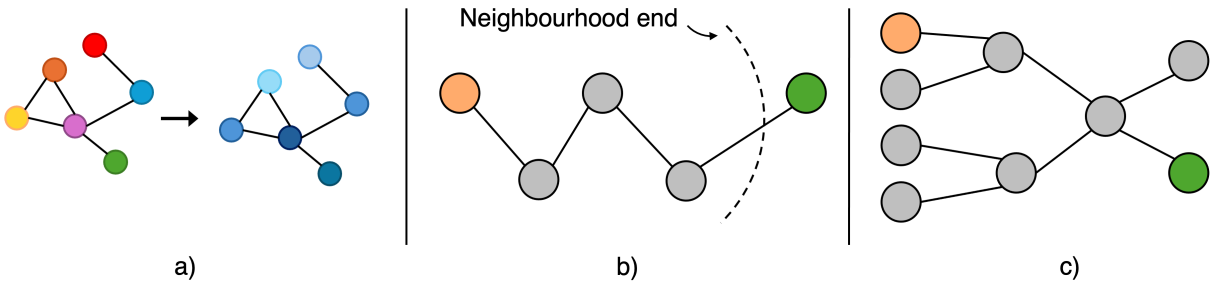

Figure 3.5: a) Over-Smoothing: node representations become too similar. b) Under-Reaching. c) Over-Squashing. In b) and c), the yellow and green nodes indicate the start and end respectively of a long-range dependency.

3.6.1 Over-Smoothing

The expressivity of many deep learning models relies on having multiple layers. For example, Residual Networks (ResNet), a deep learning architecture for image recognition tasks, can have from fifty to thousands of layers. [18] In contrast, most GNN architectures often have just a few layers to prevent over-smoothing.

Over-smoothing is defined as the convergence of all the node features towards the same constant vector of values. [35] As the number of layers increases, the neighbourhood of each node expands until it becomes the entire graph. Because every node now attends to every other node, the resulting embeddings get "smoothed out", becoming similar one to another and resembling the average information contained in the graph. Thus, the model fails to learn distinct and useful representations.

Over-smoothing is gauged through a node-similarity measure $\mu : \mathbb{R}^{N \times F} \to \mathbb{R}_{\geq 0}$ that must equal to 0 if and only if all the embeddings h_i are equal to some constant $c \in \mathbb{R}^F$. Then, over-smoothing is the layer-wise exponential convergence of μ to 0: $\mu(H^{(\ell)}) \leq C_1 e^{-C_2 \ell}$. $C_1, C_2 \in \mathbb{R}^+$ respectively indicate the initial node-similarity value and the rate of decay. An example of node-similarity measure is Dirichlet energy, which finds the average norm of the difference between each node embedding:

$$\mathscr{E}(H^{(\ell)}) = \frac{1}{N} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}(i)} ||h_i^{(\ell)} - h_j^{(\ell)}||_2^2$$

Over-smoothing occurs when a model that has too many layers relative to the graph it is processing. An alternative to reducing the number of layers is normalisation. For instance, PairNorm maintains pairwise distances constant throughout every layer via the following two operations applied at the end of each layer [54]:

$$\text{1. Centering} \quad h_i = h_i - \frac{1}{N} \sum_{j=1}^{N} h_j$$

$$\text{2. Scaling} \quad h_i = \frac{\lambda h_i}{\sqrt{\frac{1}{N} \sum_{j=1}^{N} ||h_j||_2^2}}$$

where $\lambda > 0$ is a hyperparameter.

3.6.2 Under-Reaching and Over-Squashing

Under-reaching and over-squashing occur when a GNN model fails to represent long-range dependencies, which are relationships among distant nodes in the graph. Using the idea of messages from MPNNs, if a network has long-range dependencies, then messages between non-adjacent but dependent nodes should be propagated along the network efficiently.

Under-reaching happens when the model has too few layers to propagate the message. If two nodes i, j share a long-range dependency and are δ hops distant, but the GNN has only $\mathscr{L} < \delta$ layers, then the message from i will never reach j, and vice-versa. The solution would be to increase the number layers up to δ.

Over-squashing is more nuanced. Assuming that $\mathscr{L} \geq \delta$, the message from j to i is still distorted by the messages coming from the other nodes in the receptive field of i. While a larger $\mathscr{L}$ includes longer dependencies, it also squashes more messages into a fixed-size vector, harming the information transfer between j and i.

If the aggregate function ψ and combine function ϕ of the GNN are differentiable, over-squashing can be quantified in terms of one node representation $h_i^{(\ell)}$ failing to be affected by some feature $z \in h_j^{(\ell)}$ of node j at some distance δ of node i. This can be expressed in terms of the Jacobian of the node:

$$\left| \frac{\partial h_i^{(\delta+1)}}{\partial z} \right| \leq (\alpha\beta)^{\delta+1} (\hat{A}^{(\delta+1)})_{iz}$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, and $\alpha \geq |\nabla \psi_\ell|$ and $\beta \geq |\nabla \phi_\ell|$, for $\ell \leq \delta$. Intuitively, if the message and update functions at the ℓ-th layer have bounded derivatives, then message propagation from feature z of node j to node i is controlled by a suitable power of the normalised adjacency matrix of the graph. [45]

Strategies to mitigate over-squashing involve pooling and rewiring. Graph rewiring involves modifying the structure of the graph by adding and deleting edges to enhance the performance of GNNs. [1]

($\bigstar$) In terms of expressivity, we expect GATv2 to perform the best due to the multi-head attention mechanism, which helps select important features from neighbours. However, the message passing analogy of MPNN better represents trade, the passing of goods across countries.

Also, model depth issues are likely. If we focus on regional trade, such as within the European Union, we would have a smaller graph and be more prone to over-smoothing. Conversely, including all countries in the world would introduce long-range dependencies, leading to higher chances of over-squashing and under-reaching.

Moreover, we have yet to address the temporal dimension of the problem. The graph and the GNN flavours cannot incorporate time series data. We need to upgrade to dynamic graphs.

Dynamic Graphs

4.1 Temporal Dimension

4.1.1 Motivation

Systems of interacting entities are ubiquitous, and graphs are one powerful tool to accurately model them. However, using a single graph can be limiting, as it can represent only one static snapshot of the system. In contrast, systems from the real world are dynamic. Over time, new entities may join the system, existing entities may leave it, properties of the entities and their relationship may change, and even the underlying laws governing the system may evolve.

4.1.2 Evolution

Before defining dynamic graphs, we need to consider which components of graphs may vary over time. First, there is structural evolution. This includes the addition or deletion of nodes, for systems where entities can join and leave some platform, community, or general grouping. Similarly, edges can also be added or deleted, reflecting the dynamic relationships among entities.

Then, there is side information evolution, which models systems where attributes of the nodes and/or the edges evolve over time. The nature and causes of these changes depends entirely on the system being modelled. [49]

(★) In the commodity trade problem, we would primarily see side information evolution. This applies for both country-specific attributes, such as stocks and budgets, and pair-specific attributes, such as the currency exchange rate and the trade volume. All these features are expected to vary on a monthly basis.

Structural evolution is also possible when new trade relationships are established or terminated, represented by the addition or deletion of an edge, respectively. The nodes are unlikely to change, as it would represent the creation or the dissolution of a country.

4.2 Taxonomy

The definition of a dynamic graph depends on whether time is considered as a discrete or a continuous variable. This distinction defines the two families of dynamic graphs: Discrete Time Dynamic Graphs and Continuous Time Dynamic Graphs.

4.2.1 Discrete Time

Time is considered a discrete variable when graph evolution occurs at a fixed rate, with variations in the graph spaced equally in time. Discrete time may be an intrinsic property of the modelled system or a result of collecting data at regular intervals.

We define a Discrete Time Dynamic Graph (DTDG) as a sequence of snapshots $\mathcal{G} = (G_1, \cdots, G_T)$, for $t = 1, .., T$ equally-spaced timestamps. Each snapshot acts as a static frame of $\mathcal{G}$ and is itself a graph $G_t = (\mathcal{V}_t, \mathcal{E}_t, X_t, E_t)$ with a node set $\mathcal{V}_t$ of size $N_t = |\mathcal{V}_t|$, an edge set $\mathcal{E}_t$, a node attribute matrix $X_t \in \mathbb{R}^{N_t \times d}$, and edge attribute tensor $E_t \in \mathbb{R}^{N_t \times N_t \times p}$. Under the general definition of DTDGs, the content and size of each component can change between snapshots.

A special case of DTDG is Spatio-Temporal graphs. They are characterised by a time-invariant structure, meaning that $\mathcal{V}_t = \mathcal{V}$ and $\mathcal{E}_t = \mathcal{E}$ for all t. A spatio-temporal graph is denoted as a sequence of graphs $\mathcal{G} = (G_1, \cdots, G_T)$, where each snapshot $G_t = (\mathcal{V}, \mathcal{E}, X_t, E_t)$ has only node and edge information dependent on time, as indicated by the subscripts. The name indicates how evolution occurs only in the spatial dimension of the graph, which is defined by the nodes' and edges' features. [14]

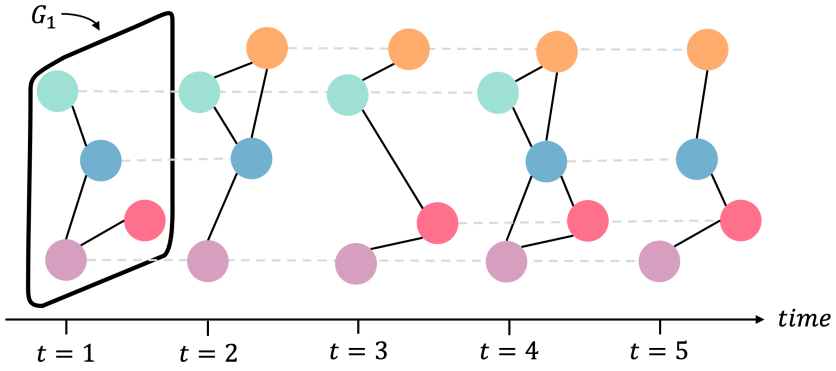

Figure 4.1: Structural evolution of a DTDG defined for five time stamps. [14]

(★) Under certain constraints, the commodity trade problem can be modelled with spatio-temporal graphs. First, data comes in monthly intervals, a discrete time measure. Second, the node set is fixed. We would need to modify the meaning of edges to have the edge set fixed as well. For instance, we could define buyer and supplier roles for the countries and use edges to model all the possible buyer-supplier relationships. Then, the only changes will be the attributes of the countries and the relationships among them, including whether they trade and how much they trade.

4.2.2 Continuous Time

Time is considered as a continuous variable when graph evolution occurs at an irregular rate. As a result, changes in the graph are not equally spaced across time but are instead act as a flow of events. Continuous time arises as an intrinsic property of the system that is being modelled.

Continuous Time Dynamic Graphs (CTDGs) are defined as a tuple $\mathcal{G} = (G_0, \mathcal{O})$, where $G_0 = (\mathcal{V}_0, \mathcal{E}_0, X_0, E_0)$ is a snapshot at $t = 0$ representing the initial state of the graph, and $\mathcal{O} = (o_1, \cdots, o_T)$ is a sequence of observations occurring at any time $t > 0$, for $t = 1, .., T$ unequally-spaced timestamps.

Each observation $o_t = (event\ type, \{i, j\}_{i,j \in \mathcal{V}_t}, t)$ describes an event with the event type, the node/s involved, and the timestamp. The event type can be either a structural evolution, such as node/edge addition/deletion, or any type of feature update. The graph G_t is found by applying in chronological order all the observed events in $\mathcal{O}$ up to time t. [14] Notably, DTDGs are a subset of CTDGs because they can be formulated with observations made at constant intervals.

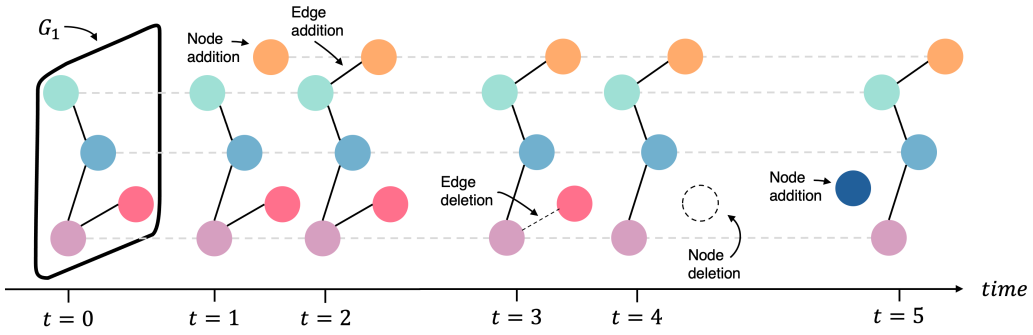

Figure 4.2: Structural evolution of a CTDG defined for five time stamps. [14]

4.2.3 Temporal Neighbours

Dynamic graphs necessitate a new definition of the neighbourhood of a node. Since connections change over time, the temporal neighbourhood of node i is defined as set of nodes j that were initially connected to i or that shared an edge with i at some point in time: $\mathcal{N}_t(i) = \{j \in \mathcal{V}_0 \vee j \in o_{t'} = (add\ edge\ (i,j), \{i,j\} \in \mathcal{V}_{t'}, t') \mid t' < t\ \wedge j \in \mathcal{V}_{t'}\}$.

4.3 Dynamic Graph Learning

Graph Representation Learning on Dynamic Graphs involves learning abstract node representations in a continuous space that preserve the structural relationships, spatial information, and temporal dependencies contained in the graph. The dynamic nature of the graph implies that information is contained not only in the structure and the attributes but also along time. This means that the evolution of the graph might be more informative than the graph's structure and attributes in a single snapshot.

Hence, the node embedding for a dynamic graph with nodes $\mathcal{V}_t$ is defined as a function $g : \mathcal{V}_t \rightarrow z_{i,t} \in \mathbb{R}^F$ that maps each node i to an F-dimensional feature vector $z_{i,t}$. This vector must compress information about the current structure and the attributes of the graph at time t, as well as reflect any past information about i and its neighbourhood. The individual node embeddings are combined into a temporal embedding Z_t for the entire graph, which is then used for downstream inference tasks.

4.3.1 Inference Tasks

Analogous to static graphs, dynamic graphs enable classification, regression, and link prediction at different levels in the graph. The main difference is that inference is now made along the temporal dimension. Because dynamic graphs store sequences of events and time series of features, they effectively become forecasting tools.

Recently, dynamic graphs have been applied to multivariate time-series forecasting. A multivariate time-series is a set of interconnected time series, where each series is a sequence of measurements over time of one variable. The goal of forecasting is to predict future values of the series based on historical trends.

W. Cai et al. (2024) conceptualised each series as a node in a graph, leveraged GNNs to learn dependencies across nodes, reformulated the problem into a node regression task, and achieved state-of-the-art results on multiple benchmarks. [7]

($\bigstar$) Simple data analysis reveals that trade volumes and many other related variables exhibit seasonal trends and lagged causal effects. Therefore, dynamic graph learning methods are the most appropriate for solving the commodity trade problem. Namely, we need to implement a GNN that captures all the structural, spatial, and temporal dynamics of trade and generates a single temporal graph embedding that enables future edge weights forecasting.

We re-define the commodity trade problem as a temporal edge regression task: predict the edge weights given monthly historical data about the structure of the graph, country-specific attributes on the nodes, and pair-specific features on the edges.

Dynamic Graph Neural Networks

5.1 Modelling Approach

Dynamic graphs are sequential in nature, as they consist of either a sequence of static graphs or a sequence of events continuously updating a static graph. Most inference tasks on dynamic graphs involve making predictions along the temporal dimension In contrast, classic Graph Neural Networks (GNNs) are static, considering only the structural and spatial dependencies.

Static GNNs take a graph G as input and produce a static embedding H. For instance, if we input a Discrete Time Dynamic Graph $\mathcal{G} = (G_1, .., G_T)$ into a classical GNN, we receive a sequence of embeddings $\mathcal{H} = (H_1, ..., H_T)$, one for each snapshot. However, these embeddings are temporally independent, meaning that H_t contains no information about H_{t-1} and all the other past node representations. From the GNN's perspective, the ordering of the graphs in $\mathcal{G}$ is irrelevant.

To capture the temporal dimension, Dynamic Graph Neural Networks (DGNNs) enhance GNNs with sequence models. In the following pages, we will cover the fundamentals of various sequence models to understand their impact in DGNNs.

5.2 Sequence Models

Sequence models are a family of neural networks specialised in processing sequential data such as text or audio. For an input sequence $\mathcal{X} = (x_1, ..., x_T), x_t \in \mathbb{R}^d$, the sequential model produces hidden representations $(h_1, ..., h_T)$, where $h_t \in \mathbb{R}^F$ is an F-dimensional embedding capturing the information from the first t observations. The earliest type of sequence model is the Recurrent Neural Network, introduced in 1986.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) extend feed-forward neural networks to sequential data. [37]

Given an input sequence $\mathcal{X} = (x_1, ..., x_T)$ and an output sequence $\mathcal{Y} = (y_1, ..., y_T)$, both of length T, the RNN computes the output y_t for each input x_t through the following two operations:

$$h_t = \vartheta(W^{(hx)}x_t + W^{(hh)}h_{t-1} + b^{(h)})$$
$$y_t = W^{(yh)}h_t + b^{(y)}$$

where h_t is the hidden representation for input x_t, $W^{(hx)}$ and $W^{(hh)}$ are the input-to-hidden and hidden-to-hidden linear transformations, $W^{(yh)}$ is the hidden-to-output linear transformation, $b^{(h)}$ and $b^{(y)}$ are bias terms, and ϑ is the sigmoid activation function, which maps $\mathbb{R}$ into $(0, 1)$. Intuitively, the RNN moves along the input sequence and generates each output based on the current input variable x_t and the representation of the previous $t - 1$ inputs h_{t-1}.

Encoder-Decoder Architecture

If input and output sequences have different lengths $T_{\mathcal{X}} \neq T_{\mathcal{Y}}$, a sequence-to-sequence (seq2seq) or Encoder-Decoder architecture is used. [42] [10] Proposed by independent authors, they share the same idea of combining two RNNs. The first RNN encodes the input sequence into a feature vector of fixed size. The second RNN decodes that vector into the target sequence. Both RNNs are then trained end-to-end on an objective function.

In practice, the encoder and decoder can be any learnable function. The authors of seq2seq point out that RNNs are unable to capture long-term dependencies due to the recursive multiplication of weight matrices, which leads to vanishing or exploding gradients. [31] Instead, they implement a Long-Short Term Memory network.

Long-Short Term Memory

Long Short Term Memory networks (LSTMs) are a variant of RNNs that address the problem of vanishing/exploding gradients by partitioning the information flow into two paths. [19] The first path is the hidden state, representing the Short Term Memory.

It works with an internal recurrence that resembles that of RNNs, combining the previous hidden representation with the current input through parameterised linear transformations. The second path is the cell state, representing the Long Term Memory. It keeps track of long-term dependencies through non-parameterised operations.

Similarly to RNNs, LSTMs produce outputs by scanning the input sequence. For each input $x_t \in \mathbb{R}^d$, the LSTM layer determines how much long-term memory is retained (Forget Gate), how much information from the input is added to the long-term memory (Input Gate), and how much information from the input is added to the short-term memory (Output Gate). A sigmoid function is used to gate each step, meaning that the value to be added to the memory is scaled by a weight between 0 and 1.

Specifically, a single LSTM layer applies the following operations:

$$\begin{aligned}
\text{Forget Gate} \quad & \mathtt{f}_t = \vartheta\left(W^{(\mathtt{f})}x_t + W^{(\mathtt{f}h)}h_{t-1} + b^{(\mathtt{f})}\right) \\
\text{Input Gate} \quad & \mathtt{p}_t = \vartheta\left(W^{(\mathtt{p})}x_t + W^{(\mathtt{p}h)}h_{t-1} + b^{(\mathtt{p})}\right) \\
\text{Output Gate} \quad & \mathtt{o}_t = \vartheta\left(W^{(\mathtt{o})}x_t + W^{(\mathtt{o}h)}h_{t-1} + b^{(\mathtt{o})}\right)
\end{aligned}$$

where t indicates the position of the input, $W^{(\cdot)}$ and $b^{(\cdot)}$ are the learnable weight matrices and biases, h_{t-1} is the hidden state of the previous $t-1$ inputs, and ϑ is the sigmoid activation function.Finally, the model computes [11]:

$$\begin{aligned}
\text{Cell State Update} \quad & \mathtt{c}_t = \mathtt{f}_t \odot \mathtt{c}_{t-1} + \mathtt{p}_t \odot \vartheta\left(W^{(\mathtt{c})}x_t + W^{(\mathtt{c}h)}h_{t-1} + b^{(\mathtt{c})}\right) \\
\text{Hidden State Update} \quad & h_t = \mathtt{o}_t \odot \tanh(\mathtt{c}_t)
\end{aligned}$$

where $W^{(\cdot)}$ and $b^{(\cdot)}$ are additional learnable weight matrices and biases, $\tanh(\cdot)$ is an activation function that maps $\mathbb{R}$ into $(-1, 1)$, and $\odot$ is the Hadamard product.

LSTMs have achieved great performance a wide variety of tasks, such as handwriting recognition. [8] However, the large quantity of parameters makes LSTMs relatively slow to train with many layers and with long sequences. Gated Recurrent Units, introduced in [10], reduce the number of parameters without sacrificing too much performance.

Gated Recurrent Unit

Similar to LSTMs, the Gated Recurrent Unit (GRU) controls the flow of information with two gates. The first gate, called the reset gate, determines whether the information from the previous hidden state is ignored. The second gate, called the update gate, determines how much information from the previous hidden state is added to the current hidden state. [10]

Specifically, a single GRU layer applies the following operations:

$$
\begin{aligned}
\text{Reset Gate} \quad & \mathtt{r}_t = \vartheta\left(W^{(\mathtt{r})}x_t + W^{(\mathtt{r}h)}h_{t-1} + b^{(\mathtt{r})}\right) \\
\text{Update Gate} \quad & \mathtt{u}_t = \vartheta\left(W^{(\mathtt{u})}x_t + W^{(\mathtt{u}h)}h_{t-1} + b^{(\mathtt{u})}\right) \\
\text{Candidate activations} \quad & \tilde{h}_t = \tanh\left(W^{(\tilde{h})}x_t + W^{(\tilde{h}h)}\mathtt{r}_t \odot h_{t-1} + b^{(\tilde{h})}\right) \\
\text{Hidden State Update} \quad & h_t = \mathtt{u}_t \odot h_{t-1} + (1 - \mathtt{u}_t) \odot \tilde{h}_t
\end{aligned}
$$

where $W^{(\cdot)}$ and $b^{(\cdot)}$ are the usual learnable parameters, and $\tilde{h}_t$ contains the possible modifications to apply to h_t. These are computed as a function of the input x_t, the previous hidden representation h_{t-1}, modulated by $\mathtt{r}_t$.

Intuitively, a GRU layer focused on retaining long-term dependencies will have $\mathtt{r}_t$ close to 0 and $\mathtt{u}_t$ close to 1, such that it retains most information from h_{t-1} and minimises the impact of $\tilde{h}_{t-1}$. Conversely, a GRU layer focused on learning short-term trends will have $\mathtt{r}$ close to 1 and $\mathtt{u}$ close to 0, such that more weight is added on the modifications of h_{t-1}. Often, multiple layers are used together to capture both types of dynamics.

5.3 Architectures

There are different ways to combine GNNs with sequence models, but the goal is the same: to create a node representation $Z_t \in \mathbb{R}^{N_t \times F}$ that holds information about the dynamic graph's structure, attributes, and their evolution from $t = 0$ to some arbitrary time t. We will refer to this representation as "temporal embedding".

Stacked

In stacked architectures, GNNs and RNNs are used sequentially. This approach is based on the idea that spatial and temporal dependencies can be computed independently. For DTDGs, the temporal node embedding for snapshot t is created as follows:

$$H_t = GNN(G_t)$$
$$Z_t = RNN(Z_{t-1}, H_t)$$

Intuitively, the GNN produces node representations H_t for the individual snapshots up to time t. Then, the RNN passes over the sequence of representations and produces the temporal graph embedding Z_t. This architecture is depicted below in 5.1.

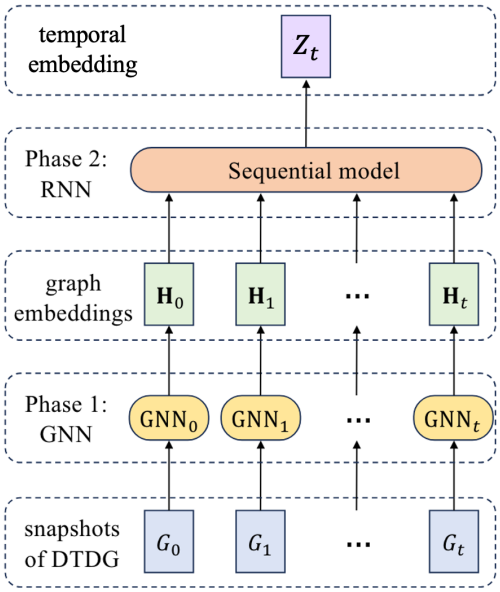

Figure 5.1: General form of the Stacked Architecture. [52]

For instance, Panapoulos et al. (2021) stacked an MPNN layer with an LSTM network to model the spread of COVID-19 across cities during 2020. [29]

In stacked architectures for Continuous Time Dynamic Graphs (CTDGs), each node representation is built in a granular fashion and continuously updated as events occur to that node. The embedding strategy varies between implementations. For example, Temporal GAT leverages attention to explore temporal neighbourhoods and maps time differences into a higher-dimensional space. [50] Similarly, Temporal Graph Networks (TGN) track the evolution of each node using a dedicated memory vector. [33]

Integrated

Integrated architectures incorporate the GNN into the RNN, capturing spatial and temporal information concurrently. This results in a single unit that computes temporal embeddings directly from the sequence of graphs or event observations. The general idea is shown in figure below 5.2.

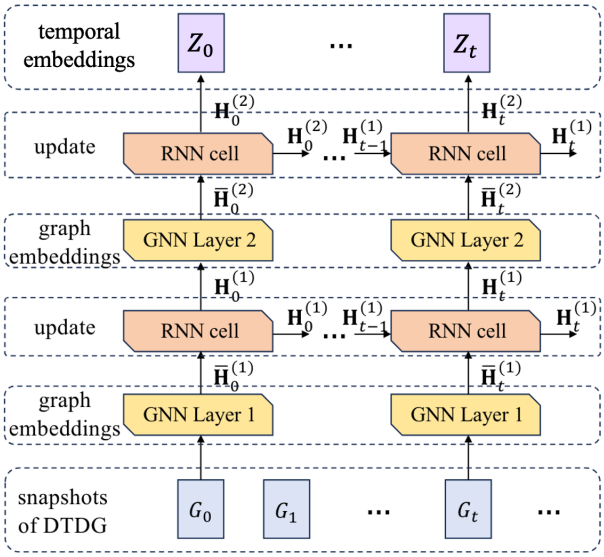

Figure 5.2: General form of the Integrated Architecture. [52]

An example in DTDG is Graph Convolution Embedded LSTM (GC-LSTM), which replaces the weight matrices in the LSTM gates with a GCN layer. [9] GC-LSTM directly processes a sequence of adjacency matrices and learns both structural and temporal dynamics simultaneously. Consider A_t the adjacency matrix at time t. Then, GC-LSTM's gates are generically formulated as:

$$\mathrm{g}_t = \vartheta\left(W^{(\mathrm{g})}A_t + GCN(H_{t-1})\right)$$

Other implementations focus on the sequences of interactions among nodes. An example for CTDGs is JODIE, which uses two RNNs and a temporal attention vector to update individual node representations only when an interaction occurs. [24]

Hybrid

In hybrid architectures, the expressive power of GNNs is enhanced with alternative techniques. For example, the position-encoding injective temporal graph net (PINT) augments TGN with Causal Anonymous Walks (CAW). CAW is a random walk-based algorithm that encodes temporal dependencies by starting from a selected edge and moving backward in time to collect information about adjacent edges. [40]

Another hybrid model is Dynamic Graph Variational Autoencoder (DyGrAE), which adapts auto-encoders to graphs. [43] At time t, DyGrAE encodes snapshot t by average pooling the embedding of a Gated GNN (GGNN), computes the temporal embedding with an LSTM, and reconstructs the adjacency matrix $\bar{A}_t$ with a second LSTM:

$$\begin{aligned} \text{GNN Embedding} \quad & H_t = pool_{avg}(GGNN(G_t)) \\ \text{LSTM Encoder} \quad & Z_t^{enc} = LSTM^{enc}(H_t), Z_{t-1}^{enc}) \\ \text{LSTM Decoder} \quad & Z_t^{dec} = LSTM^{dec}(\bar{A}_{t-1}, Z_{t-1}^{dec}) \end{aligned}$$

Meta

Meta architectures focus on linking the temporal dynamics of the graph to the parameters of the GNN. Instead of having a GNN with fixed weights for all time steps, meta architectures update the weights via an RNN. This results in a GNN layer that evolves with the graph. Pareja et al. (2019) put forward this concept with EvolveGCN. [30] The parameters of a GCN layer are set to either the hidden states of a GRU, or as the output of an LSTM. Considering Θ_{t-1} as the parameters of GCN at the previous time step, the temporal embedding at time t is found with:

$$\begin{aligned} \Theta_t &= GRU(X_t, \Theta_{t-1}) \\ Z_t &= GCN(X_t, \Theta_t) \end{aligned}$$

Most of the architectures we discussed have been developed experimentally by optimising specific objectives Hence, no framework emerges as the clear winner. The best approach varies from one inference task to another.

In the next section we will explore two real-world applications of DGNNs to understand their design. The first application uses DTDGs to forecast road traffic. $(\bigstar)$ The second is an attempt to solve the commodity trade problem using one of the best CTDG framework to date.

5.4 Real World Applications

5.4.1 Traffic Forecasting

Problem Setup

The traffic forecasting problem involves predicting future traffic conditions given past measures such as traffic speed and density. [53] Accurate forecasting is useful for urban planners, who can implement appropriate measures to address traffic increases, and for navigation software like Google Maps, which can identify faster routes and estimate travel durations more accurately.

Traffic involves intricate spatial and temporal dependencies. First, the evolution of traffic is dictated by the road network. If upstream roads receive a wave of cars, then downstream roads are expected to experience part of that wave in the near future. Second, traffic exhibits periodic trends, such as the infamous ”rush hours.” These temporal patterns can further inform predictions.

J. Zhu et al. propose the following framing of the problem. [3] Represent a road network with a spatio-temporal graph $G = (\mathcal{V}, \mathcal{E})$, where the nodes are road sections and undirected, unweighted edges are the connections among these road sections. Furthermore, store traffic speed in a feature matrix $X_t \in \mathbb{R}^{N \times d}$, where d is the length of the sequence of the historical speed measurements for each road segment. The forecasting problem then becomes the following temporal node regression task: predict the traffic speeds for T future time steps using G and a historical series of length $\bar{t}$ of node feature matrices:

$$f(G, (X_{t-\bar{t}}, ..., X_t)) = (X_{t+1}, ..., X_{t+T})$$

where f is the learnable function representing the DGNN model.

DGNN Model

The authors present A3T-GCN, a snapshot-based DGNN that stacks the following three units: a GCN, a GRU, and an attention layer. The 2-layered GCN accounts for the spatial dynamics of the road network at time t:

$$H_t = GCN(X_t, A) = \sigma(\hat{A}\,\sigma(\hat{A}X_tW_0)W_1)$$

with $W_0 \in \mathbb{R}^{d\times F}$ and $W_1 \in \mathbb{R}^{F\times F'}$, where F' is the final dimension of the static node representations. Then, the GRU computes the traffic state at time t:

$$\begin{aligned}
\mathtt{r}_t &= \vartheta(W^{(\mathtt{r})}[H_t, Z_{t-1}] + b^{(\mathtt{r})}) \\
\mathtt{u}_t &= \vartheta(W^{(\mathtt{u})}[H_t, Z_{t-1}] + b^{(\mathtt{u})}) \\
\tilde{Z}_t &= \tanh(W^{(\tilde{Z})}[H_t, \mathtt{r}_t Z_{t-1}] + b^{(\tilde{Z})}) \\
Z_t &= \mathtt{u}_t Z_{t-1} + (1 - \mathtt{u}_t)\tilde{Z}_t
\end{aligned}$$

The attention layer extracts the long-term trend in variation in Z_t by determining which hidden states from the past are most important. A neural network with two layers, parameterised by $((b_{(1)}, w_{(1)})$ and $(b_{(2)}, w_{(2)})$, is used for the attention function:

$$e_i = w_{(2)}(\sigma(w_{(1)}Z_i + b_{(1)})) + b_{(2)}, \qquad \alpha_i = \frac{exp(e_i)}{\sum_{k=1}^{q} exp(e_k)}, \qquad C_t = \sum_{i=1}^{q} \alpha_i Z_i$$

The result is a context vector $C_t \in \mathbb{R}^N$ containing all the spatial and temporal information about the road network at time t. Finally, the predictions for the next T time steps are generated by the decoder, a Multi-Layer Perceptron (MLP) that maps C_t to a vector of predictions $\hat{Y} \in \mathbb{R}^T$. Comprehensively, the architecture is:

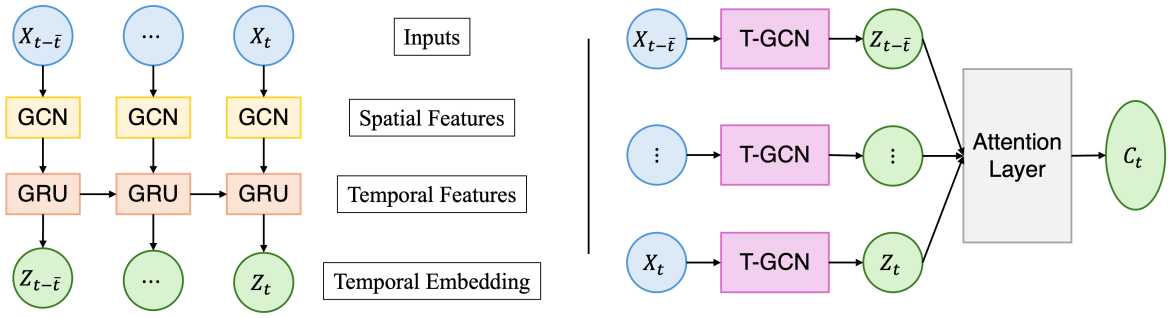

Figure 5.3: Left: Architecture of Dynamic GNN. Right: Attention layer.

Results

A3T-GCN and the decoder are trained end-to-end with backpropagation and loss metric $||Y - \hat{Y}||_2^2$, where Y are the true values of traffic for the next T time steps. The authors compare the results to Auto-regressive Integrated Moving Average (ARIMA), GCN, and GRU, which capture either the temporal or the spatial dynamics of the graph. A3T-GCN scores the highest in accuracy and lowest in Root Mean Squared Error (RMSE), proving its effectiveness in generating expressive representations.

5.4.2 Global Agriculture Trade $(\bigstar)$

Problem Setup

The global agriculture trade problem, introduced by L. Jiang et al. (2023), is the experiment in the research field that most closely resembles our commodity trade problem. [20] Both involve predicting the volumes of commodities traded among countries given past information about trade and other variables such as production and demand. Accurate trade forecasts benefit many parties, from government agencies that can design better policies to companies that can make more precise production plans

Trade occurs in a global and interconnected market that presents both structural and temporal dynamics. The structure of the market is defined by the relationships that countries share, determined by trade agreements, geographical distances, and geopolitical conditions. Activity in the market is dictated by the forces of supply and demand, which are influenced by many variables such as production and available stocks. Most of these variables fluctuate during the year, roughly following seasonal trends, which injects a temporal dimension into the problem.

Agriculture trade can be modelled as a CTDG $\mathcal{G} = (G_0, \mathcal{O})$ with countries as nodes and trades as edges. Each observation $o_t = ((i, j), t, \omega_t) \in \mathcal{O}$ indicates that a directed edge with weight $\omega_t \in \mathbb{R}$ between source node i and destination node j occurred at time t. The weight indicates the volume that has been traded between country i and country j. Moreover, $\mathcal{G}$ has a time varying node set $\mathcal{V}_t$ and edge set $\mathcal{E}_t$.

The agriculture trade problem can then be framed as the following temporal edge regression task; predict the weights of the next T edges given a historical sequence of observations of length $\bar{t}$:

$$f(G_0, \mathcal{O}_{0:\bar{t}} = (o_0, ..., o_{\bar{t}})) = (\omega_{\bar{t}+1}, ..., \omega_{\bar{t}+T})$$

where f is the usual learnable function representing the DGNN.

DGNN Model

The authors implement Temporal Graph Networks (TGN), a framework for deep learning on CTDGs. [33] TGN is an encoder module that represents a CTDG as a sequence of temporal embeddings $Z_t \in \mathbb{R}^{N_t \times F}$, where N_t is the number of nodes at time t. These embeddings are then fed to a decoder to obtain task-specific predictions.

TGN is based on the message passing view of GNNs and runs two subsequent processes in batches of B events. In the original paper, the authors achieved the best results with $B = 200$. First, TGN builds node states with the Memory, Message Function, Message Aggregator, and Memory Updater modules. Then, it generates node embeddings with a single Embedding module. The entire architecture is visualised in Figure 5.4.

Memory. The memory module stores state vectors $s_i(t)$ for each node i that has been observed by the model up to time t. We use the time index in parenthesis $\cdot(t)$. instead of the subscript $\cdot_t$ for clarity. If a node is processed for the first time, its state is initialised to zeros. Similar in RNNs, the state vector of a node holds the historical information of the events that occurred to that node. This module enables TGN to track of long-term trends. The next three modules work towards updating the memory.

Message Function. Each event in the graph provides a message that updates the memory of the involved nodes. The Message Function module (msg_{src}, msg_{dst}) quantifies this message by combining the previous states of the nodes involved in the event. It can be either a learnable function or a concatenation. In our problem, we focus on edge-wise events $e_{ij}(t)$.

Thus, the message function will compute a value for both the source node and the destination node:

$$m_i(t) = msg_{src}(s_i(t^-), s_j(t^-), \Delta t, e_{ij}(t)), \quad m_i(t) = msg_{dst}(s_j(t^-), s_i(t^-), \Delta t, e_{ij}(t))$$

where $s_i(t^-)$ is the state of node i before event $e_{ij}(t)$ happened.

Message Aggregator. TGN processes events in batches. This can lead to a situation where node i is involved in multiple events, resulting in sequence of messages $(m_i(t_a), ..., m_i(t_b))$. The Message Aggregator module compresses the sequence into a single aggregated message $\bar{m}_i(t) = agg(m_i(t_a), ..., m_i(t_b))$, where agg is a learnable function, like RNNs, or a simple operation like taking the most recent message.

Memory Updater. The memory updater module completes the update of the memory by computing the new node states with a learnable function mem, such as an LSTM:

$$s_i(t) = mem(\bar{m}_i(t), s_i(t^-))$$

After updating the memory, TGN implements an embedding module to generate node representations at time t. This prevents embeddings from becoming stale. Memory staleness occurs when a node's state $s_i(t)$ becomes obsolete due to prolonged lack of events involving i. [21] During this period, the rest of the graph evolves, causing $s_i(t)$ to miss recent trends and thus degrading the overall quality of the node representations.

Embedding. TGN addresses staleness with by aggregating states and events from the neighbourhood. When a node has been inactive for some time, its likely that its neighbours have experienced events and thus received state updates. The neighbourhood aggregation propagates information to the inactive node. In particular, the embedding module computes the temporal embedding for node i at time t with:

$$z_i(t) = \sum_{j \in \mathcal{N}_i^k([0,t])} f(s_i(t), s_j(t), e_{ij})$$

where f is a DGNN and $\mathcal{N}_i^k([0,t])$ is the k-hop neighbourhood of node i for all timestamps $\leq t$.

In the original paper, TGN performed best with Temporal GAT (TGAT), a DGNN architecture that leverages attention to select the most important neighbors based on their attributes and the timing of the information. [34] However, when solving the agriculture trade problem, the authors used a Transformer Convolution layer. [39]

Finally, the predictions are obtained with an MLP that maps Z_t into a vector of edge weight predictions $\hat{Y} \in \mathbb{R}^T = (\omega_{\bar{t}+1}, ..., \omega_{\bar{t}+T})$ ordered by timestamp.

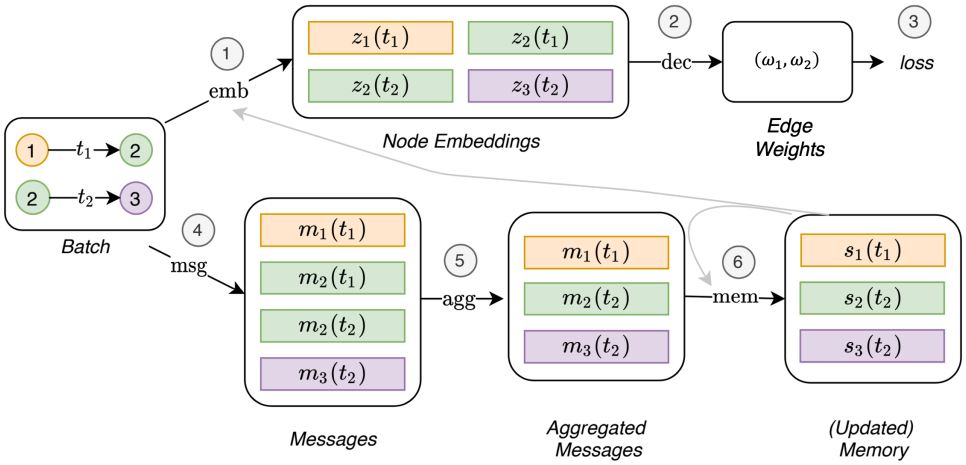

Figure 5.4: The architecture of TGN.

Results

TGN and the decoder are trained end-to-end with loss $||Y - \hat{Y}||_2^2$. Interestingly, training with negative sampling proved to be the most effective technique. Negative sampling involves randomly selecting one negative edge for each positive edge during training. [20] Negative edges are those that do not exist in the original graph and have a weight of 0. Also, note that the authors do not add additional node and edge features.

Jiang et al. compare TGN's performance against several forecasting techniques, GCN, and two other DGNNs (JODIE and DyRep). TGN consistently outperformed the DGNNs, surpassed the GCN thanks to the added temporal dimension, and achieved the second-lowest Mean Squared Error (MSE) in all but one instance, where it was the best. These findings validate TGN as an effective DGNN architecture. Strikingly, Persistence Average, a simple technique, almost always the best model. This highlights how TGN may not be sophisticated enough to perform well on temporal edge regression.

Conclusions

On the Commodity Trade Problem

The results from the previous chapter suggest that the TGN model is a step in the right direction for solving the commodity trade problem. However, they also highlight a clear research gap, as one baseline forecasting method outperformed the GNNs in all but one test condition. These results suggest that current Dynamic Graph Neural Network models may not yet be capable of solving temporal edge regression.

New training techniques may be necessary. The authors of [20] found that sampling negative edges in different proportions significantly affects test performance. Additionally, future research could explore edge-based embedding methods, such as Edge2Vec, which maps edges into structure-preserving embeddings. [48]

On Dynamic Graph Neural Networks

Throughout this thesis, we consolidated the foundations of Graph Neural Networks and explored their temporal applications. In particular, we began with graph representation learning, covered GNN theory, and detailed the main variations. Then, we introduced the dimension of time, reviewed sequence models, examined emerging Dynamic Graph Neural Networks architectures, and finally applied our knowledge by looking at two real-world uses of DGNNs.

The field of Graph Learning is still developing, and dynamic graph neural networks represent its frontier. This presents challenges, including a lack of benchmark datasets for model comparison and no clear development direction. This leads to researchers exploring large varieties of ideas in an uncoordinated manner. Zheng et al. (2024) suggest that current datasets are imbalanced, lack variety in events, and do not offer enough spatial and temporal complexity to effectively compare current dynamic models. [55]

Lastly, there is the challenge of domain diversity. Dynamic graphs are applied to a wide range of subjects due to their flexible structure. However, each application demands different graph characteristics, requiring specific models. This diversity makes it difficult to establish a standard framework. TGN is currently the sole unified framework and, although constrained to specific types of graphs, its superior performance compared to other DGNNs highlights the value of researching unified and innovative architectures.

There are multiple research opportunities within the field of Dynamic Graph Representation Learning. For instance, federated learning on graphs remains unexplored but could be beneficial given the various networks constructed with personal data. Another example is graph-level anomaly detection, which has applications in chemistry, such as recognizing special proteins, and in finance, for identifying fraudulent profiles. [52]

On a more theoretical level, the distinct nature of graphs encourages the research of new machine learning approaches. For instance, [20] note that simple graph-based normalisation tricks improve performance. Additionally, both this thesis and the broader research field focus on the encoder component of the models. It is plausible that the Multi-layer Perceptron lacks expressivity and looses information when decoding the node representations into the predictions.

Finally, there is the broader question of training complex models on intricate relational data using loss, a one-dimensional channel of information that tells the model how it needs to learn. While, this approach yielded models with human-level performance on tasks with images and text, which are standard-sized inputs, graphs lack this structure. The beauty of advanced fields like Dynamic Graph Neural Networks lies in their ability to make us reconsider the most basic foundations of our knowledge.

Bibliography

[1] S. Akansha. Over-squashing in graph neural networks: A comprehensive survey. *ArXiv. /abs/2308.15568*, 2023.

[2] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[3] Jiandong Bai, Jiawei Zhu, Yujiao Song, Ling Zhao, Zhixiang Hou, Ronghua Du, and Haifeng Li. A3t-gcn: Attention temporal graph convolutional network for traffic forecasting. *ISPRS International Journal of Geo-Information*, 10, 2021.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35, 2013.

[5] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *ICLR 2022 - 10th International Conference on Learning Representations*, 2022.

[6] Michael Bronstein. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *ArXiv/abs/2104.13478*, 2021.

[7] W. Cai, K. Wang, H. Wu, X. Chen, and Y. Wu. Forecastgrapher: Redefining multivariate time series forecasting with graph neural networks. *ArXiv. /abs/2405.18036*, 2024.

[8] Victor Carbune, Pedro Gonnet, Thomas Deselaers, Henry A. Rowley, Alexander Daryin, Marcos Calvo, Li Lun Wang, Daniel Keysers, Sandro Feuz, and Philippe Gervais. Fast multi-language lstm-based online handwriting recognition. *International Journal on Document Analysis and Recognition*, 23, 2020.

[9] Jinyin Chen, Xueke Wang, and Xuanheng Xu. Gc-lstm: graph convolution embedded lstm for dynamic network link prediction. *Applied Intelligence*, 52, 2022.

[10] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 2014.

[11] Bengio Yoshua Courville Aaron Goodfellow Ian. *Deep Learning - Ian Goodfellow, Yoshua Bengio, Aaron Courville - Google Books*. MIT Press, 2016.

[12] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *34th International Conference on Machine Learning, ICML 2017*, 3, 2017.

[13] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June, 2019.

[14] A. Gravina and D. Bacciu. Deep learning for dynamic graphs: Models and benchmarks. *IEEE, ArXiv*, 2023.

[15] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-August-2016, 2016.

[16] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14, 2020.

[17] Katja Hansen, Franziska Biegler, Raghunathan Ramakrishnan, Wiktor Pronobis, O. Anatole Von Lilienfeld, Klaus Robert Müller, and Alexandre Tkatchenko. Machine learning predictions of molecular properties: Accurate many-body potentials and nonlocality in chemical space. *Journal of Physical Chemistry Letters*, 6, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December, 2016.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997.

[20] Lekang Jiang, Caiqi Zhang, Farimah Poursafaei, and Shenyang Huang. Towards temporal edge regression: A case study on agriculture trade between nations. *arXiv*, 2023.

[21] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

[22] Thomas N Kipf and Max Welling. Variational graph auto-encoders 1 a latent variable model for graph-structured data. *NIPS workshop*, 2016.

[23] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

[24] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[25] Yan LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[27] Bogdan Nica. A brief introduction to spectral graph theory. *A Brief Introduction to Spectral Graph Theory*, 2018.

[28] Veličković P., Cucurull G., Casanova A., Romero A., Liò P., and Y. Bengio. Graph attention networks. *ArXiv. /abs/1710.10903*, 2017.

[29] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. Transfer graph neural networks for pandemic forecasting. *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, 6A, 2021.

[30] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson. Evolvegcn:

Evolving graph convolutional networks for dynamic graphs. *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, 2020.

[31] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*, 2013.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[33] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv 2006.10637*, 2020.

[34] Ryan A. Rossi, Di Jin, Sungchul Kim, Nesreen K. Ahmed, Danai Koutra, and John Boaz Lee. On proximity and structural role-based embeddings in networks: Misconceptions, techniques, and applications. *ACM Transactions on Knowledge Discovery from Data*, 14, 2020.

[35] T. K. Rusch, M. M. Bronstein, and S. Mishra. A survey on oversmoothing in graph neural networks. *ArXiv. /abs/2303.10993*, 2023.

[36] Aarush Saxena. An introduction to convolutional neural networks. *International Journal for Research in Applied Science and Engineering Technology*, 10, 2022.

[37] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv*, 2019.

[38] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45, 1997.

[39] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *IJCAI International Joint Conference on Artificial Intelligence*, 2021.

[40] Amauri H. Souza, Diego Mesquita, Samuel Kaski, and Vikas Garg. Provably expressive temporal graph networks. *Advances in Neural Information Processing Systems*, 35, 2022.

[41] Jonathan M. Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M. Donghia, Craig R. MacNair, Shawn French, Lindsey A. Carfrae, Zohar Bloom-Ackerman, Victoria M. Tran, Anush Chiappino-Pepe, Ahmed H. Badran, Ian W. Andrews, Emma J. Chory, George M. Church, Eric D. Brown, Tommi S. Jaakkola, Regina Barzilay, and James J. Collins. A deep learning approach to antibiotic discovery. *Cell*, 180, 2020.

[42] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4, 2014.

[43] A. Taheri, K. Gimpel, and T. Berger-Wolf. Learning to represent the evolution of dynamic graphs with recurrent models. *Companion Proceedings of The 2019 World Wide Web Conference, pp. 301-307*, 2019.

[44] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290, 2000.

[45] Jake Topping, Francesco Di Giovanni, Benjamin P. Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *ICLR 2022 - 10th International Conference on Learning Representations*, 2022.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-December, 2017.

[47] Petar Veličković, William Fedus, William L. Hamilton, Yoshua Bengio, Pietro Liò, and R. Devon Hjelm. Deep graph infomax. *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[48] Changping Wang, Chaokun Wang, Zheng Wang, Xiaojun Ye, and Philip S. Yu. Edge2vec: Edge-based social network embedding. *ACM Trans. Knowl. Discov. Data 14, 4, Article 45*, 2020.

[49] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Xiaojie Guo. Graph neural networks: Foundation, frontiers and applications. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2023.

[50] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[51] Keyulu Xu, Stefanie Jegelka, Weihua Hu, and Jure Leskovec. How powerful are graph neural networks? *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[52] Zhenyu Yang, Ge Zhang, Jia Wu, Jian Yang, Quan Z. Sheng, Shan Xue, Chuan Zhou, Charu Aggarwal, Hao Peng, Wenbin Hu, Edwin Hancock, and Pietro Liò. State of the art and potentialities of graph-level learning. *2301.05860 arXiv*, 2023.

[53] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, Tao Lin, M. Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21, 2020.

[54] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[55] Y. Zheng, Lu Yi, and Z. Wei. A survey on dynamic graph neural networks. *Higher Education Press 2024, Gaoling School of Artificial Intelligence*, 2024.