UNIVERSITY OF THE PELOPONNESE & NCSR “DEMOCRITOS”
MSC PROGRAMME IN DATA SCIENCE

Combining and comparing hierarchical attention and composition-based GNNs for Knowledge Graph completion

Fotios Papadimas

A thesis submitted in partial fulfillment of the requirements for the MSc in Data Science

Supervisor: Anastasia Krithara
Associate Researcher at IIT of NCSR "Demokritos" in Athens,
Co-supervisors: Fotis Aisopos
Post-Doctoral Researcher

Athens, July 2024

Thesis Title: Combining and comparing hierarchical attention
and composition-based GNNs for Knowledge Graph completionFotios Papadimas
MSc. Thesis, MSc. Programme in Data Science
University of the Peloponnese & NCSR “Democritos”, July 2024

UNIVERSITY OF THE PELOPONNESE & NCSR "DEMOCRITOS"
MSC PROGRAMME IN DATA SCIENCE

Declaration of Authorship

(1) I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

(2) I confirm that this thesis presented for the degree of Bachelor of Science in Informatics and Telecommunications, has

(i) been composed entirely by myself

(ii) been solely the result of my own work

(iii) not been submitted for any other degree or professional qualification

(3) I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or processional qualification except as specified.

(Signature)

Fotios Papadimas

Athens, July 2024

Acknowledgments

I would like to express my deepest gratitude to my advisors, Anastasia Krithara and Foris Aisopos, for their invaluable guidance, support, and encouragement throughout the course of my research and the writing of this thesis.

Dr. Fotis Aisopos, your expertise and insightful feedback have been instrumental in shaping my research. Your unwavering dedication to my progress and your thoughtful critiques have pushed me to achieve more than I thought possible. I am deeply appreciative of the time you have invested in reviewing my work and providing detailed suggestions that have significantly improved the quality of this thesis.

I would also like to extend my heartfelt thanks to Anastasia Krithara for her invaluable guidance and support throughout this project. Her expertise, insightful feedback, and constant encouragement have been instrumental in the successful completion of this work. I am deeply grateful for her time and dedication.

Finally, I would like to thank Dr. Konstantinos Bougiatiotis for his valuable insights. His support and guidance have been greatly appreciated.

Thank you for believing in me and for your unwavering support throughout this process.

To my family.

Abstract

Graph Convolutional Networks (GCNs) have enabled the application of deep learning methods to large graphs. These models create an embedding representation for each node in the graph, and we train the model on these embeddings. The trained model can then be used to predict links between nodes or classify them. Link prediction, for instance, can be applied to biomedical graphs for tasks such as drug repurposing. By improving the performance of GCNs, we can enhance their application in drug repurposing. In this thesis, we aim to improve GCN results by enriching the representation of each node in the graph using two-hop paths for each relation.

Table of contents

Table of Figures

Table of tables

Table of Abbreviations

GCN	Graph convolutional network
R-GCN	Relational graph convolutional network
GAT	Graph attention network
CompGCN	Composition based GCN
GNN	Graph Neural Network
CNN	Convolutional Neural Network
HARPA	Hierarchical attention with relation paths for knowledge graph embedding adversarial learning
MRR	Mean reciprocal rank
BLGPA	Biomedical Literature Graph Path Analysis

Chapter 1 : Introduction

Graphs are a powerful tool for representing objects and the relationships between them. A common example is a social network, where the objects are users and the relationships are the friendships between them. Another example is in the biomedical domain, where knowledge graphs represent drugs, proteins, and diseases, with relationships denoting drug-protein interactions, disease-drug interactions, and so on. Machine learning can be applied to graphs for various tasks such as node classification, link prediction, clustering/community detection, and graph classification. Traditional machine learning methods for graphs relied on manually crafted feature vectors. However, deep learning and neural networks can create better node embeddings by generating feature vectors in the hidden layers of the networks, achieving superior representations compared to manual ones (Hamilton, 2020). Graph Neural Networks (GNNs), a subtype of neural networks, are specifically designed for these tasks. Improving GNN predictions could enhance their application to real-world datasets, such as biomedical graphs, potentially uncovering unknown relationships between drugs and diseases.

1.1. What is a knowledge graph

There are various definitions of what constitutes a knowledge graph (Ehrlinger and Wöß, 2016). According to (Cimiano and Paulheim, 2016), a knowledge graph (i) primarily describes real-world entities and their interrelations, organized in a graph, (ii) defines possible classes and relations of entities within a schema, (iii) allows for the potential interrelation of arbitrary entities, and (iv) covers various topical domains. Similar views are expressed by (Pujara *et al.*, 2013), who emphasize that the data should not only have structure but also support an automatic knowledge extraction system. (Hogan *et al.*, 2021) describe knowledge graphs as graphs of data designed to accumulate and convey knowledge of the real world, where nodes represent entities of interest and edges represent the relations between these entities. This knowledge can come from external sources or be deduced from existing knowledge in the graph. Simple statements and quantified statements can both be represented in the graph; simple statements are represented as edges, while quantified statements typically require ontologies or rules.

Because knowledge graphs are often derived from real-world data, such as biomedical data, they are frequently incomplete. For instance, if the nodes in a graph represent drugs, proteins, and diseases, unknown relationships between them may not be included. Predicting these missing links is crucial, as it can aid in tasks like drug repurposing. This challenge is known as the knowledge graph completion problem.

The knowledge graph can be more formally defined as a set of triplets $\{(h, r, t)\} \subseteq E\ x\ R\ x\ E$. E is the set of nodes and R the set of relations between the nodes. Based on two existing entities we search for the third one. Either $(h, r, \#)$ or $(h, \#, t)$. In the first we predict the tail of the triplet and in the second the missing link between two nodes.

1.2. Knowledge graph refinement methods

Knowledge graphs are constructed from real-world data. The automated methods used to collect this data, such as NLP and others, are prone to errors, leading to incomplete graphs or incorrect entries and relations. Knowledge graph refinement methods aim to address these issues. The two main categories of these methods are completion and error detection (Cimiano and Paulheim, 2016). These methods can generally be categorized based on their use of data: whether they use external data to correct the graph or not. Methods that use external data are called external methods, while those that do not are called internal methods. Further categorization can be based on what aspect of the graph they aim to correct, such as nodes, relations, or data types.

- Knowledge graph completion

  The aim here is to increase the coverage of the knowledge graph by predicting missing elements, such as relations, nodes, node types, and the relationships that hold between nodes (Cimiano and Paulheim, 2016).

- Knowledge graph error detection

  In knowledge graph completion, we target issues such as incorrect type assertions, relations, and Intralinks within the knowledge graph. (Cimiano and Paulheim, 2016).

Graph Neural Networks (GNNs) were created to perform this work. The first GNN was applied to graphs where labels for some of the nodes were known, and the task was to label the unknown nodes(Kipf and Welling, 2016).

1.3. Classical methods for graph completion

For the graph completion problem, two main categories of methods are used: rule-based methods and embedding methods.

1.3.1. Rule based methods

This category of methods mines rules from the graphs and predicts relations between nodes based on these rules.

1.3.1.1. AnyBURL

In this method the correct paths between nodes are generalized into rules that can be used to infer missing links in other cases (Meilicke *et al.*, 2020). The algorithm learns the rules and then ranks them according to an estimated level of confidence.

1.3.1.2. SemaTyP

This method, employed on the biomedical knowledge graph SemKG, aims to detect (drug–target-disease) triples and recommend appropriate drugs for treating diseases, leveraging the scores associated with these triples. SemaTyP operates by traversing paths of

length l originating from the drug node, passing through the target node, and terminating at the disease node. Features are derived by aggregating the semantic types of the subject/object nodes and the semantic types of the relations encountered along the path. These features are then inputted into a logistic regression classifier to discern the relevant paths (Sang *et al.*, 2018).

1.3.1.3. Biomedical Literature Graph Path Analysis (BLGPA)

This is an algorithm that is used in biomedical knowledge graphs(Aisopos and Paliouras, 2023). The idea here is for every path of length l to create an embedding that captures the semantics of each path as well as how often chains of relations appear in the paths. First we define a path between node starting node d and finish node t. The path of length l passing through nodes v_i and relations r_i between the nodes d, t is $(d, v_1, r_1 \dots \dots \dots .. v_{l-1}, r_{l-1}, t)$. There can be multiple paths of the same length between the two nodes they are represented as π_i^l and all the paths create the set $\Pi_{d,t} = \{\pi_1^l, \pi_2^l \dots . \pi_{N_{(d,t)}}^l\}$.

To capture the semantic information of each path for each pair v_i, r_i in the path a vector embedding of length 162 is created. The 127 dimensions represent the different available semantic types for the nodes and the remaining 35 dimensions represent the available semantic types for the relations. The graph that the methods was used on was created from MetaMap. The semantic types of the nodes and the relations are not uniquely defined there. Each paper gave different semantic types to each node or relation. In the embeddings the method creates basically the number of times each relation type or node type is mentioned in the literature is written in the corresponding dimension of the embedding. As such for a path of length l a vector of length l x 162 is created by concating the different embeddings of each pair of nodes and relations in the path. Then the aggregate of these embeddings for the different paths of length l is used to represent the path.

The other half of the procedure focuses on the sequence of relations between nodes d and t in the paths of length l. We search for the M most important paths. Then we create a vector that has length equal to the number of paths which is M. The entries in this new vector is the number each of these M paths appears in $\Pi_{d,t}$.

Finally, to capture both types of information the two vectors are concatenated.

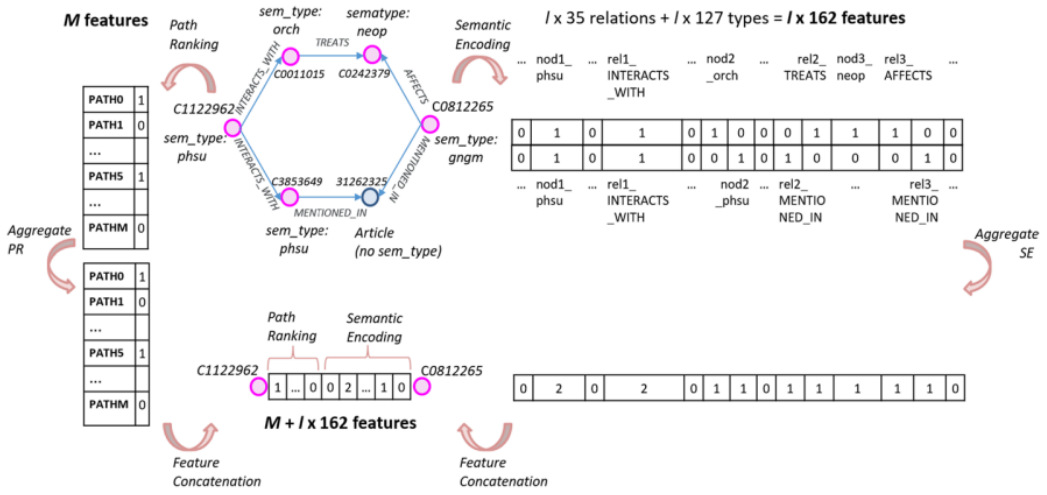


Figure 1-1 BLGPA from (Aisopos and Paliouras, 2023)

1.3.2. Embedding methods

These methods rely on an encoder-decoder architecture. The encoder assimilates information from the graph into an embedding vector, while the decoder predicts the presence and type of relation between nodes. Contemporary approaches often employ Graph Neural Networks (GNNs) for encoding. In the context of graph completion tasks, decoders leverage vector embeddings to compute the likelihood of a relation existing between nodes (Zamini, Reza and Rabiei, 2022). Here are some of the most prevalent decoders commonly encountered in literature for link prediction:

1.3.2.1. Translational Models

In this framework, relations are conceived as translation operators acting upon vector embeddings of nodes. The objective is to learn translations of the form h + r = t,

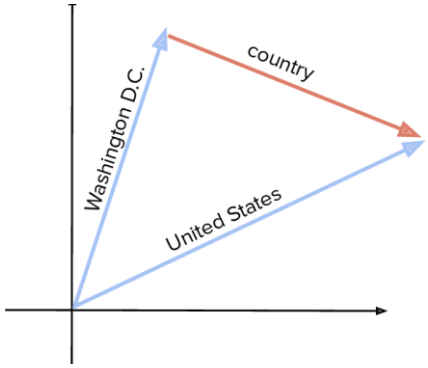


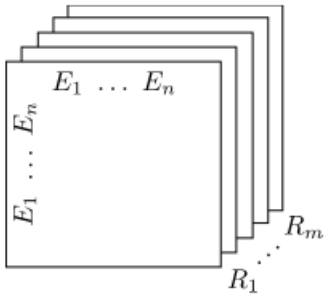
Figure 1-2 TransE representation from (Bianchi et al.*, 2020) the embedding of head node(h): Washington D.C plus the embedding of the relation(r): country equals the embedding of the node(t): United states. TransE learns h+r = t*

where h represents the head node embedding, r the relation embedding, and t the tail node embedding. The embeddings are iteratively updated to satisfy this relation. A scoring function assigns a score to each triplet, and an optimization algorithm refines the vector embeddings based on these scores. TransE (Bordes *et al.*, no date) stands as the foremost translational model, but it struggles when multiple relationships exist between nodes in a

one-to-many or many-to-many fashion. To address this limitation, other models such as TransH (Wang *et al.*, 2014) and RotatE (Sun *et al.*, 2019) have been introduced, employing different operators to represent relations and enhancing the performance of TransE.

1.3.2.2. Tensor Dompositional Models

These models try to model the interactions as tensor products. They can be used without the need of an encoder by just randomly initializing the embeddings of each node. This class of models has models like RESCAL(Nickel, Tresp and Kriegel, 2011) ,DistMult(Yang *et al.*, no date) and ComplEx (Trouillon *et al.*, 2016). From these three the most used is DistMult. RESCAL models the nodes as vectors and the relations as asymmetric n x n matrices. It calculates a score as $score = head^T * Relation_{matrix} * tail$ for each triplet



which scores how likely it is for the relation to exist.

Figure 1-3 E1,..,En are nodes and R1,..,Rm are the relation matrices from (Nickel, Tresp and Kriegel, 2011)

Again, here the vectors embeddings and relation matrices are optimized with an optimization algorithm. This method is prone to overfitting due to large number of parameters which is $o(d^2)$ per relation. DistMult fixes this by using only diagonal matrices which get the number of parameters to $o(d)$. However, it can't handle asymmetric relations. It can be used as a shallow model in large data bases where the RESCAL could not. ComplEx uses complex valued vector representations for the nodes and complex diagonal matrices for the relations. This allows the model to capture asymmetric relations.

1.3.2.3. Convolutional-Based Mode

These models utilize convolutions and projections to determine the existence of a link between two nodes, offering reduced parameterization while maintaining comparable

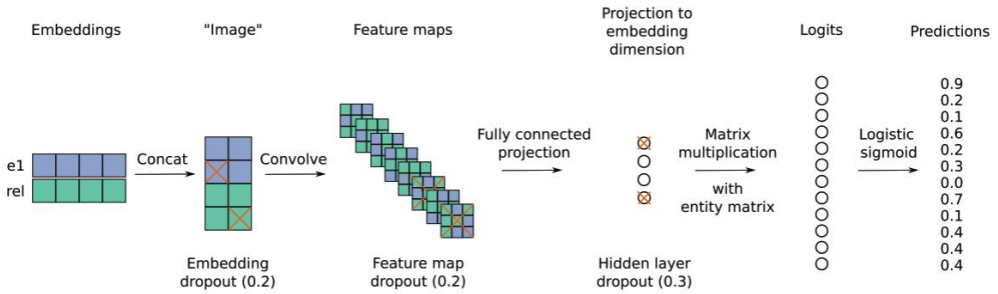


Figure 1-4 ConvE process from (Dettmers et al., 2017)

performance to DistMult. They can operate independently without the need for an encoder. The first of these models is ConvE (Dettmers *et al.*, 2017). In ConvE, 2D convolution layers are employed for link prediction. The algorithm comprises a convolution layer, a projection layer, and an inner product layer.

In Figure 1-4 ConvE process from (Dettmers et al., 2017)we see how ConvE works. In this algorithm, the embedding of the node e1 and the relation rel are concatenated, forming an image-like structure subjected to convolution following dropout. The resulting feature maps generate a tensor, which is then reshaped into a vector. This vector undergoes linear transformation to match the embedding dimension. Subsequently, this final vector is multiplied with the entity matrix, an n x embedding dimension matrix where n represents the number of entities (nodes). The resulting vector passes through a sigmoid function, yielding the probability of each node being connected to node e1 with relation rel. Similar to other models, embeddings are refined based on scores using an optimization function.

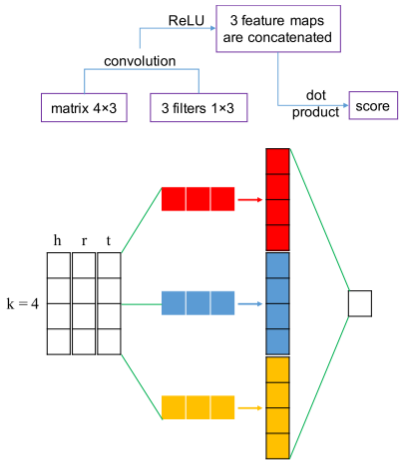


Figure 1-5 ConvKB process from (Nguyen et al., 2017)

In Figure 1-5 ConvKB process from (Nguyen et al., 2017) we see the process for ConvKB(Nguyen *et al.*, 2017) here a score for each triplet is created and the goal is to have minimum score for real triplets. Here no reshaping is done to the original embedding dimension.

In ConEx(Demir and Ngomo, 2020) first a Hadamard product composition of a 2D convolution takes place, then an affine transformation and then a Hermitian inner product take places. The embeddings used are complex-valued. Here the asymmetric properties of Hermitian products and the parameter sharing property of a 2D convolution are utilized.

1.4. Encoding using Graph Neural Networks (GNN's)

Graph Neural Networks (GNN's) create the vector embeddings. Due to the nature of graphs and that they can be visually represented in different but equivalent ways methods such as CNN's cannot be used to make embeddings. The same graph could be represented in a lot of different ways by permuting the nodes making predictions unreliable. Another problem would be the variable size of each neighborhood of each node due to the different connecting nodes.

With GNN's in a Graph G=(V, E) where V are the nodes and E the edges, we want to update the embedding of each node as an operation on the embeddings of the neighbored nodes and relations between them. If we define (N_u) as the neighborhood of the node u that belongs to V, x_u the embedding of node u and e_{uv} the edge the embedding of the relation of the nodes(u,v) we can define the most general form of a GNN which is a message passing layer:

$$h_u = \varphi(x_u, \oplus_{u \in N_u} \psi(x_u, x_v, e_{uv}))$$

Here Φ,ψ are differentiable functions,⊕ a permutation invariant aggregation operator and the new representation of the node is h_u. Multiple message passing layers can be stacked to create better representations for h_u (Bronstein *et al.*, 2021)

1.4.1. What we can predict with GNN's

GNN's embeddings can be used for node classification or clustering. If we use GNN's as encoders in the encoder – decoder model described before they can also be used for relation prediction. They can be used along decoders such as transE, convE etc. for this task. The better node embedding achieved by GNN's improves the results compared to randomly initializing the embeddings before passing them from the decoders. (Bronstein *et al.*, 2021)

1.4.2. Where they can be applied (biomedical graphs etc)

They can be used in social networks, knowledge graphs natural language processing, drug discovery, crystal property prediction amongst other things. We are mostly interested in their use in biomedical graphs.

1.5. Research goals

In the course of this thesis, our primary focus was on surveying the extensive body of literature to identify the most promising Graph Convolutional Network (GCN) methodologies. Upon meticulous examination, we pinpointed the most optimal GCN model accessible with openly available source code. Subsequently, we proceeded to incorporate the HARPA path embedding technique into this selected GCN framework.

Our overarching objective revolved around attempting to replicate the performance enhancements detailed in the HARPA paper within the chosen GCN model. This entailed rigorous experimentation and analysis to assess the efficacy of integrating the HARPA method in improving the model's performance.

Furthermore, we established a secondary objective aimed at investigating the various implementations of the path embedding method. Through systematic evaluation, we sought to ascertain which implementation yielded the most favorable outcomes, thereby contributing valuable insights into the practical application of path embedding techniques within GCN architectures.

1.6. Structure of thesis

The current dissertation is split in 6 chapters.

Chapter 1 serves as the introduction to the thesis, providing a comprehensive overview of knowledge graphs. Definitions sourced from existing literature elucidate the concept of knowledge graphs, followed by an exploration of common methods employed for link prediction within these graphs. The fundamental encoding-decoding structure of models is elucidated, with a focus on widely used decoders like TransE and ConvE. Additionally, a foundational understanding of Graph Neural Networks (GNNs), which are pivotal for encoding knowledge graph information, is presented.

Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε. delves into the theoretical underpinnings of the encoders utilized in the experiments. Starting from the original Graph Convolutional Network (GCN), the chapter navigates through the evolutionary trajectory of GCN-based methodologies, culminating in advanced techniques like R-GCN, COMPGCN, and HARPA.

Chapter 3 outlines the methodology adopted for the thesis

Chapter 4 presents the results derived from the conducted experiments, providing empirical insights into the performance and efficacy of the implemented methodologies. We also engage in a thorough discussion of the findings obtained throughout the research, offering critical analysis and interpretation of the results.

Chapter 5 encapsulates the work undertaken in the thesis, summarizing the key findings, discussing their implications, and proposing avenues for future research.

Chapter 2 Graph convolution networks (GCN's)

Graph convolutional networks (GCN's) were introduced in (Kipf and Welling, 2016). It offers a method for (semi)-supervised node classification. It set the theoretical background by utilizing spectral methods for message propagation on the graph. Analogously to the convolution on images, the node embeddings for each node are the pixel values. Because for the nodes it is computationally hard to compute the convolution an approximation is used instead. We use the approximation to convolve over the graph and learn node embeddings that capture the finer structure of the graph. This is in the same way finer details are learned about images in regular CNN's. These new embeddings are used for tasks such as node classification or link prediction.

2.1.1. GCN propagation rule

The original Graph convolutional network was created for undirected graphs. Vector representation for each node of the graphs' called embeddings are created. The node representations are passed onto layers of our GCN network to learn deep representations for each node.

First, we initialize the node embeddings. Because the nodes are different and we want to express this difference in their embeddings, we randomly initialize the embeddings of the nodes. In a GCN layer we learn embeddings of the form $H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$.The subscript l denotes the layer. The embeddings of all the nodes are stacked in a matrix $H^{(l)}$. $H^{(l)}$ are the node embeddings that are fed into the layer l. $H^{(l+1)}$ are the node embeddings produced by the GCN layer l. Because the matrix $W^{(l)}$ acts on $H^{(l)}$, $H^{(l+1)}$ can be of different embedding dimension. Sigma is defined as an activation function such as relu. $\widetilde{A}$ is the adjacency matrix with added self-connections We define $\widetilde{A}\ as\ \widetilde{A} = A + I_N$ where A is the adjacency matrix of the graph and I_N the identity matrix. $\widetilde{D}^{-\frac{1}{2}}$ are the inverses of the degree matrix. As such the $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ is the normalized Laplacian L of the graph (Kipf and Welling, 2016).

Below we go through the message passing of a layer in a GCN.

In **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** the first step takes place. The node embeddings are multiplied with the learnable matrix $W^{(l)}$ and projected in a new vector space.

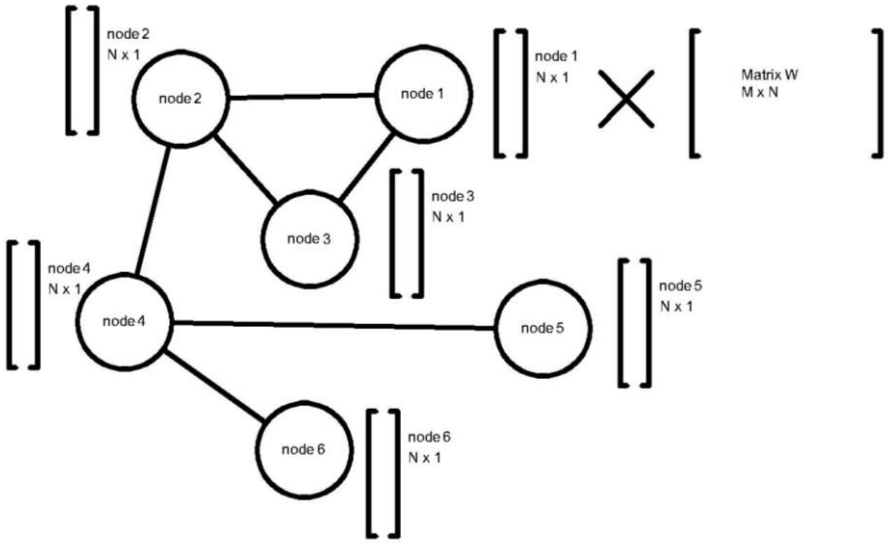
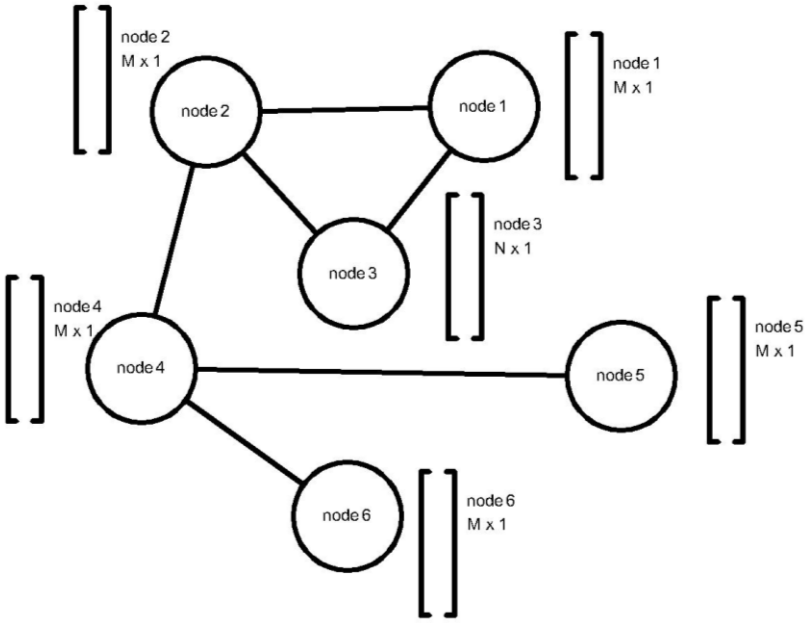


Figure 2-1 Node embeddings of a network before projection

After $W^{(l)}$ the embedding dimensions are like in **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**At this point we have



created $H^{(l)}W^{(l)}$.

Figure 2-2 Node embeddings after projected in a new vector space

Then as a middle step when computing $H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$, we multiply $H^{(l)}W^{(l)}$ with the normalized Laplacian $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$. The normalized Laplacian purpose is to inject the graph structure in the computation. What the normalized Laplacian does is for each node to use the immediate neighbor nodes to update its embedding. When the matrix $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ $acts\ on\ H^{(l)}W^{(l)}$ the embeddings of each node are summed with its

neighbors, which have been multiplied with some constant from $\widetilde{D}$. This is represented in **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**.

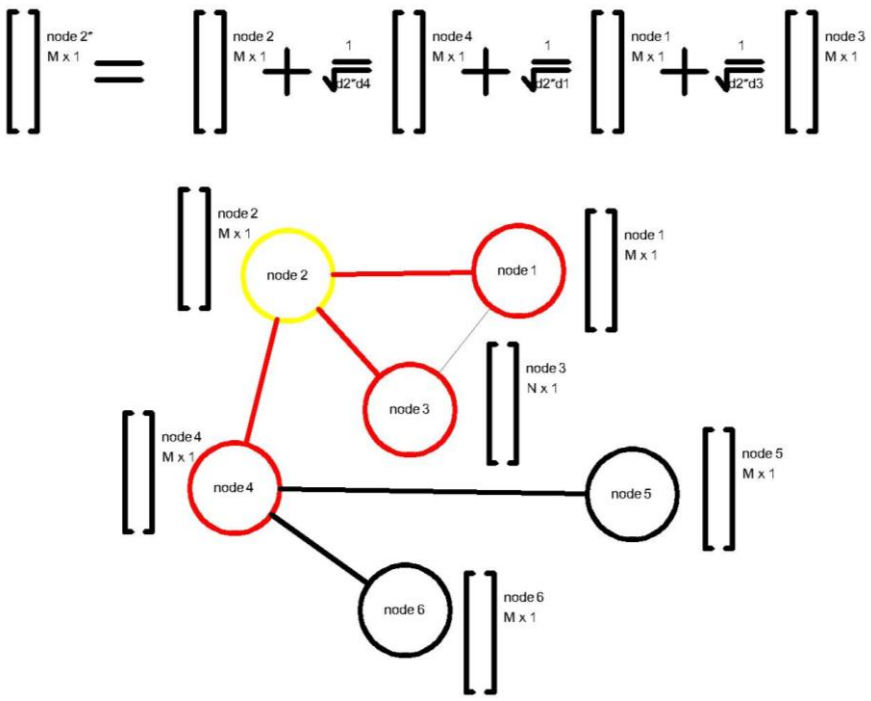


Figure 2-3 Updating node2 embedding from the surrounding nodes

At this point we pass the new embeddings from an activation function σ and get our new embedding representation for the nodes, this is represented in the matrix $H^{(l+1)}$. The node embeddings are now aware of their immediate neighbors. We can pass these new embeddings from a new GCN layer to make the embeddings aware of nodes even further away

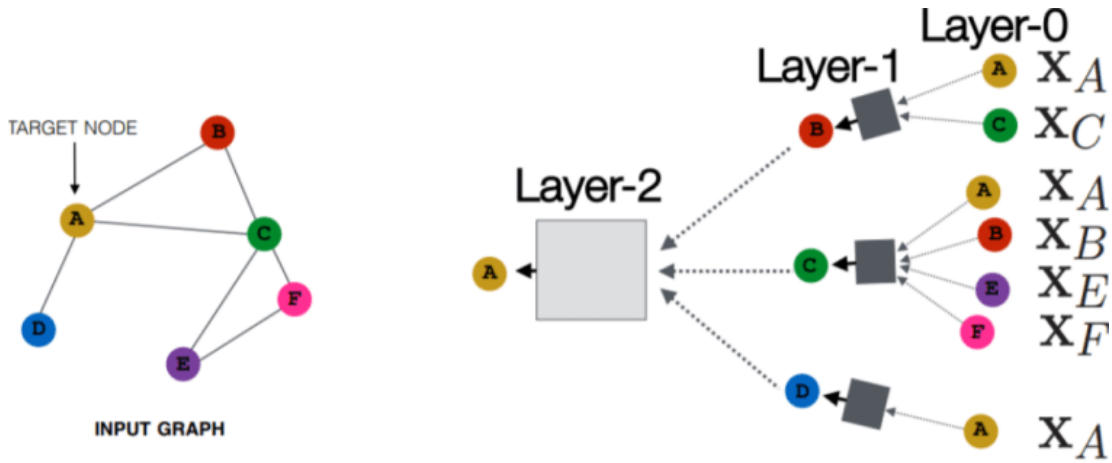


Figure 2-4 from (Hamilton, 2020)

2.1.2. Theoretical formulation

The GCN propagation rule is motivated from spectral filters on graphs. Here a spectral convolution on the graph is used. We are interested in the convolution of signal $x \in R^N$ and filter g_θ. We represent with N the number of nodes. Signal x is composed from a scalar for each node (Kipf and Welling, 2016).

The convolution $g_\theta \star x$ can be computed as the inverse fourier transform of the multiplication of x and g_θ in the Fourier space. The filter $g_\theta = diag(\theta)$ where $\theta \in R^N$ in the Fourier space. The base for the Fourier transform will be the one that form the eigenvectors of the normalized graph Laplacian defined as $L = I_N - D^{-1/2}AD^{-1/2}$. The normalized graph Laplacian's eigenvectors are represented in the matrices U. The diagonal matrix that has the Laplacians eigenvalues is defined as Λ. For the fourier transform of x first, we project the signal x into the base created from the eigenvectors of the normalized graph Laplacian. We project the signal x in the basis formed by the eigenvectors as per the Fourier transform as $U^T x$ (Kipf and Welling, 2016).

Then we multiply with the filter g_θ, we get $g_\theta U^T x$. Then to get the convolution in the original space we do the inverse Fourier transform$U g_\theta U^T x$ and get the convolution $g_\theta \star x = U g_\theta U^T x$ (Kipf and Welling, 2016).

The filter $g_\theta$is diagonal and can be thought of as function of Λ which are the eigenvalues of L. With this in mind there exists an approximate solution for $g_\theta(\Lambda)$ which is : $g_\theta(\Lambda) = \sum_{k=0}^{K} \theta'_k T_\kappa(\tilde{\Lambda})$, where $\tilde{\Lambda} = \left(\frac{2}{\lambda_{max}}\right) * \Lambda - I_N$, $\theta' \in R^K$ is a vector of Chebyshev coefficients , λ_{max} is the maximum eigenvalue of Λ and T_κ are Chebyshev polynomials. As such the convolution can be simplified as $g_\theta \star x = \sum_{k=0}^{K} \theta'_\kappa T_\kappa(\tilde{L})x$ where we have defined $\tilde{L} = \left(\frac{2}{\lambda_{max}}\right) * L - I_N$. $T_\kappa(\tilde{L})$ is a matrix that is proportional to $\tilde{L}$ to the power of the order κ. The 0th order polynomial will be a scalar, the 1st analogous to $\tilde{L}$, the 2nd analogous to $\tilde{L}^2$ etc. Because $\tilde{L}$ is analogous to the adjacency matrix and the power of the adjacency matrix gives the number of k length paths between nodes we say that the approximation is k – steps localized (Kipf and Welling, 2016).

We create a neural network based on these convolutions. Each layer of the network is based on using only up to the K =1 order Chebyshev polynomials. So, it is a K =1 localized convolution. The filter g_θ is made from trainable parameters that our network will learn. Simplifying the expression so far we have $g_\theta \star x = \theta'_0 \chi + \theta'_1 (L - I_N)x = \theta'_0 \chi + \theta'_1 \left(D^{-1/2}AD^{-1/2}\right)x$ by also assuming that our network will learn λ_{max} =2 during the training and thus omitting it. If we further approximate $\theta'_0 = \theta'_1$ we get $g_\theta \star x = \theta\left(I_N + D^{-1/2}AD^{-1/2}\right)x$, This has eigenvalues in the range of [0,2] so to avoid exploding gradients we normalize once again and get to $g_\theta \star x = \theta(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2})$, where $\tilde{A} = A + I_N$, $\tilde{D}$ is the degree matrix of $\tilde{A}$. This is just for a scalar per node and one filter. If instead we had a vector per node(N nodes with C dimensional vector each) and F filters we would get the expression $Z = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X\Theta$,where $X \in R^{N \times C}$, $Z \in R^{N \times F}$, $\Theta \in R^{C \times F}$. We pass this through an activation function to give it the non-linearity needed. Now if we want to make embeddings that have information from 2 steps away, we create a second trainable layer and pass through it the embeddings from the first layer. These vectors can be passed through a logit function and with a cross entropy loss function be used to predict the class of a node (Kipf and Welling, 2016).

2.1.3. Advantages of GCN

One main advantage is that by not using the limiting forms of the Chebyshev polynomials and the neural network can learn better representations. The authors also

expect that the problem of overfitting in local neighborhood structures for graphs with very wide node degree distributions (Kipf and Welling, 2016).

2.2. Graph attention networks (GAT's)

Here attention between different nodes is measured. By doing this for each node and its one hop neighborhood we can create deep representations for each node.

2.2.1. Encoder

In GAT's we start by randomly initiating an embedding for each node. This is just to have a different representation for each node. As such we have the set of embeddings of the N nodes which is h = $\{h_1, h_2, \dots \dots . h_N\} \in R^F$. We project these embeddings in a new vector space by using a matrix W $\in R^{F' x F}$. We now have h'= $\{h'_1, h'_2, \dots \dots . h'_N\} \in R^{F'}$. On this new representation we perform self-attention. We compute a function that takes as inputs two vectors from h' and gives a score based on them. Because if we do this between all two nodes, we disregard the structure of the graph, we choose to compute the attention of each node with its one hop neighbors, which we represent as N_i. More specifically the score between two nodes i and j is $e_{ij} = a(Wh_i, Wh_j)$. The attention can be anything of the form $R^{F'} x R^{F'} \rightarrow R$. In GAT's this is usually done by concatenating $Wh_i \; and \; Wh_j$ and taking the inner product with a vector a of dimension $R^{2F'}$,and the passing the result from an activation function such as LeakyReLU. To make the coefficients easily comparable with other nodes they are normalized with a SoftMax function $a_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k \in N_i} \exp(e_{i,k})} = \frac{\exp(LeakyRELU[a^T(Wh_i||Wh_j])}{\sum_{k \in N_i} \exp(LeakyRELU[a^T(Wh_i||Wh_K]}$. The new embedding we create for node I becomes $h'_i = \sigma(\sum_{j \in N_i} a_{i,j} W h_j)$ where σ is an activation function. This is identically done for multiple heads to stabilize the attention mechanism. We then concatenate the results so the final embedding after a layer of a GAT becomes $h'_i = ||_{k=1}^{K} \sigma(\sum_{j \in N_i} a_{i,j}^k W^k h_j)$ where k denotes the attention head. In the final layer where concatenation makes no sense, we just instead average the results of each head $h'_i = \sigma\left((1/K) \sum_{k=1}^{K} (\sum_{j \in N_i} a_{i,j}^k W^k h_j)\right)$ (Veličković *et al.*, 2017) .

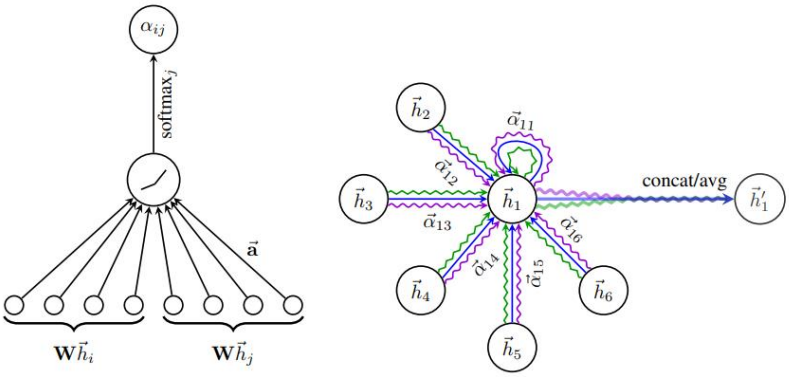


Figure 2-5 Attention calculation and multihead attention in the newtwork from (Veličković et al.*, 2017)*

2.2.2. Decoders

For link prediction the available decoders here are the ones we saw before ConvE, DistMult etc.

2.3. Relational graph convolutional network (R-GCN)

Here the authors used graph convolutional networks probably for the first time for link prediction and not just for node classification. They also created richer embeddings that consider the relationships between nodes. The model works as an autoencoder where the encoder part is the authors R-GCN embedding method for nodes and the decoder can be models like the DistMult.

2.3.1. Encoder

In this method the authors tweaked the original GCN method. The basic idea here is to use the GCN embedding per edge type.

The formula that describes the node embedding at each layer of the network is $h_i^{(l+1)} = \sigma(\sum_{r \in R} \sum_{j \in N_i^r} \left(\frac{1}{c_{i,r}}\right) W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)})$. The prefix l symbolizes the layer of R-GCN. As such $h_i^{(l)}$is the embedding of node i at layer l. R symbolizes the set of relation types of node i to its connecting neighbors. N_i^r is the set of the connecting neighbors to node i under relation type r. $1/c_{i,r}$ is a normalization constant which is either learned or chosen. For each node i and for each of its relation types r we select the connecting neighbors' embeddings $h_j^{(l)}$ and sum them. As such for node i for relation r we have $\sum_{j \in N_i^r} h_j^{(l)}$ Then we use matrix $W_r^{(l)}$ to project these new embeddings in a new vector space and we normalize them by $c_{i,r}$, we get a $\sum_{j \in N_i^r} (1/c_{i,r}) W_r^{(l)} h_j^{(l)}$. For our node i we do this process for each relation type and then sum these embeddings $\sum_{r \in R} \sum_{j \in N_i^r} (1/c_{i,r}) W_r^{(l)} h_j^{(l)}$. Then we add the embedding of node i projected in this new vector space $\sum_{r \in R} \sum_{j \in N_i^r} (1/c_{i,r}) W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}$. Finally we pass this new vector from an activation function to get the hidden representation of the node $h_i^{(l+1)} = \sigma(\sum_{r \in R} \sum_{j \in N_i^r} \left(\frac{1}{c_{i,r}}\right) W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)})$. This $h_i^{(l+1)}$embedding can be used as the node embeddings in the next layer of RGCN (Schlichtkrull *et al.*, no date).

The R-GCN process is presented in the following pictures. We see that for each edge type $r \in R$ the connecting nodes ($j \in N_i^r$) to our node are selected. (Schlichtkrull *et al.*, no date).

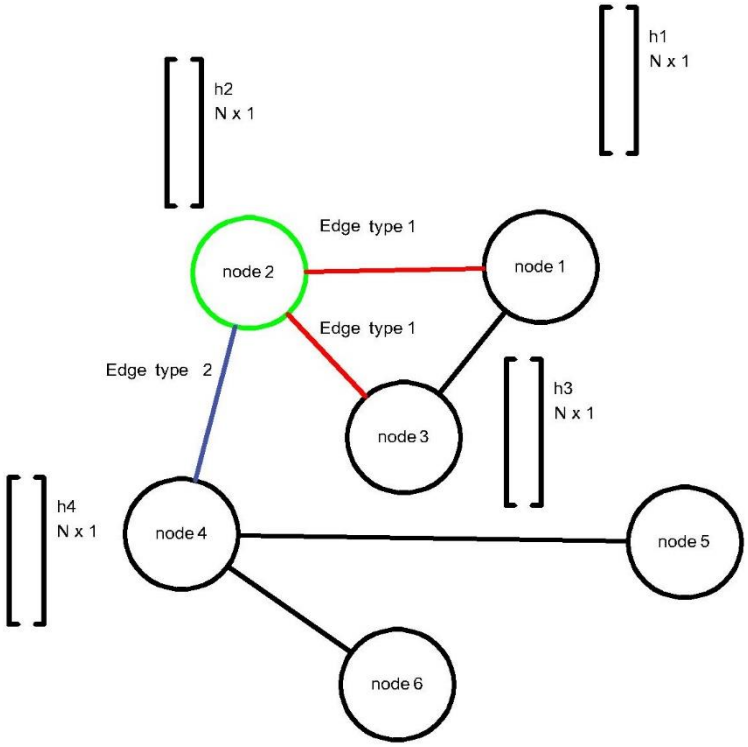


Figure 2-6 Nodes of a graph and their relations. Node 1 and Node 3 are connected to node 2 with the same edge type. Node 4 is connected to node 2 with a different edge type compared to the previous nodes. Near each node is illustrated its embedding as N x 1 vector

Their embeddings are summed and then transformed to a new vector space by matrix that is specific to each relation $W_r^{(l)}$.

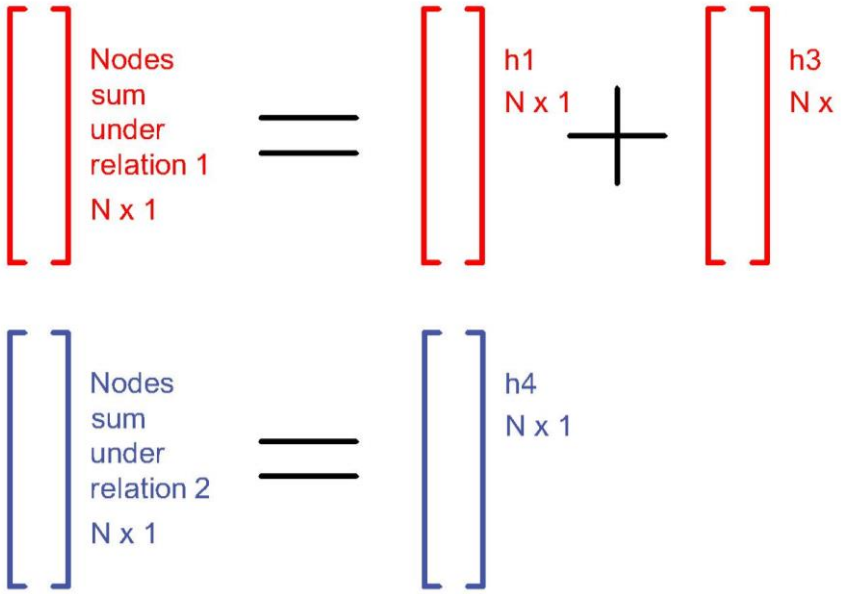


Figure 2-7 Summing of the node embedding per relation type in the example. The node we are updating is node 2.

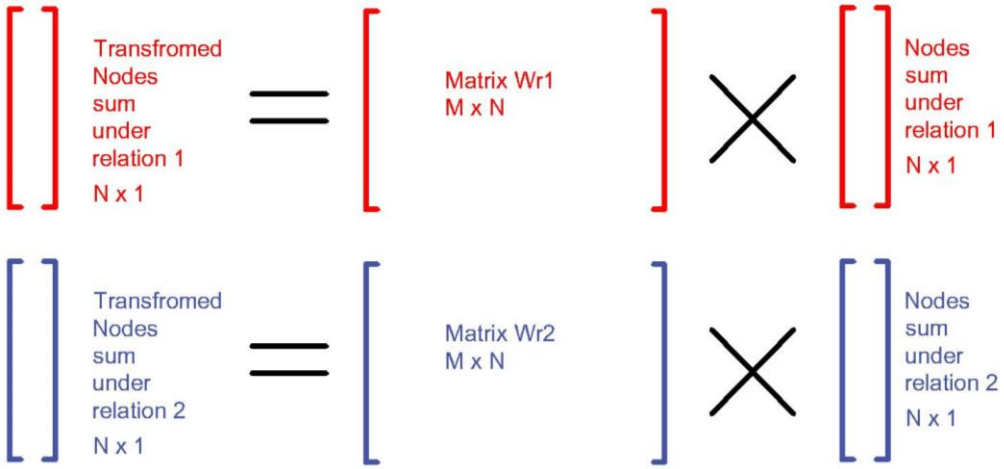


Figure 2-8 Transforming to a new vector space

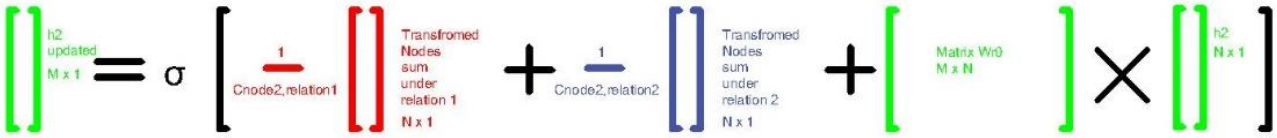


Figure 2-9 New embedding of node 2

Then the new embeddings that are created for each edge type are normalized with a parameter and summed again for all the different edge types. Then to this sum we add the projected embedding on the new vector space of our original node embedding.

Finally, after passing the sum from the non-linear function σ we get the updated node $h_i^{(l+1)}$. Then we can create new layers the same way as in GCN to get information from further regions of the graph in our embeddings.

A problem we encounter when using this method is that as we scale with the number of relations so do the matrices associated with each relation and their parameters. To alleviate this problem either we use basis matrices to write the others $W_r^{(l)} = \sum_{b=1}^{B} a_{rb}^{(l)} V_b^{(l)}$ or use a block-diagonal decomposition $W_r^{(l)} = \oplus_{b=1}^{B} Q_{br}^{(l)}$. In the basis decomposition method, we define basis transformations as $V_b^{(l)} \in R^{d^{(l+1)} x\, d^{(l)}}$. The information of the relation is learned in the $a_{rb}^{(l)}$ parameters. Here the basis decomposition functions similarly to weight sharing between relation types. In the block-diagonal decomposition we enforce sparsity in the matrices to limit the number of parameters. By forcing $W_r^{(l)}$ to be a block diagonal matrix of the form diag($Q_{1r}^{(1)}$ … … … $Q_{br}^{(l)}$) we require the matrix to have other matrices of size $R^{(d^{(l+1)}/B)\, x\, (d^{(l)}/B)}$ in its diagonal. The rest of the matrix must be zero. As such we limit the number of parameters in the matrices $W_r^{(l)}$. The idea of block decomposition is that

Sets of variables that are more tightly connected within groups than across groups can be formed from the latent characteristics.

2.3.2. Decoders (Distmult,conve,transE,etc)

For decoders DistMult, and Conve can be used depending on what we want to predict. If we want to predict the tail of the triplet Conve is usually used or DistMult if we want to predict the relation type and we have the head and the tails of the triplets. For loss function usually binary cross entropy is used.

2.4. Composition based GCN (Comp-Gcn)

This model is also based on GCN's. This method creates embeddings for each edge type in addition to the embeddings created for the nodes. This edge type embeddings are composed with the node embeddings to create better representations that lead to improved results.

2.4.1. CompGCN encoder

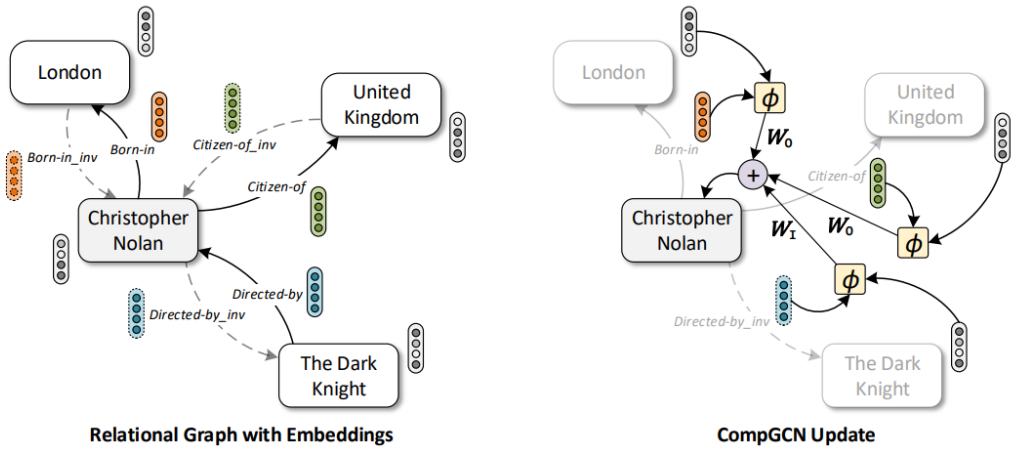


Figure 2-10 from (Vashishth et al.*, 2019) the embedding process for CompGCN*

If we consider the graph G =(V, R, E, X, Z) where V is the set of nodes, E is the set of edges, R is the set of relation types, X are the node embeddings and Z are the edge types embeddings. More precisely $X \in R^{|V| \, x \, do}$ where do is the embedding dimension of each node and $Z \in R^{|R| \, x \, do}$. The CompGCN procedure is based on the assumption that information travels both ways in the edges. As such in the edge set of E which is the triplets (u,v,r) u and v the nodes and r the relation, the inverse relations (u,v,r^{-1}) are added. Also the self-loops are added as well. We have then $E' = E \cup \{(u, v, r^{-1}|(u, v, r) \in E\} \cup \{(u, v, T)|u \in V\}$. (Vashishth *et al.*, 2019)

The idea here is to first select our node of interest. Then figure out which nodes are connected with our node and with what edge type. Then we compose each connecting nodes embedding with embedding of our relation type. After that we treat the new embeddings, we created as the embeddings of nodes and proceed similarly to the original GCN method. We do this once for the original relations, once for the inverse relations and once for the self-loops.

More precisely the new embedding that are created for each connecting node are mathematically expressed as $e_o = \varphi(e_s, e_r)$. Here e_r is the embedding of the relation, e_s the embedding of the connecting node with the relation r and ϕ is a composition operator such as subtraction, multiplication and circular-correlation.

For comparison the updated embeddings in the R-GCN method an outward only directed graph are given by a sum of the form $h_u = f(\sum_{(u,r)\in N_u} W_r h_u)$. Where N_u is the set of 1 hop away neighbors of u and with what edge type they are connected to u. h_u are the embeddings of the neighbor nodes. W_r are weight matrices for each edge type and they multiply the h_u vectors that are connected to our node under relation r.

In compGCN the update equation for each layer becomes $h_u^{k+1} = f(\sum_{(u,r)\in N_u} W_{\lambda(r)}^k \varphi(h_u^k, h_r^k))$ here h_u^{k+1} is the updated embedding of each node, $W_{\lambda(r)}^k$ are the weight matrices of each k layer of the network, $\varphi(h_u^k, h_r^k)$ is the composition function described before and h_u^k, h_r^k are the node embedding and edge type embeddings of the neighbor nodes to update our node of interest. One key difference here compared to R-GCN method is that the $W_{\lambda(r)}^k$ is actually 3 matrices per layer and not one per edge type per layer as it was in RGCN. It is more akin to the GCN model in that regard that uses only one parameter matrix per layer. One for each type of relation. The original relations W_O $r \in R$,the inverse relations W_I $r \in R_{inv}$ and a self-loop matrix.

For the model to scale with large number of relations the initial relation representations are a linear combination of a set of basis vectors. If the set is $\{u_1, u_2 u_3 \dots \dots u_b\}$ each relation is expressed in these vectors as $z_r = \sum_{b=1}^{B} a_b u_b$. The information to be learned about the relation is in the a_b. It is a learnable scalar weight($a_b \in \mathcal{R}$ that depends on each relation.

In the end in each layer a transformation of the relation embeddings is done to bring them to the new vector space the new node embeddings are. This transformation mathematically is described as $h_r^{k+1} = W_{rel}^k h_r^k$.

2.4.2. Available decoders

The decoders available here are the same as in GCN and RGCN.

2.5. Harpa

The model's basic idea is to use two GAT's. One to learn the hidden node embeddings and one to learn for the paths between nodes embeddings. The authors pass on the decoder the path embeddings as well as the hidden node embeddings. Their aim Is by doing this to improve the predictions achieved by simply using the hidden node embeddings.

2.5.1. Harpa encoder

2.5.1.1. Encoder

Harpa uses a two-layer GAT network to encode information. The first attention layer is used to embed information about the triplets of the network. The second layer is used to embed information about the paths between nodes.

2.5.1.1.1. Triplets-level embedding

According to (Zhang, Wang and He, 2023) we start by creating an embedding for each triplet. The embedding is of the form $u_{hrt} = \sigma(\omega_1[h||r||t])$. In the previews relation

the initial representation of the triplet's head is represents as h, relation as r and tail as t. ω_1 is a learnable matrix,σ is an activation function and || is the concatenation operator.

On these triplets' embeddings the first GAT layer measures the attention between them. First the attention coefficient between the embeddings a_{hrt} is measured. This is done according to $a_{hrt} = \frac{\exp(LeakyReLU(\omega_2 u_{hrt}))}{\sum_{t\in N_h}\sum_{r\in R_{ht}}\exp(LeakyReLU(\omega_2 u_{hrt}))}$. As with ω_1, ω_2 is a learnable matrix, used to linear transform u_{hrt}, N_h is the set of shortest paths that connect our node to its N-hops neighbors, R_{ht} is the set of relations between head and tail. (Zhang, Wang and He, 2023)

These coefficients will be used in order to compute new representations for the node embeddings. The process described is a multiheaded one. For each head m the quantity$\sum_{t\in N_h} \alpha_{hrt}^m u_{hrt}^m$) is computed. The multi head attention is used to stabilize the attention mechanism and better learn the node representations. The embedding after the

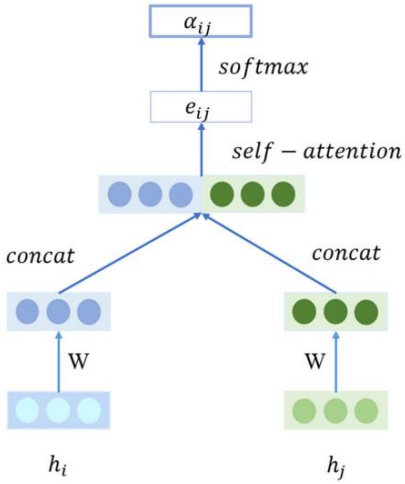


Figure 2-11 GAT propagation from
(Zhang, Wang and He, 2023)

multihead attention is according to $h' = \sigma(\sum_{m=1}^{M}\sum_{t\in N_h} \alpha_{hrt}^m u_{hrt}^m$). The m superscript is the head ,|| is the concatenation operator, α_{hrt}^m are the attention coefficient for head m ,and triplet hrt and u_{hrt}^m is the node embedding in head m of the triplet hrt. On these new h' embeddings of the node the process from before can be performed again to create even deeper embeddings.

On the last layer concatenation no longer makes sense and a simple averaging takes its place of form $h'' = ((\frac{1}{M})(\sum_{m=1}^{M}\sum_{t\in N_h} \alpha_{hrt}^m u_{hrt}^m)$).

Furthermore, at the final layer of the GAT we have the h'' embeddings. The matrix E_e is composed of these embeddings. At this matrix as a skip connection, we add the original embeddings which are composing the matrix E_0 multiplied by a learnable matrix ω_3.
$E' = E_e + \omega_3 E_0$

Finally, the embeddings of the relations are updated to the dimension of the last linear transformation by another matrix ω_R $R' = \omega_R R$.

2.5.1.2. Path embedding encoder

The path embedding layer aims to learn the semantic similarity between paths and relations. Here the idea is that the two hop paths between triplets that are similar to the relation between these nodes are selected and the rest discarded.

For each path p that consists of relations $\{r_1, r_2 r_3 r_4 r_5 \ldots . r_l\}$ we create its embedding by summing the relation embeddings $p = r_1 + r_2 + r_3 + r_4 + r_5 \ldots . + r_l$. Then we calculate the similarity scores t_{pi}. In order to calculate t_{pi} for each relation type we search the graph and find all two hop paths between nodes that are also connected by the corresponding relation

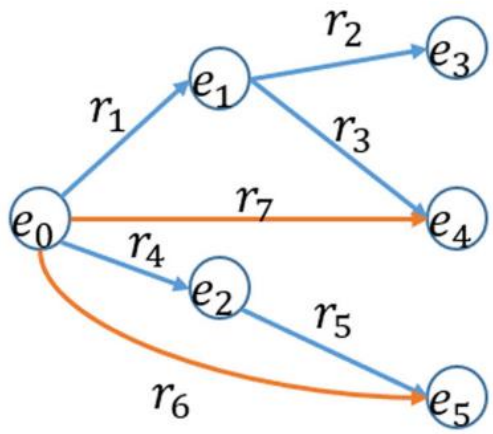


Figure 2-12 The 1 hop neighborhood of node e0 from (Zhang, Wang and He, 2023)

type. Then we calculate t_{pi} according to $t_{pi} = \sigma(\delta([r||p_i]))$ where || is the concatenation operator r the relation type, p_i the two hop path , δ a linear transformation matrix and σ an activation function. We select a threshold for the similarity and if t_{pi} is above this threshold we keep the path if not we discard it.

Afterward in the selected paths we measure attention between the two hop paths and the relation type. The relative attention coefficient of each path to the relation is $\beta_{p_i,r}$. In order to compute $\beta_{p_i,r}$ we first compute $\rho_{p_i,r} = \mathrm{V}(\mathrm{W}[p_i||r])$ where V and W are trainable parameters. Then we compute $\beta_{p_i,r} = \frac{\exp\left(LeakyReLU\left(\rho_{p_i,r}\right)\right)}{\sum_{i=1}^{s}\exp\left(LeakyReLU\left(\rho_{p_i,r}\right)\right)}$ to measure attention. Finally, we create a path embedding representation of the form $P_r = \sum_{i=1}^{s}\beta_{p_i,r}\, pi$ where pi are the two hop paths.

The paths are stacked in a matrix P_r and are added to the matrix of the transformed relations R' along with a trainable matrix ω_p or more specifically $R'' = R' + \omega_p P_r$.

2.5.2. Decoders

The model uses ConvKB model as a decoder.

2.5.3. Training of the model

The Harpa method according to the authors trains the encoder as well as the decoder. It makes predictions only with the encoder and then with the encoder-decoder. The encoded embeddings are trained with the TransE method and the decoder with the ConvKB. On both the predictions are used with cross-entropy loss.

Chapter 3 : Research Methodology

The idea we tried was to enhance the relation embeddings of the CompGCN model with the HARPA path embedding method.

3.1. CompGCN procedure

In the CompGCN model, the representation update for each node incorporates a composition of both the neighbor nodes and the embeddings representing relation types. This differs from the conventional GCN, which solely relies on the node embeddings of the neighbor nodes. In CompGCN, an embedding is initialized for each relation type in addition to each node. For instance, in the datasets we will be utilizing, such as WN18RR and FB15k-237, there are 11 and 237 relation types, respectively. Hence, there will be 11 and 237 embeddings for the respective relation types in addition to the node embeddings.

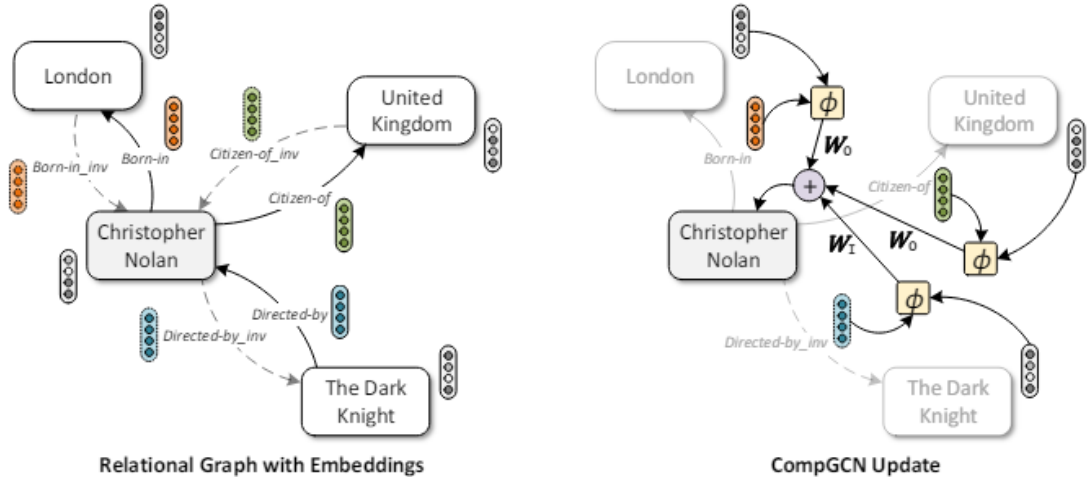


Figure 3-1 Update of CompGCN

In Figure 3-1 Update of CompGCN we observe an example of the update process. Christopher Nolan is the node undergoing update. The neighbor nodes, such as "The Dark Knight," "United Kingdom," and "London," are combined with their corresponding relation type embeddings. Typically, the composition function ϕ is selected to be a simple subtraction. Thus, the new embeddings to be utilized for the update are:

$\varphi(The\ dark\ knight, directedby) = The\ dark\ knight - directedby$

$\varphi(United\ kingdom, Citizen\ of) = United\ kingdom - Citizen\ of$

$\varphi(London, Born\ In) = London - Born\ In$

London, The dark knight and United kingdom are node embeddings.

Directedby, Citizen of ,Born In are relation type embeddings.

These composed embeddings are used in the update of the CompGCN procedure.

So far, the updated representation of the node embedding is in each of the composed embeddings the filter matrix W_o is applied and then the embeddings are summed to create

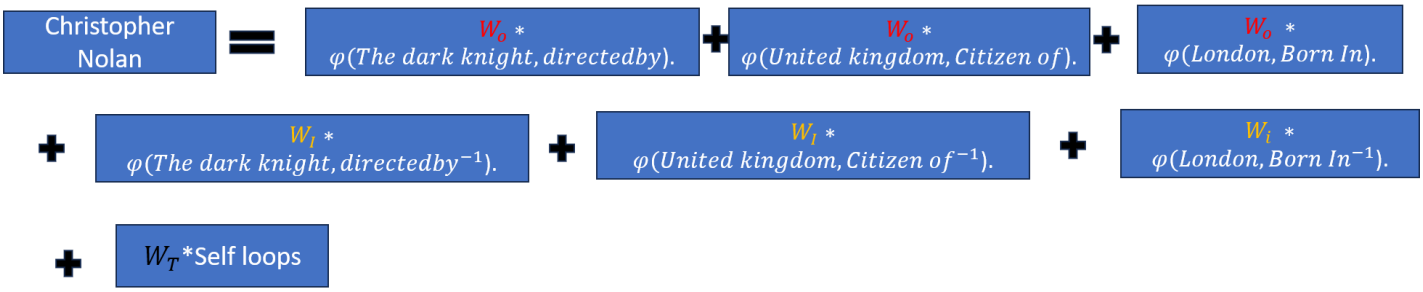
Christopher Nolan = $W_o * \varphi(The\ dark\ knight, directedby)$ + $W_o * \varphi(United\ kingdom, Citizen\ of)$ + $W_o * \varphi(London, Born\ In)$

the updated representation.

The CompGCN model extends the edge set by including inverse relations and employs them with a distinct filter matrix denoted as W_I. In datasets like WN18RR and FB15k-237, which originally consist of 11 and 237 normal relations, CompGCN augments the edge set with inverse relations. Consequently, if there are 11 and 237 relation types, respectively, CompGCN generates an additional 11 and 237 embeddings for the inverse relations. For these reverse edges new compositions with the neighboring nodes are created.

Dataset	Relation types	Normal relations	Inverse Relations
WN18RR	11	0,…..,10	11,…..,21
FB15k-237	237	0,…..,236	237,……,473

The model uses a separate filter matrix for the self-loops W_T as well. The final representation is :



Where the representations of the inverse relation types are represented as:

$Citizen\ of^{-1}$, $Born\ In^{-1}$ $and\ directedby^{-1}$.

3.2. Harpa update of the relation type

The first thing we did is for each relation type to collect all two hop paths in the graph.

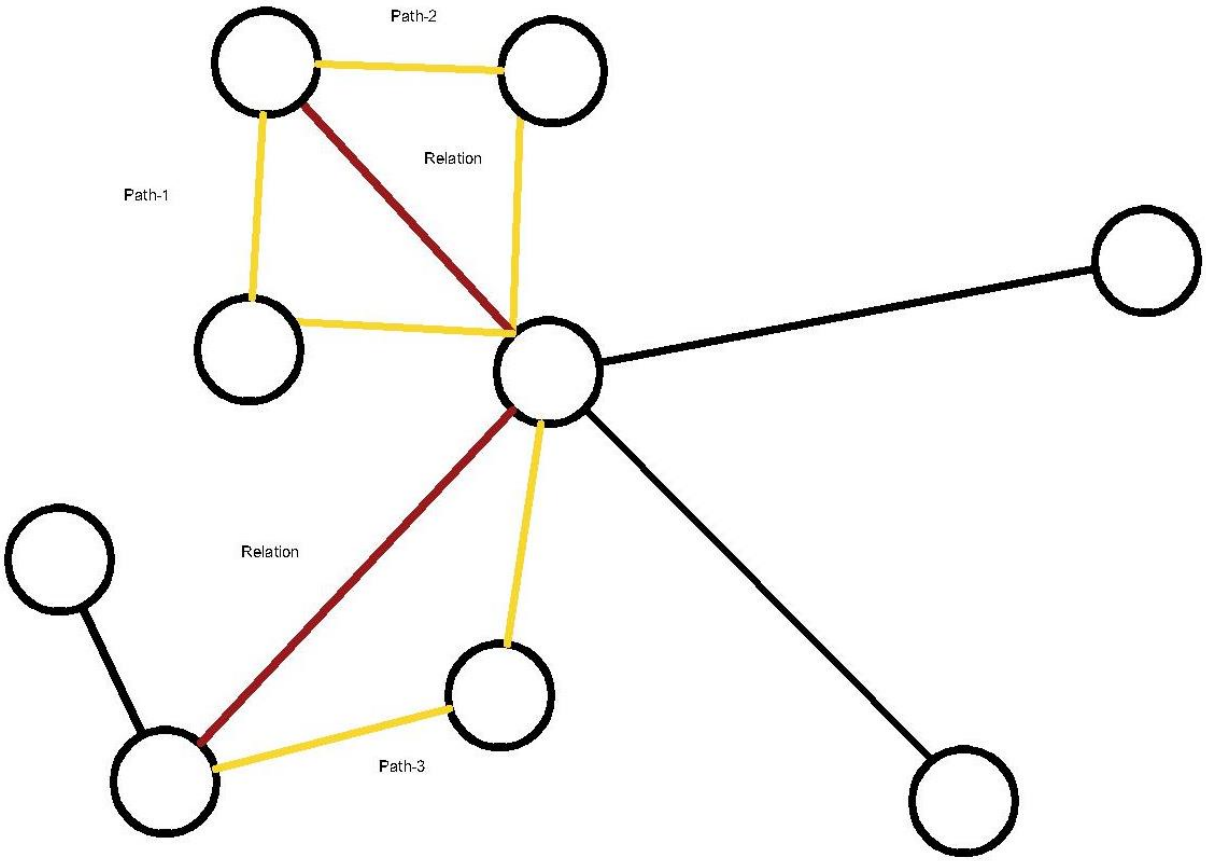


Figure 3-2 Path searching for a specific relation type in the graph.

After collecting all the paths for each relation type, we implement the HARPA path embedding procedure. For each relation, we calculate the t_{pi} coefficient between the relation type embedding and the path embedding. The path embedding is simply the sum of the relation type embeddings comprising the path. Subsequently, for paths with t_{pi} greater than the threshold, we evaluate their attention to the relation type embedding using a Graph Attention Network (GAT), as recommended by the authors.

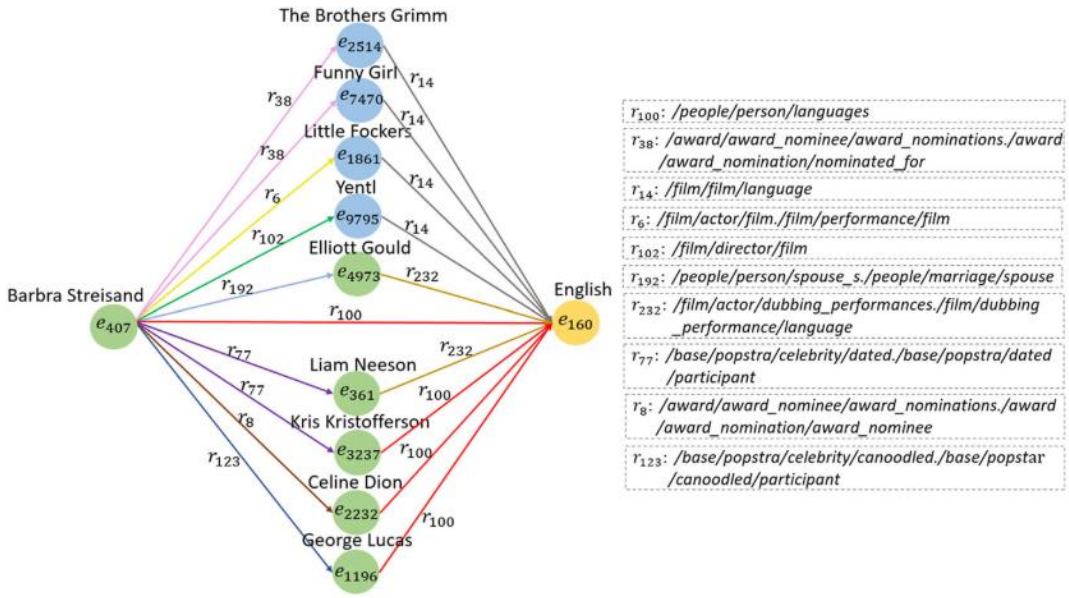


Figure 3-3 HARPA path attention from (Zhang, Wang and He, 2023)

path	similarity score	attention score
p_1, p_2={r_{38}, r_{14}}	-0.3553 *(filtered)*	—
p_3={r_6, r_{14}}	-0.9053 *(filtered)*	—
p_4{r_{102}, r_{14}}	-1.4337 *(filtered)*	—
p_5{r_{192}, r_{232}}	0.7781	0.0013
p_6{r_{77}, r_{232}}	0.6475	0.0016
p_7{r_{77}, r_{100}}	0.6020	0.0036
p_8{r_8, r_{100}}	0.6158	0.0038
p_9{r_{123}, r_{100}}	0.6137	0.0030

Figure 3-4 HARPA t_{pi} calculation for the relation r100 and its two hop paths from (Zhang, Wang and He, 2023).

In Figure 3-3 HARPA path attention from (Zhang, Wang and He, 2023)Figure 3-4 HARPA t_{pi} calculation for the relation r100 and its two hop paths from (Zhang, Wang and He, 2023)., we illustrate the selection of available two-hop paths based on the t_{pi} calculation. For paths with a similarity score exceeding the set threshold, we compute an attention score to the relation type. Using this attention score and the path embeddings, we create an embedding for all the two-hop paths of each relation type $P_r = \sum_{i=1}^{s} \beta_{p_i,r}\, pi$ Here, pi represents the two-hop path, and $\beta_{p_i,r}$ signifies the attention coefficient. Our objective is to utilize the P_r embeddings alongside the relation type embeddings.

3.3. HARPA -CompGCN combination

The normal update procedure of the CompGCN is the following:

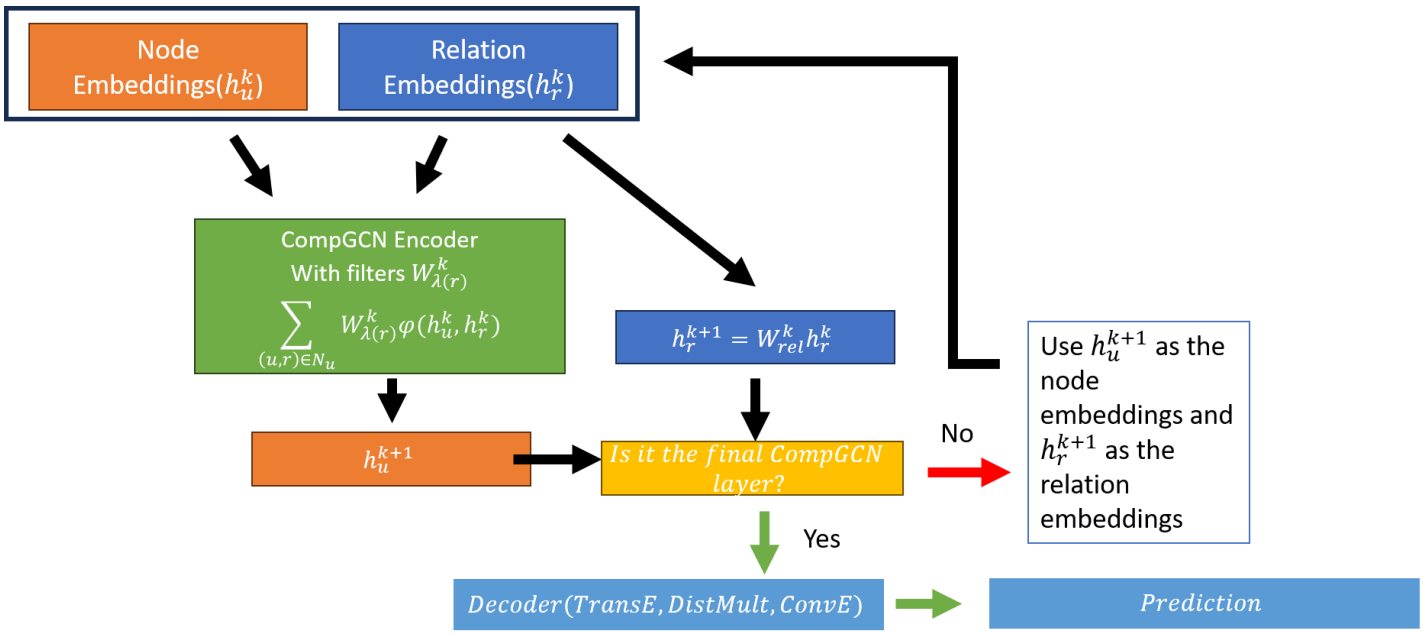


Figure 3-5 CompGCN diagram

The relation type embeddings of relation r (h_r^k) at layer k of the CompGCN are updated (h_r^{k+1}) to match the embedding dimension of the updated node embedding $\left(h_u^{k+1}\right)$. We integrated the HARPA process into CompGCN in the following two ways:

1. **Method 1**: Using the standard t_{pi}calculation as originally described in the HARPA paper.
2. **Method 2**: Employing a modified t_{pi} calculation to better align with the CompGCN framework.

The difference in the two ways we tried is in the t_{pi} calculation

Method 1 :

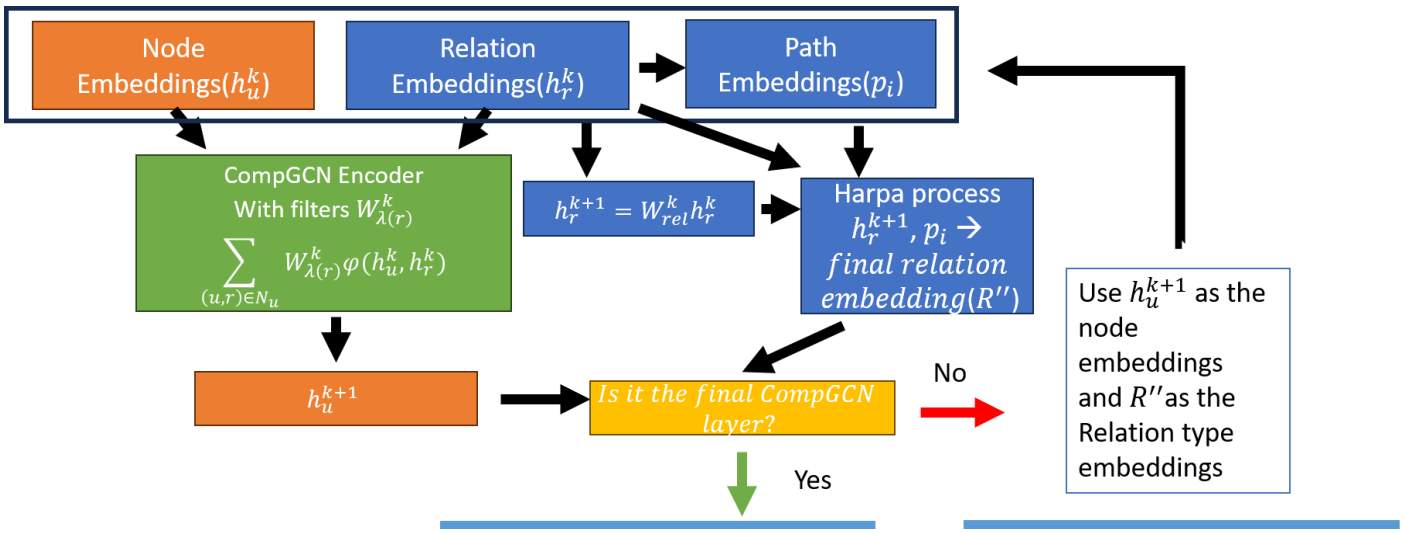


Figure 3-6 First way the HARPA process is integrated in the CompGCN procedure

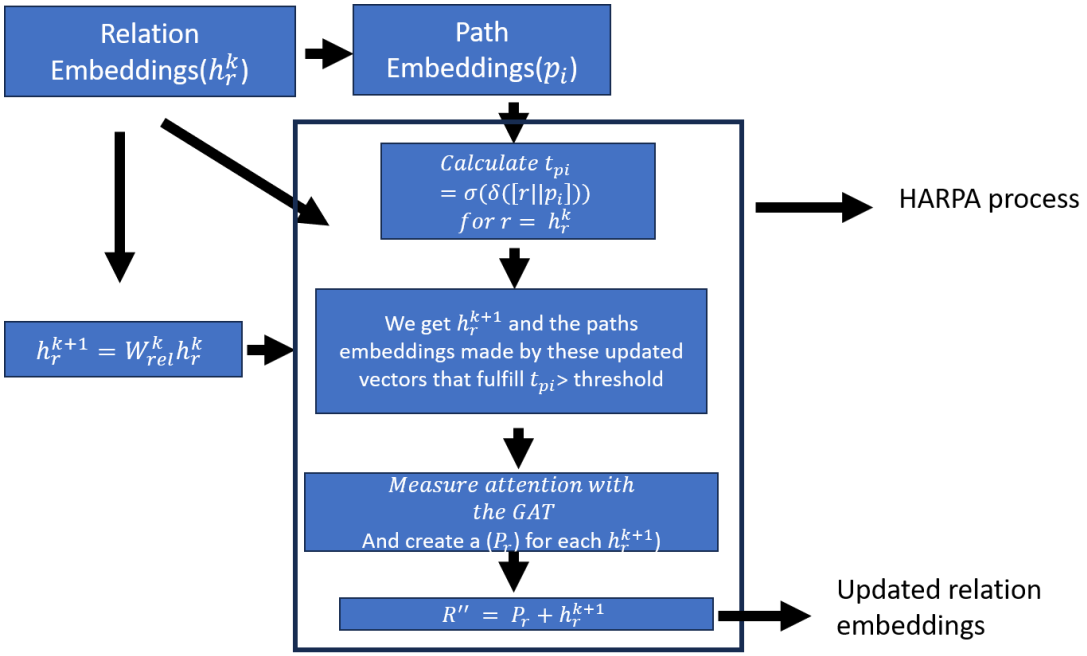


Figure 3-7 Diagram of how the HARPA process is combined with the CompGCN procedure.

For example, in the paths in

path	similarity score	attention score
p_1, p_2={r_{38}, r_{14}}	-0.3553 *(filtered)*	—
p_3={r_6, r_{14}}	-0.9053 *(filtered)*	—
p_4{r_{102}, r_{14}}	-1.4337 *(filtered)*	—
p_5{r_{192}, r_{232}}	0.7781	0.0013
p_6{r_{77}, r_{232}}	0.6475	0.0016
p_7{r_{77}, r_{100}}	0.6020	0.0036
p_8{r_8, r_{100}}	0.6158	0.0038
p_9{r_{123}, r_{100}}	0.6137	0.0030

Figure 3-8 Using the previews t_{pi} calculation example

In the first approach, the t_{pi} calculation was performed using the relation type embeddings that are fed into each layer of CompGCN. For paths that have an appropriate t_{pi} , we create the updated path embedding. This embedding is the sum of the updated relation type embeddings that constitute the two-hop path. The Graph Attention Network (GAT) then measures the attention between the updated relation type embeddings and the updated path embeddings, creating the P_r embedding. This P_r embedding is then added to the corresponding relation type embedding ($R'' = P_r + h_r^{k+1}$).

The relation type embeddings r_i in Figure 3-8 Using the previews t_{pi} calculation example are symbolized as h_r^k in Figure 3-7 Diagram of how the HARPA process is combined with the CompGCN procedure.. K symbolizes the layer of the CompGCN and r the relation type.For reference in Figure 3-8 these are the two hop paths of the $r_{100}(h_{100}^k)$ relation type embedding. The path embeddings p_i made for each two-hop path are used in the t_{pi} calculation shown in Figure 3-8 as the similarity score. For the paths that have an adequate similarity score new path embeddings using the h_r^{k+1} relation type embeddings are used. These paths are used to measure the attention score in GAT. So for path 4 we would create a path embedding $h_{102}^k + h_{232}^k$ to measure the similarity score with h_{100}^k. If the scoring was adequate we would then create $h_{102}^{k+1} + h_{232}^{k+1}$ to measure attention with h_{100}^{k+1}.

Method 2 :

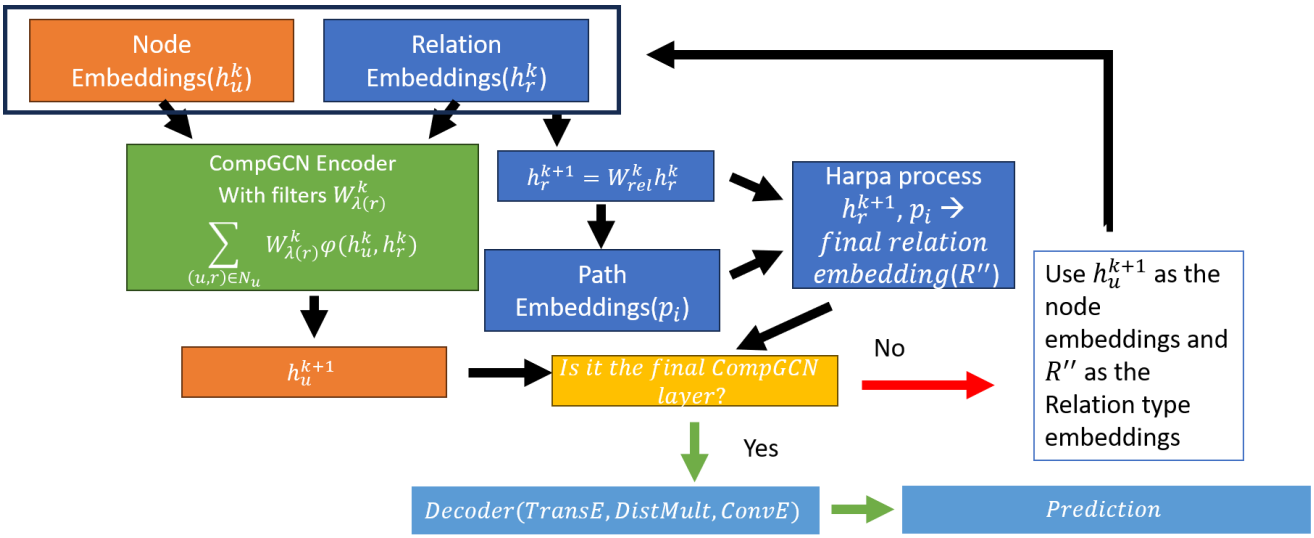


Figure 3-9 Second way we implemented the HARPA path embedding process in the CompGCN

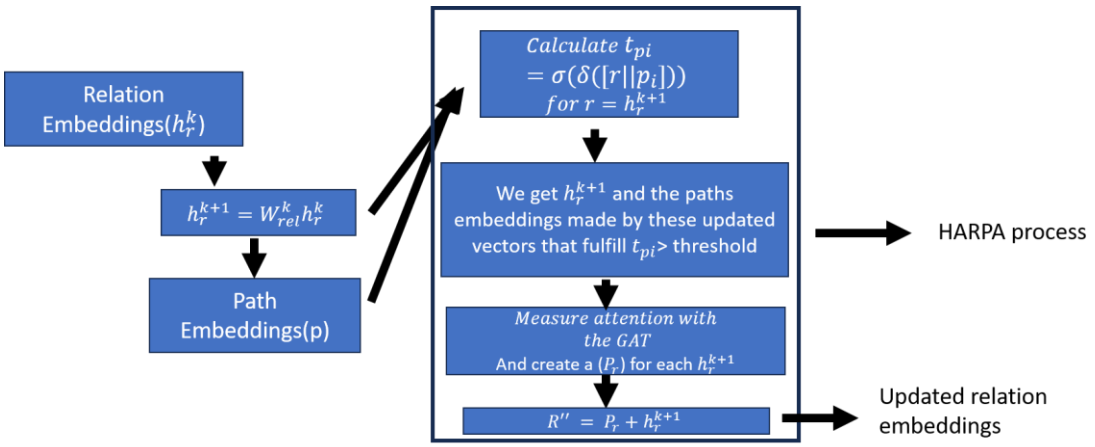


Figure 3-10 How the path embedding is implemented in the second try of HARPA attentions is measured between the path embeddings(p) and the corresponding updated relation type h_r^{k+1}

In the second approach, the t_{pi} calculation was performed using the updated relation type embeddings and the updated path embeddings. This means that after the initial update of the relation type embeddings in CompGCN, the t_{pi} coefficient was calculated based on these updated embeddings. For paths (p) that met the criteria with the updated t_{pi}, we created the updated path embeddings as the sum of the updated relation type embeddings that make up the two-hop path. Following this, the Graph Attention Network (GAT) assessed the attention between the updated relation type embedding(h_r^{k+1}) and the updated path embeddings(p). The P_r embedding generated from this attention mechanism was then combined with the corresponding relation type embedding.

Here the relation type embeddings r_i in Figure 3-8 Using the previews t_{pi} calculation example are the h_r^{k+1} in Figure 3-10 How the path embedding is implemented in the second try of HARPA attentions is measured between the path embeddings(p) and the

corresponding updated relation type h_r^{k+1}. Again using path 4 as an example. We would create a path embedding $h_{102}^{k+1} + h_{232}^{k+1}$ to measure the similarity score with h_{100}^{k+1}. If the scoring was adequate we would then use it again to measure attention with h_{100}^{k+1}.

3.4. HARPA-CompGCN tweaking

The original HARPA method makes two predictions. First, it uses the triplet embeddings u_{hrt} and relation type embeddings (R)) before they are updated. Second, it uses the updated triplet embeddings $u_{hrt}{}'$ and updated relation type embeddings (R''). The updated relation type embeddings are calculated as $R'' = R' + \omega_p P_r$, where ω_p is a linear projection matrix P_r are the total path embeddings. The relation type embeddings $R' = \omega_R R$ are obtained by projecting the original relation type embeddings(R) tothe same space as the updated triplet embeddings(R' *and* $u_{hrt}{}'$ have the same dimension).

In the first prediction, HARPA uses the TransE decoder, and in the second prediction, it uses the ConvKB decoder. To investigate if these different decoders affect the predictions, we modified HARPA and experimented with various prediction methods.

1) First, we made only one prediction and experimented with different similarity thresholds and loss functions. We used only the updated node embeddings and relation type embeddings of the CompGCN and made only one prediction. In the case of CompGCN, the updated relation type embeddings are calculated as $R'' = P_r + h_r^{k+1}$, where P_r are the path embeddings and h_r^{k+1} are the updated relation type embeddings. This approach allowed us

Model	CompGCN layers	Loss function	Similarity threshold	t_{pi} method used	Description
ConvE sim threshold 0	1	ConvE	0	2	Our first HARPA implementation
ConvE sim threshold 0,5	1	ConvE	0,5	2	Changed the similarity threshold from the first one
TransE	1	TransE	0,5	2	Changed the Loss function to TransE
Two layer GCN ConvE	2	ConvE	0,5	2	We tried two layers of GCN

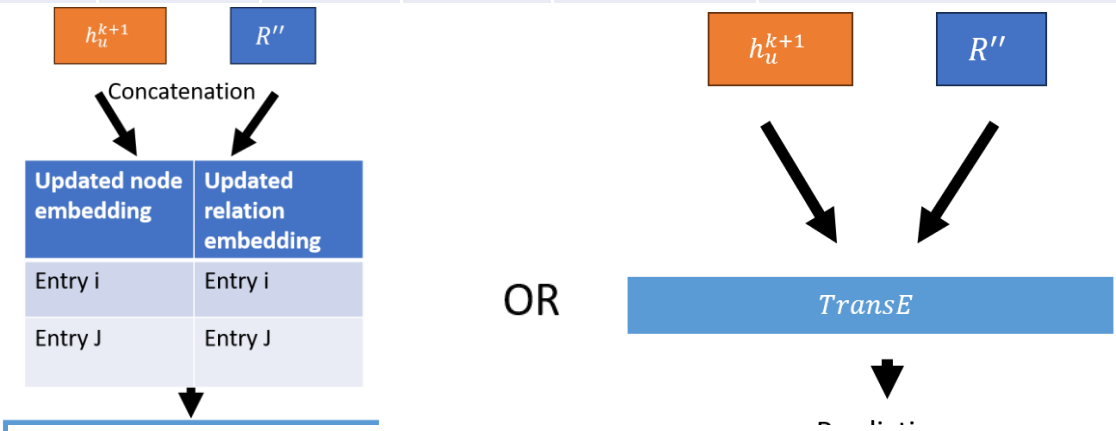
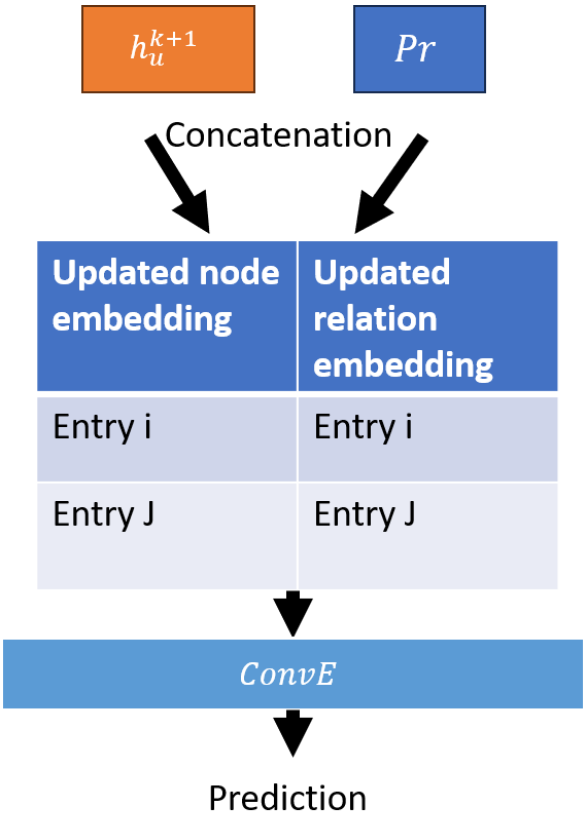


Figure 3-11 The first predictions we tried with CompGCN-HARPA

to streamline the prediction process and focus on the impact of the updated embeddings on the overall model performance.

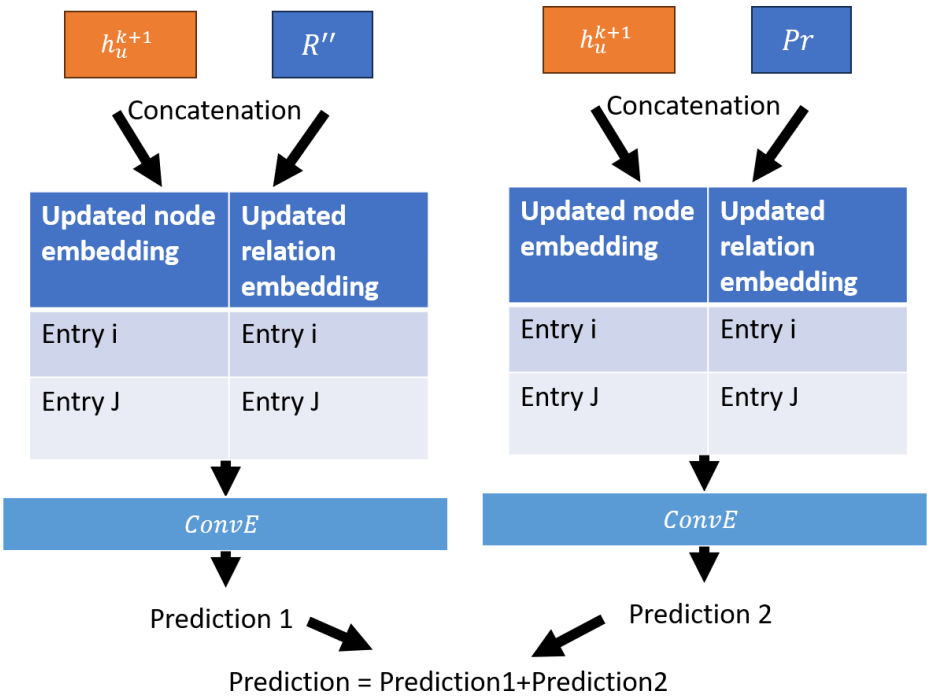
2) In our second approach, we also made only one prediction instead of two. This time, rather than updating the relation type embeddings, we used only the path embeddings. Specifically, we replaced the relation type embeddings with their corresponding P_r embeddings. This modification aimed to determine the effectiveness of relying solely on the path embeddings for predictions.

Model	CompGCN layers	Loss function	Similarity threshold	t_{pi} method used	Description
$P_r$only	1	ConvE	0,5	2	We tried making a prediction only with the path embeddings



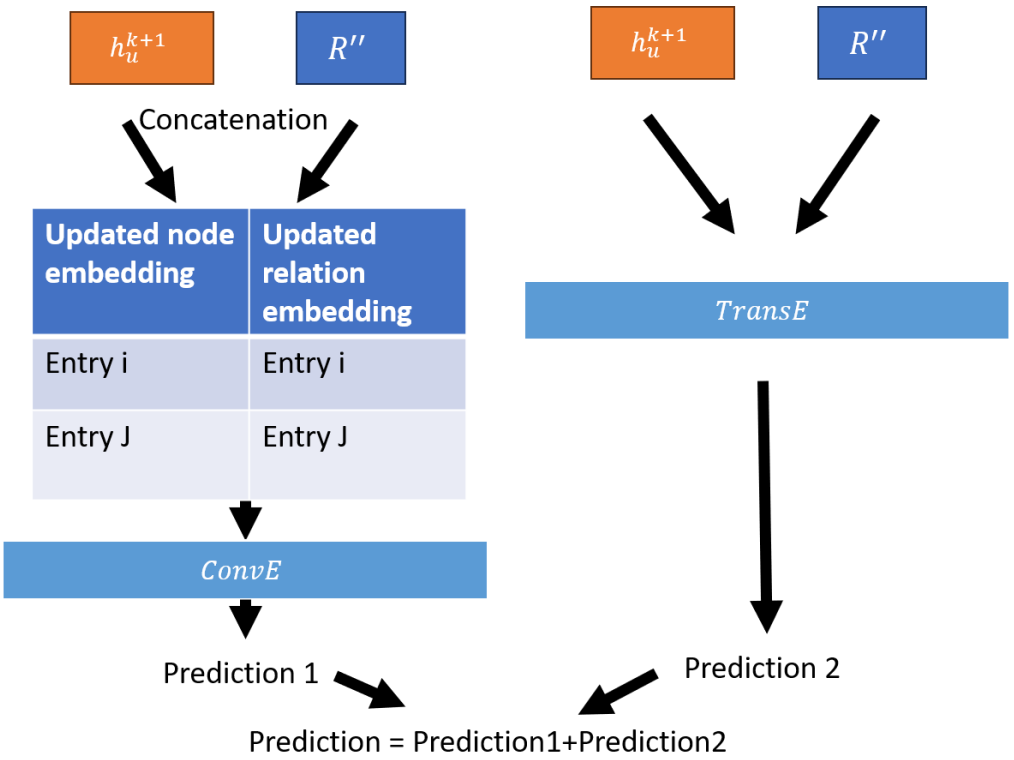
3) In our third attempt, we employed two predictions, similar to the approach in HARPA, which were subsequently aggregated. Both predictions utilized the same ConvE decoder. However, one prediction utilized the updated relation type embeddings, while the other utilized only the total path embeddings (P_r). These predictions were then aggregated, and the resulting prediction was utilized within the CompGCN framework.

Model	**CompGCN layers**	**Loss function**	**Similarity threshold**	**t_{pi} method used**	**Description**
Dual loss calculation on P and P_r	1	ConvE,	0,5	2	We tried a version of HARPA where we make one prediction using the updated relation R'' and one with only path embeddings and averaging their result



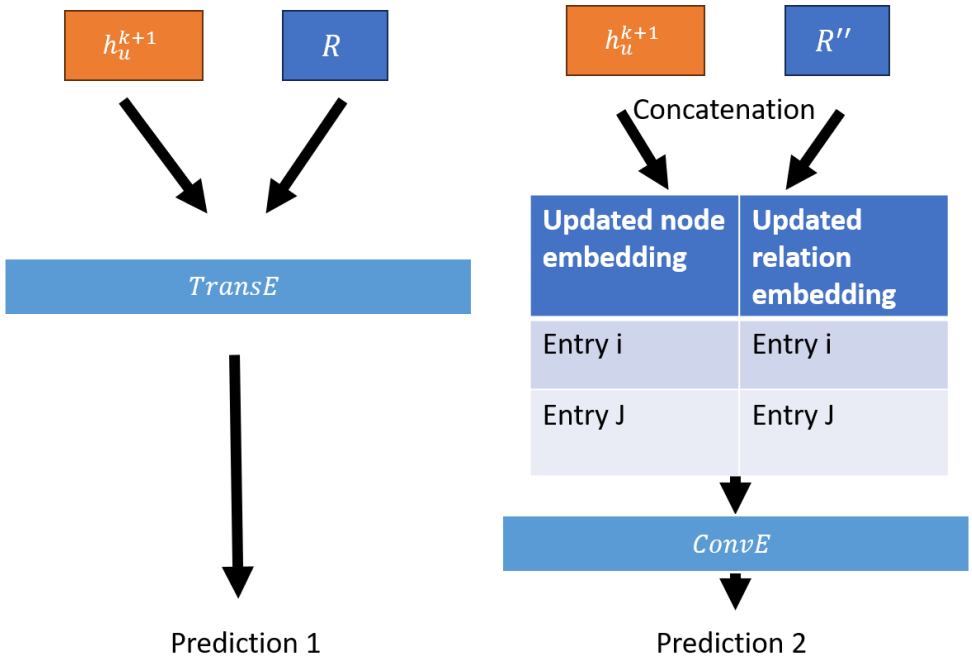
4)Similarly to the third attempt, in our fourth iteration, we utilized updated relation type embeddings in both predictions. However, the key difference lies in the choice of decoders for each prediction. In the first prediction, we employed the ConvE decoder, while in the second prediction, we utilized the TransE decoder. This approach allowed us to examine the impact of decoder selection on the predictions made within the CompGCN model.

Model	CompGCN layers	Loss function	Similarity threshold	t_{pi} method used	Description
Double loss function TransE and ConvE	1	ConvE, TransE	0,5	2	We did two predictions one with ConvE and one with TransE



5) Finally, we attempted to replicate the approach described in the HARPA paper as closely as possible. In the first prediction, we utilized the node embeddings and relation type embeddings at the start of the layer, while in the second prediction, we employed the updated node embeddings and updated relation type embeddings. For the first prediction, we utilized the TransE decoder, while for the second prediction, we employed ConvE. This approach allowed us to closely align our experimentation with the methodology outlined in the HARPA paper, facilitating a direct comparison of results.

Model	CompGCN layers	Loss function	Similarity threshold	t_{pi} method used	Description
HARPA as in paper	1	ConvE, TransE	0,5	1	We tried to implement HARPA as similar as possible to the paper



Chapter 4 : Experiments and results

We evaluated our method on the Freebase (FB15k-237) and WordNet (WN18RR) datasets, both of which were partitioned into train and test sets. Additionally, the train set was subdivided into a validation dataset and the dataset utilized for training the model. This partitioning strategy ensured that we could effectively train and validate our model while also rigorously evaluating its performance on unseen test data

4.1. Implementation details

4.1.1. Tools

The tools used in this dissertation are:

A computer running Ubuntu 22.04.4 LTS, the server of the laboratory, python version 3.9 in the server and the linux system, modules of python such as pytorch 2.0.1, pytorch geometric 2.4.0 and NetworkX 3.1.

4.1.2. Datasets

Two different datasets were used:

1. The WN18RR dataset

The WN18RR dataset is a refined version of the WN18 dataset, which is derived from WordNet, a large lexical database of English. The original WN18 dataset had a significant issue of test data leakage due to inverse relations present in the dataset. This led to overly optimistic performance metrics for knowledge graph models. WN18RR was created to address this problem by removing these inverse relations, thus providing a more realistic and challenging benchmark for evaluating knowledge graph completion and link prediction models.

Key Features of WN18RR:

Entities: 40,943

Relations: 11

Triples: 93,003

2. FB15k-237 datasets

The FB15k-237 dataset is a cleaned version of the original FB15k dataset, which was based on Freebase, a large collaborative knowledge base. Similar to WN18RR, FB15k-237 was created to address the issue of test data leakage caused by inverse relations in the original FB15k dataset. By removing these inverse relations, FB15k-237 provides a more rigorous and reliable benchmark for evaluating knowledge graph models (*FB15k-237 Dataset | Papers With Code*, no date).

Key Features of FB15k-237:

Entities: 14,541
Relations: 237
Triples: 310,116

Both WN18RR and FB15k-237 datasets are essential for advancing research in knowledge graph completion and link prediction. They share the common goal of removing inverse relations to prevent test data leakage and provide more accurate performance evaluations. However, they differ in scope and complexity:

WN18RR: Fewer relations (11) but a larger number of entities (40,943), making it suitable for evaluating models on diverse entity types.

FB15k-237: More relations (237) with fewer entities (14,541), increasing the complexity in terms of relation diversity.

These datasets have become standard benchmarks in the field, facilitating the development of more sophisticated and effective knowledge graph models

4.1.3. Data preprocessing

We utilized Networkx to extract all the two-hop paths between nodes, which were then stored in a file. For each relation type, we mapped it to all the unique two-hop paths present in the graph corresponding to that relation type. Subsequently, we generated the embeddings for these two-hop paths using the relation embeddings. This process was applied to both the normal relations and the inverse relations within the CompGCN framework.

With these preparations complete, we proceeded to measure attention between the relations and the paths. This involved creating new path embeddings as part of the HARPA process, thereby facilitating the integration of attention mechanisms between relations and paths.

4.2. Parameters of the experiments

4.2.1. Parameter selection of CompGCN

We decided to adhere to the standard parameters of CompGCN for our experiments. Among these parameters, key settings included using subtraction as the composition function between the relation embeddings and the node embedding. Additionally, we employed a default batch size of 128 during our experiments. For optimization, we utilized L2 regularization with a starting learning rate of 0.001. The dimensionality of the embeddings was set to 100, and after passing through the CompGCN layer, the hidden units of the GCN were expanded to 200.

Furthermore, we chose not to use basis vectors, as indicated in the CompGCN paper, which demonstrated that optimal results were achieved by employing distinct vectors for each relation type instead. This decision was made to align our experimentation with the findings reported in the literature.

We used two decoders for our experiments ConvE and TransE. The most important parameters of the two decoders are listed below.

ConvE :, kernel size for convolution 5

TransE: Margin : 9,Dropout after GCN: 0.1

4.2.2. Harpa implementations

Since HARPA utilizes two decoders to compute two loss functions, we initially attempted to implement it using only one loss function. Our objective was to discern the source of any observed improvements by simplifying the approach and focusing on a single loss function.

1) ConvE sim threshold 0: In the first implementation of Harpa we used only one decoder, the ConvE decoder. As mentioned before, we calculated the similarity between the paths and the relation types using $t_{pi} = \sigma\big(\delta([r||p_i])\big)$. This was done using the relation embeddings after they were updated by the CompGCN layer. We

selected a similarity threshold of 0 for t_{pi} and used it to calculate the path embeddings. Then we added the new path embeddings to the relation embedding of the CompGCN and passed it on to the CompGCN decoder.

2) ConvE sim threshold 0,5: We did the same process as in 1 with a similarity score of 0.5
3) TransE : We changed the decoder to TransE with similarity score 0.5
4) Two layer GCN ConvE : We used two GCN layers with a ConvE decoder and similarity score of 0.5.
5) P and P_r : Here as a scoring function we used ConvE. We calculated two losses using ConvE one using only the path embedding vector (P_r) and one using the R'' embeddings. We get their average as the score function and propagated the error from there.
6) P_r only :We used the P_r embeddings instead of the relation embeddings in the decoder.
7) Double loss function transE and ConvE : We used two different loss functions, one with TransE and then one with ConvE using the R'' embeddings. We backpropagated the TransE loss function first and then the ConvE. (dual loss)
8) Harpa as in paper : We created an implementation that was as close as possible to Harpa paper.

4.2.3. Model Evaluation

The evaluation was conducted on the test set segments of our two datasets. We utilized binary cross-entropy loss function to assess the results obtained from our scoring functions.

To compare the outcomes of our experiments, we employed Mean Reciprocal Rank (MRR) as the evaluation metric. MRR measures the efficiency of the algorithm in locating the desired item. Mathematically, MRR is defined as:

$$MRR = \frac{1}{U} \sum_{I=1}^{U} \frac{1}{rank_i}$$

Here, U represents the number of searches performed by the algorithm, and $rank_i$ denotes the position at which the item being searched for appears in the results during the i-th attempt.

For instance, consider the scenario of an algorithm used by Spotify to recommend songs to users. For each user i, $rank_i$ would indicate the position of the first song that the user liked in the list of recommendations. If the liked song is the first in the list, $rank_i$ would be one; if it is the second, $rank_i$ would be two, and so on.

In summary, MRR provides the average over all users of how quickly the algorithm identifies the items they are interested in.

4.3. Results for FB15k-237

Below are the results of the different implementations for FB15k-237 and the best performance of the CompGCN paper in this dataset

Harpa implementation	Validation MRR	Test MRR
conve sim threshold 0	0,35733	0,3533
ConvE sim threshold 0,5	0,35879	0,35406
Two layer GCN conve	0,3521	0,3503
TransE	0,32559	0,32718
P and P_r	0,35742	0,3532
P_r only	not possible	not possible
Double loss function transE and ConvE	0,35235	0,34873
Harpa as in paper	0,35216	0,3508

Table 4-1 Results per implementation for FB15k-237

FB15k-237	
Baseline for FB15k-237	MRR
ConvE	0.355

Table 4-2 Baseline for FB15k-237

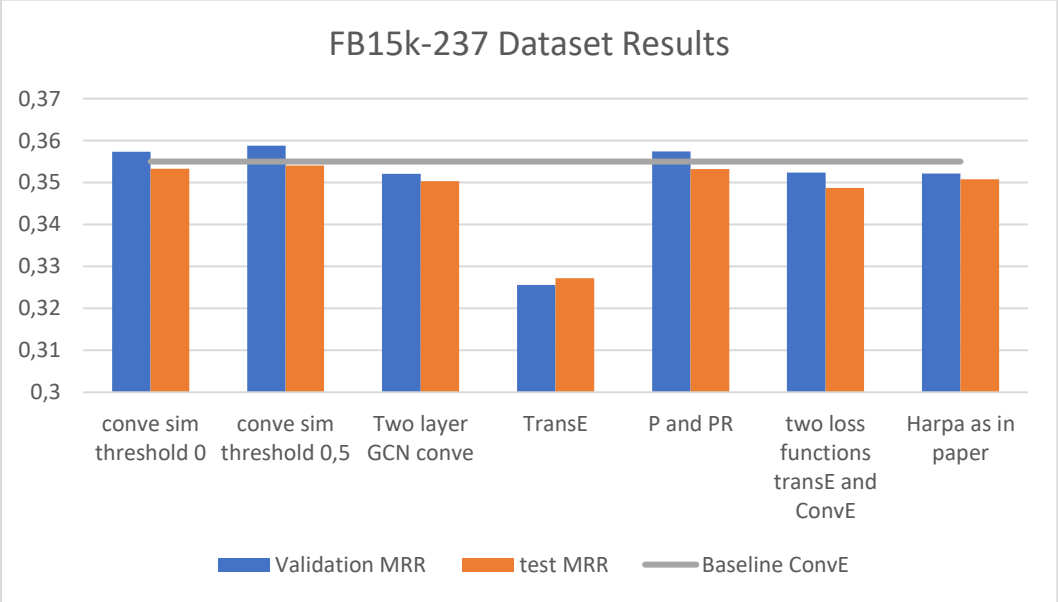


Figure 4-1 Results for FB15k-237

We observed that there was no significant difference between the methods, except for TransE, which exhibited inferior performance compared to ConvE. This outcome was anticipated, as ConvE typically outperforms TransE in these tasks. Additionally, combining methods yielded worse results compared to ConvE alone. Furthermore, a slight degree of overfitting was noted in the validation test set.

Comparison with baseline: No improvement was observed from the HARPA procedure on any implementation on the FB15k-237 dataset.

4.4. Results for WN18RR

Below are the results of the different implementations for WN18RR and the best performance of the CompGCN paper in this dataset

Harpa implementation	**Validation MRR**	**Test MRR**
conve sim threshold 0	0,46432	0,46383
ConvE sim threshold 0,5	0,46447	0,46495
Two layer GCN conve	0,45079	0,45256
TransE	0,21525	0,22213
P and P_r	0,41312	0,42858
P_r only	0,3531	0,35254
Double loss function transE and ConvE	0,46372	0,46261
Harpa as in paper	0,4295	0,4281

Table 4-3 Results per implementation for WN18RR

WN18RR	
Baseline for WN18R	MRR
ConvE	0.479

Table 4-4 Baseline for WN18RR

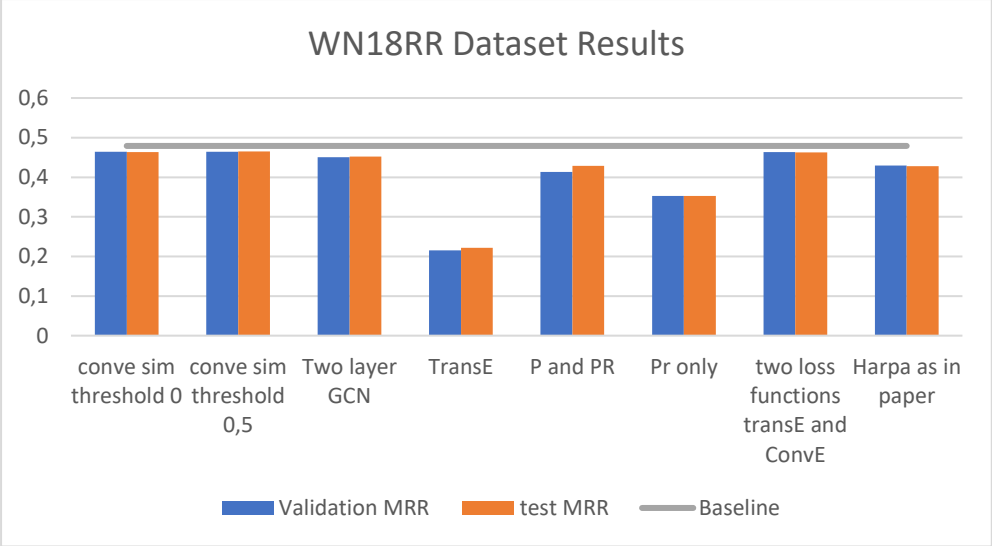


Figure 4-2 Results for WN18RR

Similar results with FB15k-237 are observed here as well. One notable difference is that here it doesn't seem for the model to overfit on the Validation set.

Comparison with baseline: No improvement was observed from the HARPA procedure on any implementation on the WN18RR dataset.

4.5. Discussion

As outlined in the research goals section (1.5), our aim was to enhance the Mean Reciprocal Rank (MRR) score of the CompGCN model and investigate whether the HARPA path embedding method could contribute to improving the performance of various methods, including CompGCN. However, despite our efforts, we were unable to replicate the improvement reported by the HARPA path embedding method within the CompGCN framework.

4.5.1. Problems with combining HARPA and CompGCN

The first problem we came across was to combine the CompGCN relations with the HARPA path embeddings. The CompGCN model works with the normal relations, the inverse relations and the self-loop relations. The updated representation of a node is given by becomes $h_u^{k+1} = f(\sum_{(u,r)\in N_u} W_{\lambda(r)}^k \varphi(h_u^k, h_r^k))$. The weight matrices $W_{\lambda(r)}^k$ for each k layer of the network are parametrized by the direction of the relations symbolized here as λ(r). This means that $W_{\lambda(r)}^k$ is chosen to be three matrices. One for the normal relations, one for the inverse relations and one for the self-loop relations $W_{\lambda(r)}^k = \begin{cases} W_O \; normal \; relations \\ W_I \; inverse \; relations \\ W_T \; self \; loop \; relations \end{cases}$. So, in CompGCN when updating the node embedding the $W_{\lambda(r)}^k \varphi(h_u^k, h_r^k)$ entities are summed for all type of relations. When combining this with the HARPA method we created the path embeddings for both the normal and the inverse relations. But because both the normal and the inverse relation two hop path embeddings depend on the same relation embeddings and

they are used in the same forward pass in the backpropagation we have to use `retain_graph=True`.

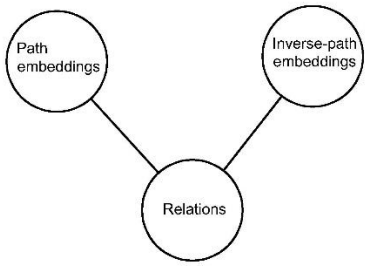


Figure 4-3 The path and the inverse-path embeddings depend on the relation embeddings and as such the retain_graph =True command must me used

This greatly slowed down the code from needing a few hours to run to need a few days in these two test sets.

Another challenge we encountered during the implementation of the method was the small contribution of the path embeddings $P_r = \sum_{i=1}^{s} \beta_{p_i,r}\, pi$ causing the HARPA weights to not update. To address this issue, we normalized P_r and added this normalized P_r in $R'' = R' + \omega_p P_r$, where R' represents the updated relation embeddings of CompGCN and R'' are the relation embedding vectors send to the decoder in the CompGCN method.

As we implemented the HARPA procedure, we encountered some variations between the original Graph Attention Network (GAT) implementation and the HARPA implementation.

1. The primary difference between the HARPA method and the original Graph Attention Network (GAT) method is the absence of multi-head attention in the graph of relation and path embeddings. While the original GAT paper leverages multi-head attention to stabilize the results, HARPA does not.

2. Another one is that it uses the t_{pi} to narrow down on which paths each relation measures attention with.

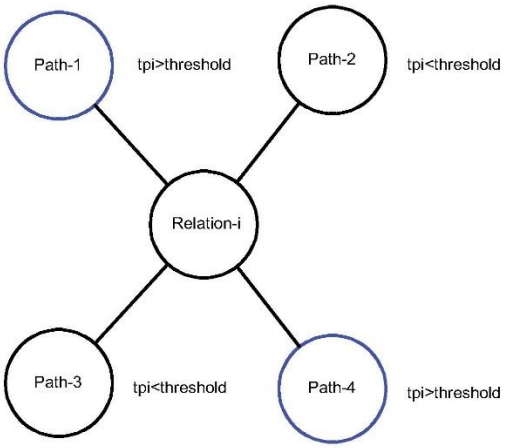


Figure 4-4 The relation measures attention only with the paths that have a tpi bigger than the selected threshold

3. Another notable difference that may have influenced the results is the variation in the attention mechanism between HARPA and GAT. In HARPA, the relation embeddings and the two-hop path embeddings can be viewed as constituting a separate graph. Within this graph, HARPA calculates attention between relations and two-hop paths in a manner akin to the GAT method. However, when comparing this approach to the original GAT method, some differences emerge.

Attention coefficient	
GAT	HARPA
$a_{i,j} = \frac{\exp(LeakyRELU[a(Wh_i \| \| Wh_j])}{\sum_{k \in N_i} \exp(LeakyRELU[a(Wh_i \| \| Wh_K]}$	$\beta_{p_i,r} = \frac{\exp\left(LeakyReLU(\ \mathrm{V}(\mathrm{W}[p_i \| \| r])\ )\right)}{\sum_{i=1}^{s} \exp\left(LeakyReLU(\ \mathrm{V}(\mathrm{W}[p_i \| \| r])\ )\right)}$
a $\in R^{2F'}$,	$V \in R^{2F'}$,
$Wh_i \in R^{F'}, Wh_j \in R^{F'}$,	$\mathrm{W}[p_i \| \| r] \in R^{2F'}$

Table 4-5 Differences between attention coefficients

We see that in V(W$[p_i||r]$) uses a bigger W matrix while GAT uses $a(Wh_i||Wh_j)$. Both a and V are vectors that are used to calculate the inner product with $Wh_i||Wh_j$ and W$[p_i||r]$ respectively. We see that in the case of HARPA a bigger matrix W is chosen compared to GAT. In GAT the matrix W is used on each node embedding while in HARPA the matrix W is used on the concatenation of the embeddings of the relations.

The above differences in the implementation of the attention mechanism could be potential reasons why it did not perform as expected.

4.5.2. Possible explanations why the models did not perform as expected

A plausible explanation for the inferior results could be attributed to the GAT utilized in HARPA. This GAT measures attention between relation types, essentially forming a small network where the relation types and the two-hop paths serve as the nodes. Considering the relatively small number of nodes in these networks, which comprises the relation types and the two-hop paths, with only 11 relation types in WN18RR and 237 in FB15k-237, the overall network size remains limited. In WN18RR, for instance, there are 40,943 nodes, and in FB15k-237, there are 14,541 nodes. It's conceivable that within such small networks, the performance of Graph Convolutional Networks (GCN) may not be optimal.

Another potential explanation for the decline in performance could be the limited utility of the two-hop path embeddings to the decoder. Decoders like ConvE are designed to identify patterns in the image formed by the node embedding and the relation embedding. In HARPA, new relation embeddings are generated by adding the original relation embedding and the corresponding two-hop path embedding. Consequently, the composite relation + path embedding vector assumes the role previously fulfilled by the original relation embedding. This process may not contribute new information, leading the network to perceive the path embeddings as noise. Consequently, the network might choose to zero out the path embeddings, resulting in diminished performance.

One more problem with the HARPA method is that in the path embedding process each path embedding is defined as the sum of the relation embeddings that compose it $p = r_1 + r_2 + r_3 + r_4 + r_5 \ldots . + r_l$. This means that in the two-hop path embedding $r_1 + r_2$ is the same as $r_2 + r_1$ which could potentially have adverse effects on the performance of the algorithm.

Another factor that may have contributed to difficulties for the model is the utilization of all the two-hop paths for each relation. This approach could have resulted in the aggregation of local information between two nodes with unrelated nodes, potentially diluting the useful information extracted from the HARPA process.

Another potential discrepancy that might have impacted the results is the manner in which HARPA creates embeddings for each triplet, compared to CompGCN, which generates embeddings for each node. Additionally, HARPA employs a different node neighborhood compared to CompGCN. HARPA considers nodes that are at least two hops away and connected to the target node via the shortest path as part of its neighborhood. This approach provides HARPA with additional information that could potentially influence the results.

CompGCN	HARPA
triplet (h,r,t)	Triplet (h,r,t)
Embedding for h Embedding for r Embedding for t	Embedding for (h,r,t) Embedding for r

Table 4-6 Differences between node embeddings between HARPA and CompGCN

Chapter 5 : Conclusion

As per the research goals outlined, our endeavor was to implement the HARPA path embedding method and reproduce its success on a distinct encoding-decoding model. However, accomplishing this task proved challenging. We encountered issues both in integrating the HARPA method with the CompGCN model and in understanding potential shortcomings inherent to the HARPA method itself.

One notable challenge arises from the small size of the relation graph, which may impact the efficacy of path learning. Factors like these contribute to the complexity of effectively applying the HARPA method in conjunction with CompGCN . The differences between our implementation and the papers of the HARPA method could have negatively affected the results. The way we implemented HARPA the decoder didn't improve its performance from the path embeddings. Using edge-based method for collecting the paths could possibly yield better results, they are however computationally costly.

5.1. Future work

A potential avenue is to apply models, such as CompGCN and HARPA, in biomedical graphs to predict links between drugs and diseases. This approach holds promise for identifying drugs that could potentially be effective for diseases beyond their original indications. IASIS, introduced by (Nentidis *et al.*, 2020), offers a valuable resource for testing these models. IASIS addresses the need for an open-source solution for the automated development of comprehensive and up-to-date disease-specific knowledge graphs.

IASIS utilizes two primary harvesters: a literature harvester and a structured harvester. The literature harvester gathers information from the Entrez API of the NCBI to perform online semantic retrieval of available literature, while the structured harvester collects information from structured databases. The gathered information for each disease undergoes processing and is utilized to construct a knowledge graph. This approach facilitates the creation of comprehensive disease-specific knowledge graphs, which can be leveraged for drug-disease link prediction tasks using GCN models.

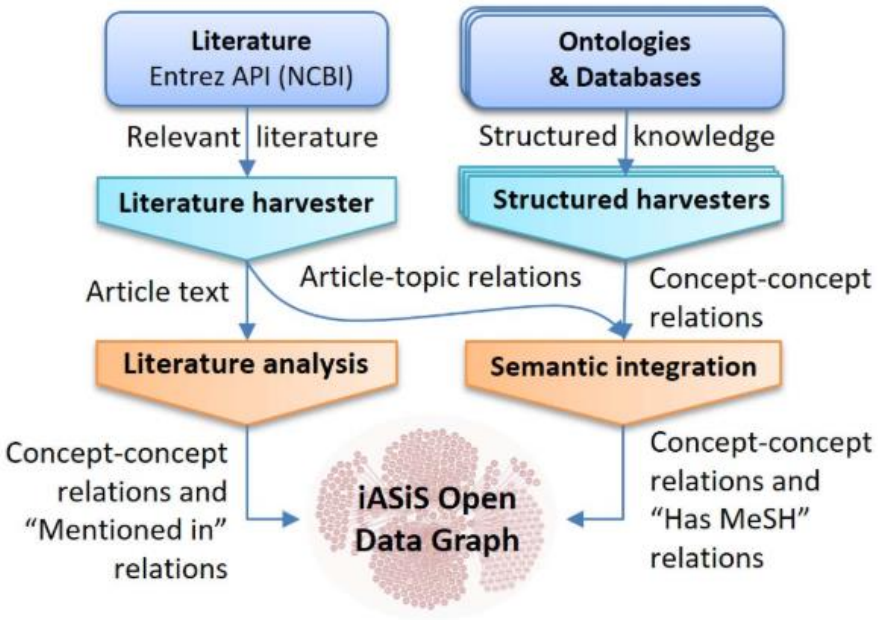


Figure 5-1 Iasis graph creation procedure from (Nentidis et al.*, 2020)*

IASIS offers a valuable application in the realm of rare syndromes. By leveraging this tool, we can focus on predicting relations between the proteins associated with the disease and potential drugs. This targeted approach enables us to narrow down the pool of drugs to be researched, thereby saving both time and resources for medical companies.

Exploring different decoders presents another avenue to potentially enhance the results of link prediction in graphs. By experimenting with various decoding mechanisms, we can assess their effectiveness in capturing the underlying relationships within the graph data. Different decoders may offer unique capabilities and performance characteristics, allowing us to identify the most suitable approach for a given dataset or problem domain. This iterative exploration enables us to refine our understanding of graph structures and improve the accuracy of link prediction tasks, ultimately contributing to advancements in various fields such as biology, social networks, and recommendation systems.

Chapter 6 : Bibliography

Aisopos, F. and Paliouras, G. (2023) 'Comparing methods for drug–gene interaction prediction on the biomedical literature knowledge graph: performance versus explainability', *BMC Bioinformatics*, 24(1), pp. 1–21. Available at: https://doi.org/10.1186/S12859-023-05373-2/TABLES/5.

Bianchi, F. *et al.* (2020) 'Knowledge Graph Embeddings and Explainable AI'. Available at: https://doi.org/10.3233/SSW200011.

Bordes, A. *et al.* (no date) 'Translating Embeddings for Modeling Multi-relational Data'.

Bronstein, M.M. *et al.* (2021) 'Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges'. Available at: https://arxiv.org/abs/2104.13478v2 (Accessed: 11 March 2024).

Cimiano, P. and Paulheim, H. (2016) 'Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods', *Semantic Web*, 0, pp. 1–1. Available at: http://www.geonames.org/ (Accessed: 8 March 2024).

Demir, C. and Ngomo, A.C.N. (2020) 'Convolutional Complex Knowledge Graph Embeddings', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12731 LNCS, pp. 409–424. Available at: https://doi.org/10.1007/978-3-030-77385-4_24.

Dettmers, T. *et al.* (2017) 'Convolutional 2D Knowledge Graph Embeddings', *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 1811–1818. Available at: https://doi.org/10.1609/aaai.v32i1.11573.

Ehrlinger, L. and Wöß, W. (2016) 'Towards a Definition of Knowledge Graphs'. Available at: http://www.semantic-web-journal.net/content/ (Accessed: 8 March 2024).

FB15k-237 Dataset | Papers With Code (no date). Available at: https://paperswithcode.com/dataset/fb15k-237 (Accessed: 1 April 2024).

Hamilton, W.L. (2020) 'Graph Representation Learning', 14(3).

Hogan, A.; *et al.* (2021) 'Knowledge Graphs', *ACM Computing Surveys*, 54(4), pp. 1–37. Available at: https://doi.org/10.1145/3447772.

Kipf, T.N. and Welling, M. (2016) 'Semi-Supervised Classification with Graph Convolutional Networks', *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* [Preprint]. Available at: https://arxiv.org/abs/1609.02907v4 (Accessed: 14 March 2024).

Meilicke, C. *et al.* (2020) 'Reinforced Anytime Bottom Up Rule Learning for Knowledge Graph Completion'. Available at: https://arxiv.org/abs/2004.04412v1 (Accessed: 7 April 2024).

Nentidis, A. *et al.* (2020) 'IASiS open data graph: Automated semantic integration of disease-specific knowledge', *Proceedings - IEEE Symposium on Computer-Based Medical Systems*, 2020-July, pp. 220–225. Available at: https://doi.org/10.1109/CBMS49503.2020.00049.

Nguyen, Dai Quoc *et al.* (2017) 'A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network', *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language*

Technologies - Proceedings of the Conference, 2, pp. 327–333. Available at: https://doi.org/10.18653/v1/N18-2053.

Nickel, M., Tresp, V. and Kriegel, H.-P. (2011) 'A Three-Way Model for Collective Learning on Multi-Relational Data'.

Pujara, J. *et al.* (2013) 'Knowledge graph identification', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8218 LNCS(PART 1), pp. 542–557. Available at: https://doi.org/10.1007/978-3-642-41335-3_34/COVER.

Sang, S. *et al.* (2018) 'SemaTyP: A knowledge graph based literature mining method for drug discovery', *BMC Bioinformatics*, 19(1), pp. 1–11. Available at: https://doi.org/10.1186/S12859-018-2167-5/TABLES/5.

Schlichtkrull, M. *et al.* (no date) 'Modeling Relational Data with Graph Convolutional Networks Peter Bloem'.

Sun, Z. *et al.* (2019) 'RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space', *7th International Conference on Learning Representations, ICLR 2019* [Preprint]. Available at: https://arxiv.org/abs/1902.10197v1 (Accessed: 9 March 2024).

Trouillon, T. *et al.* (2016) 'Complex Embeddings for Simple Link Prediction'. Available at: https://github.com/ (Accessed: 9 March 2024).

Vashishth, S. *et al.* (2019) 'Composition-based Multi-Relational Graph Convolutional Networks', *8th International Conference on Learning Representations, ICLR 2020* [Preprint]. Available at: https://arxiv.org/abs/1911.03082v2 (Accessed: 15 March 2024).

Veličković, P. *et al.* (2017) 'Graph Attention Networks', *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* [Preprint]. Available at: https://doi.org/10.1007/978-3-031-01587-8_7.

Wang, Z. *et al.* (2014) 'Knowledge Graph Embedding by Translating on Hyperplanes', *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), pp. 1112–1119. Available at: https://doi.org/10.1609/AAAI.V28I1.8870.

WN18RR Dataset | Papers With Code (no date). Available at: https://paperswithcode.com/dataset/wn18rr (Accessed: 1 April 2024).

Yang, B. *et al.* (no date) 'EMBEDDING ENTITIES AND RELATIONS FOR LEARN-ING AND INFERENCE IN KNOWLEDGE BASES'. Available at: http://freebase.com (Accessed: 9 March 2024).

Zamini, M., Reza, H. and Rabiei, M. (2022) 'A Review of Knowledge Graph Completion', *Information (Switzerland)*, 13(8), p. 396. Available at: https://doi.org/10.3390/info13080396.

Zhang, N., Wang, J. and He, J. (2023) 'HARPA: hierarchical attention with relation paths for knowledge graph embedding adversarial learning', *Data Mining and Knowledge Discovery*, 37(2), pp. 521–551. Available at: https://doi.org/10.1007/S10618-022-00888-3/METRICS.