Otto-von-Guericke University Magdeburg

Department of Computer Science
Institute of Technical and Business Information Systems

Master Thesis

Evaluating the quality of Graph Neural Network-based embeddings over a Procedural Knowledge Graph

Author:
Baalakrishnan Aiyer Manikandan

August 11, 2023

Advisers:

Supervisor
Prof. Dr.-Ing. Ernesto William De Luca

Department of Computer Science
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Supervisor
M.Sc. Erasmo Purificato

Department of Computer Science
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Aiyer Manikandan, Baalakrishnan:
Evaluating the quality of Graph Neural Network-based embeddings over a Procedural Knowledge Graph
Master Thesis, Otto-von-Guericke University
Magdeburg, 2023.

Contents

Abstract

Knowledge graphs have various applications, from recommendations to information retrieval. It is seen as an efficient way of storing information from various sources about real-world entities and relations in a form that can be used to make further inferences. On the other hand, the usage of Knowledge Graphs in the Manufacturing process has been a relatively recent field of application, especially in procedural knowledge graphs, a very recent phenomenon. Procedural Graphs represent real-world operator knowledge involving parameters and the quality of the products produced as the effect of the parameters initially set. To utilize the operator knowledge modelled as a Knowledge graph, we use various embedding methods to learn a low-dimensional representation that can be further used in the downstream tasks. Graph neural networks have been known to effectively capture the local neighbourhood structures of Large Knowledge Graphs as per the available literature. In our thesis, we would like to demonstrate the quality of Graph neural network-based embedding for procedural knowledge graph representations. We use a practical manufacturing use case of the Fused Diffusion Model 3D printing for procedural knowledge graph and introduce an evaluation metric Matches@k evaluate the quality of embedding produced using the Traditional embedding models like translation model and semantic models against the Graph neural network-based approaches like Graph convolution network and its variants. We further will show that while Graph neural networks perform well in the case of high abstract weighted procedural graphs, these models suffer from complex graph properties at lower abstraction levels.

We now think of internal representation as great big vectors, and we do not think of logic as the paradigm for how to get things to work. We just think you can have these great big neural nets that learn, and so, instead of programming, you are just going to get them to learn everything.

Geoffrey E Hinton

Acknowledgements

I thank my supervisor MSc Erasmo Purificato for his support and guidance during my thesis work. I would also like to thank Prof. Dr.-Ing. Ernesto William De Luca for allowing me to pursue my thesis under his chair.

I want to thank my colleague and friend Manish Vipinraj Bhandari of XITASO GmbH for his constant support and check-ins, which kept me going. In the same vein, I also want to thank Richard Nordsieck of XITASO GmbH for his work, which provided the foundation for my thesis work.

Last but not least, I want to acknowledge the continuous moral support of my family and friends throughout my master's journey.

1 Introduction and Motivation

Data-Driven Machine learning has made impressive strides in various applications across different domains- language modeling, drug discovery, autonomous vehicles and many more. The eventual step and the driver of Industry 4.0 is the use of Machine learning-based automation that would learn from various manufacturing applications on the factory floor or production line. These tasks can range from quality control to load prediction. The future of the manufacturing industries is to leverage these machine learning systems to automate, plan, control and optimize the production process to eventually lead to monetary benefits and high-quality product outputs with available resources.

The most crucial aspect of data-driven machine learning processes is to structure the data and enable its usage in the various downstream tasks. Knowledge Graphs(KG) have been viewed as an efficient method to represent structured heterogeneous data, allowing us to have semantic meaning in their representation. Incorporating this knowledge stored in the form of Knowledge Graphs to efficiently capture the semantic meaning of the graphs into representations that can be used in machine learning tasks.

1.1 AIPE project

Expert knowledge is crucial in many production processes. Finding a parameterization for which the production process provides appropriate results is a crucial task in manufacturing automation. Experienced operators perform these parameterization processes using an iterative, resource- and time-intensive workflow. This workflow of parameteriza-

tion consists of four significant steps. At first, an operator sets the default starting parameters for a Fusion Diffusion model additive manufacturing process (3D -printing) and starts the process. Second, the product, thus produced, undergoes quality tests by specific quality control personnel who rate the quality of the products on various pre-defined criteria. Third, depending on the quality of the product produced in the first iteration, the expert operator adjusts the process parameter to obtain a better result. Fourth, at each cycle and quality test, the operators may decide to accept the product at the current quality level or improve it through re-parameterization for another iteration. NORDSIECK et al. (2019) envisions a way to automate the re-parameterization process. The parameters for any given iteration are defined by the quality level of the product from the previous iteration and the process parameters previously set to achieve them.

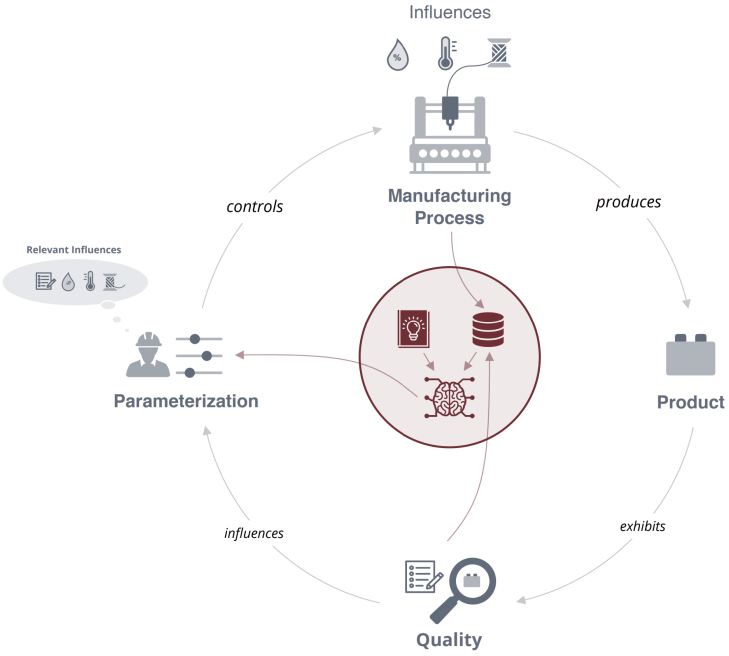


Figure 1.1: Iterative parametrization process of AIPE (NORDSIECK et al., 2021)

The operator knowledge used for parameterization can be represented in a Knowledge graph as a relationship between various quality characteristics and process parameters, and NORDSIECK et al. (2019) further defines various representations which can be used to represent them at varying

levels of abstraction. This semantically meaningful information in these Knowledge Graphs can be further represented as low-dimensional vectors called embeddings, which could be incorporated into a downstream task that involves predicting the next set of parameters to obtain better results.

1.2 Motivation

In order to incorporate the knowledge stored in the form of a knowledge graph, knowledge graph representation or embeddings have been used extensively in many practical applications and research. These embeddings represent nodes and edges and the semantic meaning they carry through the relationship. Most Knowledge graph information has been viewed as a method to store data from different sources in a heterogeneous structured format. The embedding generated from these graphs are representations of facts presented in the knowledge graph as low-dimensional vectors. However, our use case of the AIPE project uses a Knowledge graph to model operator knowledge. Operator knowledge is a tacit resource and one that is obtained through years of experience and investment. This resource is not just facts but a representation of a knowledge-backed decision-making system. Procedure Knowledge graphs are used to represent these operator knowledge as connections between the process parameters and quality characteristics.

Procedural knowledge graphs are different from the widely available Open Knowledge Graphs. These knowledge graphs have complex, chained binary, n-ary, and reified relations. Together, the neighbourhood reconstruction process becomes challenging for any model which learns embeddings from the procedural data representations. The quality of the embeddings is vital to our parameter prediction system. The prediction's quality determines the quality of the products produced in the next iteration. So, we posit using state-of-the-art embedding models to capture the graph's node features and relation features. Currently, in literature and practice, these embedding models, which are used to learn dense low-dimensional vectors, are trained over fact-based Knowledge graphs with different entity types and relation types. Some of these models perform extensively in capturing the neighbourhood structures when trained on available Open knowledge graphs. In the same context, we would like to

know how these embedding models perform when the knowledge graph is not just general knowledge facts but procedural information from the manufacturing process.

In this thesis, we aim to evaluate the use of Graph Neural Networks(GNN)-based embeddings learnt in the context of procedural knowledge graphs. Graph neural networks have recently been at the forefront of knowledge representation learning. Known for their expressiveness and performance when compared to other baseline models, Graph Neural networks have been seen to outperform many baseline models like translation and semantic models in Link prediction or Entity classification tasks. These models are known to effectively model the local neighbourhood of a node and have the flexibility to accommodate a large number of node features and edge features to learn from them through message passing. Most performance indicators in the literature for all embedding models are against known Open Knowledge graph ontologies. Here, we would like to gauge the performance of GNNs in the context of procedural knowledge graphs, which differ in data sizes, atomicity and graph sub-graph structures.

Our thesis evaluates the learned embeddings from some popular Graph neural network models over the procedural graphs. Here, we would implement the Graph convolutional network, Relational Graph convolutional network and Relational Graph Attention network models for training the embeddings over a procedural graph. The procedural graph will be implemented in differing levels of abstraction, and the GNN models mentioned above will be trained over these representations individually. Additionally, we have introduced a new evaluation metric, $Matches@k$, which suits our prediction task designed over the procedural graph to compare the GNN models against other state-of-the-art translational, semantic matching and Random walk-based models.

1.3 Research Questions

We address the performance of Graph neural networks in the context of the procedural knowledge graph as following research questions:

- **RQ 1**: How do Graph Neural Network perform over a procedural knowledge Graph when compared against other state of the art baseline models?
- **RQ 2**: How does the GNN model configurations affect performance against different representations?
- **RQ 3**: Among the GNN-based models, how does Multi-relational models perform against homogeneous models??

1.4 Structure of this thesis

Chapter 2 of this thesis will go into the basic concepts behind Knowledge Graphs, Knowledge Graph Embeddings and the short conceptual primer of Graph Neural Networks. In the following Chapter, we discuss other topics on Knowledge graph related to use in Manufacturing, embedding models available in literature. In the Chapter 4 we will discuss the Graph neural network models which will be used over our procedural graph in our experiments. Here, we would also describe the basic concepts and architecture used for training these models. Post this, we would look at the evaluation, experiment setup , metrics and the actual experiment results which we would use to answer our research questions. Finally, we will summarize our thesis with the conclusion drawn from the results and propose future work which will fill in the gaps we found during our thesis.

2
Background

This section gives a short primer on basic concepts related to this thesis work. In the first section, we discuss Knowledge graphs, their history and a formal definition from the literature. We then talk about Knowledge embedding and its parallels in natural language processing. Here we introduce different knowledge embedding techniques but go into the details in the Related work chapter. The last section of this chapter introduces the Graph neural network, its structure and key concepts such as pooling and message passing. The foundational concept of Graph neural networks forms the basis of our models, which we discuss in the Methods chapter of our thesis.

2.1 Knowledge Graphs

In recent years, Knowledge Graphs(KG) have emerged as a common-sense method of representing structured knowledge in the form of an abstraction that encompasses data from different sources. It is a knowledge base represented as a graph that provides ease of interpretation and inference over facts. Knowledge pertaining to a specific domain can be represented as Knowledge graphs that are then used as input in downstream tasks using machine learning models to get better prediction results.

2.1.1 A brief history of Knowledge Graphs

The concept of representing knowledge as a graph has a long history, spanning several decades. However, the term "Knowledge Graph" gained popularity after it was reinvented by Google in 2012 (EHRLINGER and WÖSS, 2016). Google introduced the term with the intention of improving the

understanding of search queries by gaining insights into the meaning of words. Instead of treating words as mere combinations of letters, the goal was to represent them as real-world objects, such as people, movies, or houses, with attributes and relationships to each other. By collecting information and mapping the relationships between these objects (entities), Knowledge Graphs provide a better understanding of the connections in the real world. This is particularly valuable because language can be ambiguous and lead to misunderstandings, while Knowledge Graphs offer a more precise and structured representation of knowledge.

Before the use of Knowledge Graphs as a structured methodology of storing real world facts, the commonly used methods of storing and handling knowledge were Knowledge bases and Ontology.

According to GRUBER (1989), an ontology refers to a formal and explicit representation of a conceptual framework. The term "ontology" is derived from the field of philosophy, where it refers to a thorough and systematic analysis of the basis of existence. In the context of AI systems, the notion of existence is contingent upon its representation. Knowledge bases, conversely, pertain to repositories or databases that house structured information and factual data on the world. They typically include a collection of assertions and statements organized in a systematic manner(RUSSELL, 2016). While Knowledge Graphs and Knowledge Bases are concerned with storing and representing knowledge, Ontologies primarily focus on defining the structure and vocabulary for knowledge representation.

EHRLINGER and WÖSS (2016) defines a knowledge graph as a combination of Knowledge bases and a reasoning engine, with other essential characteristics of collection, extraction and integration of information from different sources.

2.1.2 Formal Definition of Knowledge Graphs

Formally, a Knowledge Graph (KG) G can be defined as a directed labeled graph that consists of a set of triples T, given by $T \subseteq E \times R \times (E \cup L \cup C)$, where:

- *E* represents the set of resources referred to as entities. An entity can represent a real-world object or an abstract concept. Each entity is uniquely identified by a Uniform Resource Identifier (URI).
- *R* represents the set of relations or properties that define the connections between entities in the KG. Relations capture the relationships or associations between entities.
- *L* represents the set of literals, which are values associated with entities or relations. Literals can take various forms such as text, dates, numbers, images, or other data types.
- *C* represents the set of semantic types or classes of entities. Semantic types classify entities into specific categories or classes based on their characteristics or attributes.

In summary, a KG is a graph structure where entities, relations, literals, and semantic types work together to represent knowledge. Entities represent real-world objects or abstract concepts, relations capture connections between entities, literals represent values associated with entities or relations, and semantic types classify entities into specific categories. This structured representation enables the modeling and organization of knowledge in a comprehensive and interconnected manner (BISWAS, 2023).

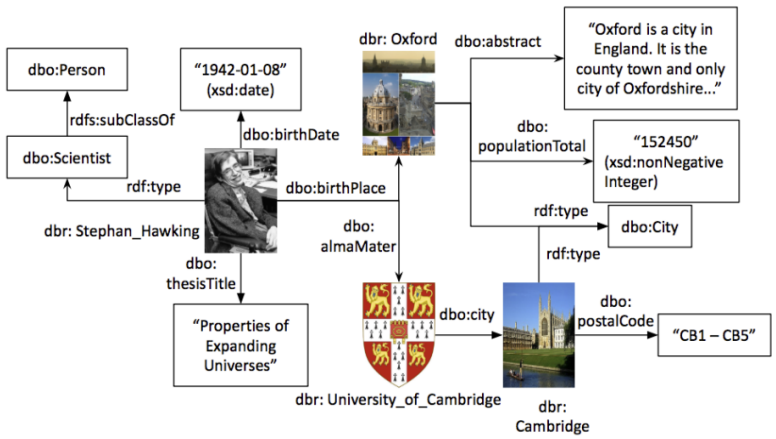


Figure 2.1: Example of a Knowledge Graph extracted from DBpedia adapted from BISWAS (2023)

The facts in a Knowledge Graph are stored as a collection of triples which is an ordered set containing (h, r, t) where h, $t \epsilon E$ and $r \epsilon R$. h denotes a head or source entity and t denotes a tail or target entity. r denotes the relationship between the two entities. The Triple $('UniversityofCambridge','city', 'Cambridge')$ is an example from the figure 2.1 where directed edge enables us to identify the source and target entities.

Here, we define Open World and Closed World Assumptions (HOGAN et al., 2021). The concept of Open World Assumptions posits that the facts present in a Knowledge Graph are considered to be true, but the facts that are not observed within the graph may either be untrue or simply absent. During an Open World Assumption training session, negative sampling is carried out with heuristics that make assumptions based on a local closed world. In the context of the Closed World Assumption, all further facts that are not observable within the Knowledge Graph are postulated to be entirely incorrect.

Categories of Knowledge Graphs

Knowledge Graphs can be categorized broadly as Open or Enterprise Knowledge Graphs based on their availability to the public. Open KGs are publicly available and can be accessed by everyone. These form the basis of many research papers and open access search engines. Some of the widely known Open KGs are DBpedia (BIZER et al., 2009) , Freebase (BOLLACKER et al., 2008), YAGO (HOFFART et al., 2013), Wikidata (VRANDEČIĆ, 2012), BabelNet (NAVIGLI and PONZETTO, 2012) and so on. These KGs cover multiple domains offer multilingual lexicalizations extracting their input data from either Wikipedia or are collated by active community volunteers. Open KGs have also been published for specific domains such as media (RAIMOND et al., 2014), life sciences (DIMITROV et al., 2020) and travel (ZHANG et al., 2020) etc.

Enterprise Knowledge Graphs are typically owned by a company which are used for their internal purposes or are used for their commercial use cases. These graphs are constructed for industries which imbibe large amount of connected data such as Web search (SHRIVASTAVA, 2017), commerce (KRISHNAN, 2018), social networks (NOY et al., 2019) and are used as inputs

to recommendation systems, search , advertising , business analytics, automation etc.

2.2 Knowledge Graph Embeddings

Knowledge Graph Embeddings are low-dimensional continuous vector representation of entities and relations of a Knowledge Graphs (JI et al., 2021). The objective of Knowledge graph Embeddings is to capture the structure of the knowledge and the semantic information of the entities and relations. The basic idea of the embeddings is inspired from the language model concept of Word2vec.

2.2.1 Basics of Embeddings

A language model learns the probability of word co-occurrences from a text corpus, based on the task either being the likelihood of a given word based on a sequence of words or the probability of a word given a sequence (MANNING and SCHUTZE, 1999). In the context of Neural Langauge Models, embeddings are continuous low-dimensional representations of discrete variables, here words. By encoding the words in low dimensional representation space, semantically similar words appear closer in the embedding space (BENGIO et al., 2000). When considering a vast corpus, the most common method of representing categorical variables - One hot encoding or the probabilistic approach of the Statistical Langauge model, but this suffers from the curse of dimensionality.

Word2Vec

Word2Vec embedding is a popular technique that uses a shallow neural network to predict the context of the word defined by a sliding window of a given amplitude. MIKOLOV et al. (2013) defines two different architectures to train these embeddings, namely Continuous Bag of Words(CBOW) and Skip-gram. In CBOW, the current word is predicted from the available context of words defined by the sliding words. It is important to note here that the order of the context words does not have any influence on the prediction process. On the other hand, Skip-gram takes the current word as input and is trained to predict the context words. Skip gram is more

efficient as it requires less training data and performs better at predicting non-frequent words, whereas CBOW performs better with repeated words.

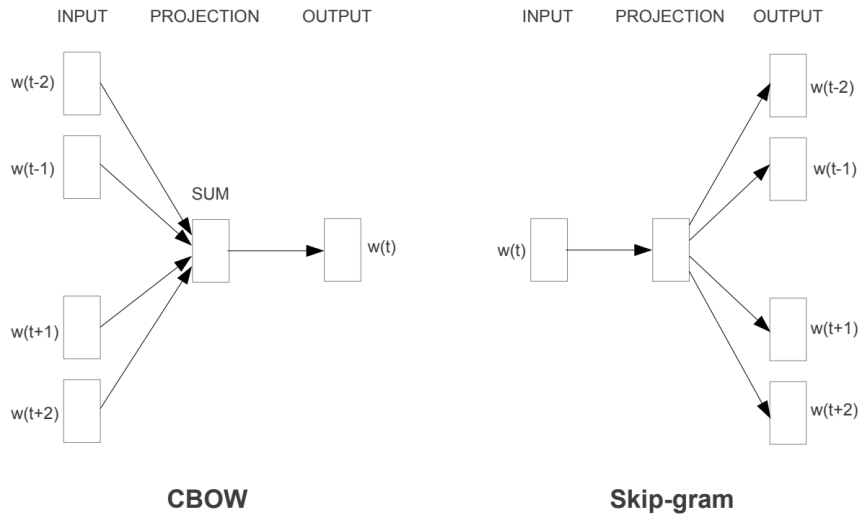


Figure 2.2: CBOW and Skip gram architectures from (MIKOLOV et al., 2013)

In Word2Vec training, the optimization problem revolves around adjusting the word embeddings to maximize the likelihood of correctly predicting the context words or a target word. This optimization problem is typically formulated as a maximum likelihood estimation (MLE) task. Given a training corpus, the objective is to learn the word embeddings that maximize the probability of the observed context-target pairs. This probability is calculated using the softmax function, which assigns higher probabilities to the correct context words and lower probabilities to other words in the vocabulary. To efficiently solve this optimization problem, Word2Vec employs negative sampling. Instead of considering the entire vocabulary for each context-target pair, negative sampling randomly selects a small number of negative (non-context) words. The objective is to differentiate the true context words from these negative samples.

With a significant amount of unannotated data, the Word2vec model learns syntactic and semantic relationship between words. Model yields a embedding vector equal to the size of the hidden layer for each word, with remarkable linear relationships for example $vec("king") - vec("men") + vec("woman") = vec("queen")$.

2.2.2 Parallels in Knowledge Graph Embeddings

Knowledge Graph Embedding methods, analogous to Word2Vec, generate a vector for the elements of Knowledge graphs. These embeddings are such that the latent semantic properties of the knowledge graph are captured: similar entities and similar relations will be represented by similar vectors (PALMONARI and MINERVINI, 2020).

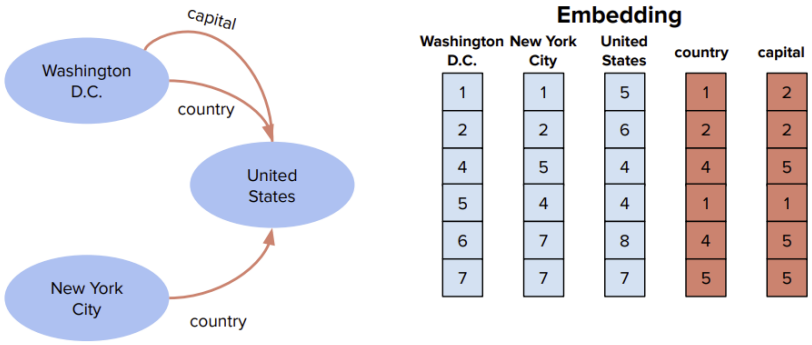


Figure 2.3: Embedding methods generating representations for entities and relations adapted from (PALMONARI and MINERVINI, 2020)

Some embedding methods directly inspired by the Word2vec are DeepWalk (PEROZZI et al., 2014) and node2vec (GROVER and LESKOVEC, 2016). These methods extract sequences of nodes from the Knowledge Graphs by performing random uniform walks. The sequences of nodes are fed to CBOW or Skip-gram models as text to construct embedding vectors. These embedding techniques treat the Knowledge graphs as homogeneous graphs with no differentiation between Entities or relations.

2.3 Knowledge Graph Embedding Techniques

There are four key aspects through which Knowledge Graph Embeddings are generated:

- **Representation space:** This refers to the manner in which relations and entities are represented. It encompasses various approaches such as pointwise space, manifold, complex vector space, Gaussian distribution, and discrete space.

- **Scoring function:** It involves measuring the plausibility of factual triples. Scoring metrics are typically categorized into distance-based and similarity-based matching scoring functions.

- **Encoding models:** These models are employed to represent and learn relational interactions. Current research in this area explores different encoding models, including linear/bilinear models, factorization techniques, and neural networks.

- **Auxiliary information:** This aspect involves incorporating additional information, such as textual, visual, and type information, into the embedding methods.

A significant number of embedding models in the literature provide unique combinations and techniques in the above-mentioned four aspects to propose a new modelling paradigm. Broadly, the Knowledge Graph Embedding techniques can be divided as (primarily based on their variation in the scoring functions): Translational Models, Semantic Matching Models and Neural Network models (WANG et al., 2017).

Translational models are distance-based models where the translation distance between the head and tail in the triplet is assumed to be defined by the relationship. Semantic matching models use similarity-based scoring functions to measure the existence of the fact matching the semantics of entities and relations. The neural network model implements the scoring function mentioned above through their hidden layers and captures complex patterns because of their non-linearity. More detailed information on the techniques and the various models are explained in the Related Works Chapter.

2.4 Graph Neural Networks

The ubiquitous nature of graphs as a choice of representation of many real-world objects, including text and images, has led to the decade-long research on the development and use of Graph data-specific deep neural networks called Graph Neural Networks (GNNs).

Graph Representation for Deep Neural Networks

Graphs, as explained before in the context of knowledge graphs, represent relations(edges) between a collection of entities(vertex or node). Intuitively, this representation does not lend itself to being compatible with a neural network which generally takes in a rectangular array or a matrix as input. The three components, i.e., entities, relations and connectivity, must be defined as matrices to facilitate the application of neural networks over graphs.

For the context of this discussion on Graph neural networks, we will call entities as nodes and relations as edges as represented in the Graph. A simple way to represent the entities or nodes is to have a matrix where the i^{th} row is the feature vector representing the $node_i$. The same method can be adopted to represent the edges or relations (SCARSELLI et al., 2008).

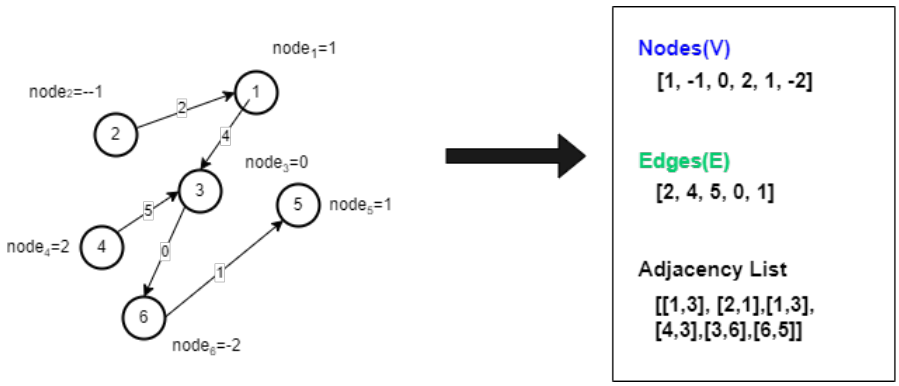


Figure 2.4: Basic Input format for GNNs

However, the notion of representing graph connectivity can be more complicated. A straightforward way is to represent the connectivity as an adjacency matrix, but there can be millions of nodes, and the number of edges per node can be highly variable. This will lead to an adjacency matrix which is very sparse and highly space inefficient.

Another problem with the use of adjacency matrices is that they can have equivalent versions which can be obtained by permutation of the nodes, and these different inputs can lead to varying results when processed through a deep neural network which is not typically permutation invariant (HAMILTON, 2020).

One efficient way of representing the connectivity is to use adjacency lists instead of matrices which contain only edges which exist in the graph. Each element of this list will be a tuple with the indices of nodes i and j, and k^{th} position in the list would denote the edge e_k.

2.4.1 GNN Layer

A GNN is an optimizable transformation on all attributes of a graph (SANCHEZ-LENGELING et al., 2021). These attributes are nodes(vertices) V, edges E and global context U. Global context attribute refers to all properties of a graph as a whole, for example, the number of nodes, longest path and graph domain. A simple GNN layer uses a Multi-layered Perceptron on each of the attributes separately. Therefore, MLP is applied to obtain a learned vector over each node vector. We can stack these GNN layers together, similar to neural network modules or layers.

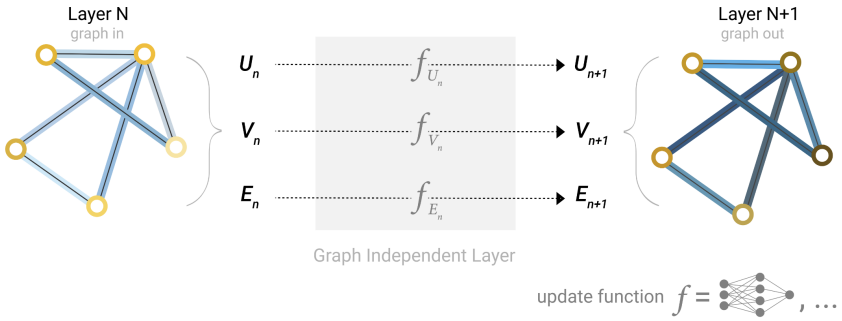


Figure 2.5: A single layer of GNN. A graph input in the form of the 3 components-Nodes(V), Edge(E), global context (U) are fed into the different MLP. Subscript n indicates output of nth layer from SANCHEZ-LENGELING et al. (2021)

.

We can represent the output graph of a GNN with the same adjacency list and the same number of feature vectors as the input graph because a GNN does not change the connectivity of the input graph. However, because the GNN modified each node, edge, and global-context representation, the output graph has updated embeddings.

Pooling or Aggregation for downstream tasks

Since individual components of the Graph are processed individually by their respective MLPs, the information about a Neighbourhood or a Subgraph is stored separately in nodes and edges that belong to it. When implementing graph-based downstream tasks such as node classification, an aggregation or pooling operation is performed where the embeddings are gathered from all the edges connected to the context node and the context node itself. These undergo user-defined aggregation or pooling operations, yielding a new context node embedding that can be further used for classification. Figure 2.6 shows the pooling operation, which is represented by $\rho_{E_n->V_n}$ over the edges and context nodes.

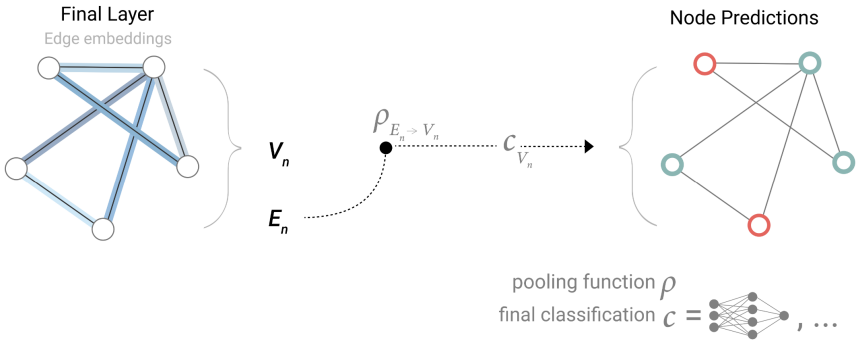


Figure 2.6: Pooling of information from edges to nodes for node prediction from SANCHEZ-LENGELING et al. (2021)

A similar operation can be done to improve edge level predictions. The information on a binary edge can be obtained by pooling over the nodes connected to these edges.

Figure 2.8 shows the end to end GNN pipeline in the case of a prediction task. The graph components to a GNN block containing stacked GNN layer is transformed to obtain learned embedding. The learnt embedding can be further utilised in any downstream tasks. Any differentiable model can simply take the place of the classification model c in our examples, or a generalized linear model can be used to convert it to multi-class classification.

It is important to note that in the simplest form of the GNN formulation, the connectivity does not get used in any of the GNN layer. Each node

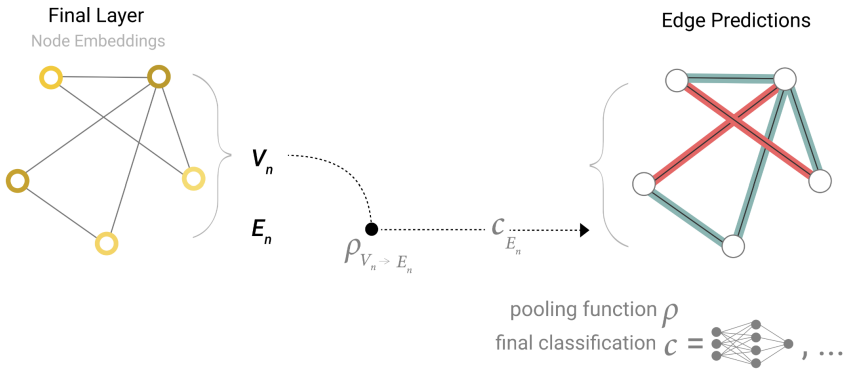


Figure 2.7: Pooling of information from nodes to edges for edge prediction from SANCHEZ-LENGELING et al. (2021)

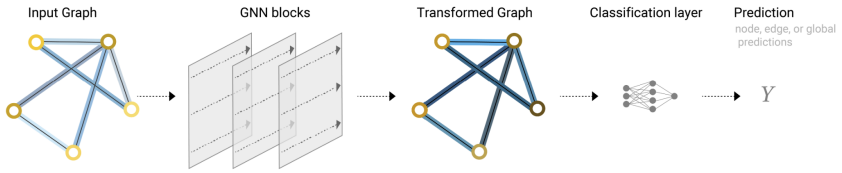


Figure 2.8: An end-to-end prediction task with a GNN model from SANCHEZ-LENGELING et al. (2021)

or edge is processed independently without considering connection until the pooling operation is done to improve embedding for the downstream prediction task.

2.4.2 GNN Message passing

In order to produce more accurate prediction results, it is intuitive that we merge the concept of pooling into the GNN layer. This would mean that the embedding learned through GNN layers with pooling will be aware of the graph connectivity. GILMER et al. (2017) called this methodology as message passage which is defined as the exchange of information between edges and nodes and thereby influences the updating of the embedding values.

As shown in the Figure 2.9 , for a specific target node, the neighbourhood information is aggregated. In this case, the target node A aggregates the

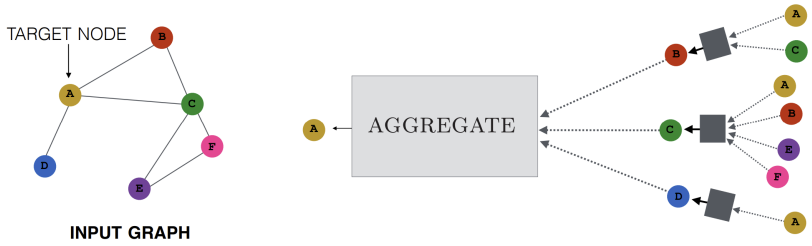


Figure 2.9: Message Passing - Target node aggregates messages from local neighbourhood from HAMILTON et al. (2017)

information from the neighbourhood nodes B, C and D which in turn have information aggregated and updated from their respective neighbourhoods. This visualization represents a 2- layer version of message passing model which is obtained by stacking two GNN layers. Let us now formulate this idea.

For a node $u \epsilon V$, the hidden embedding $h_u^{(k)}$ is updated according to the information in the neighbourhood $N(u)$. The message passing update of the target node u can be defined by :

$$h_u^{(k+1)} = UPDATE^{(k)}\left(h_u^{(k)}, AGGREGATE^{(k)}\left(\{h_u^{(k)}, \forall v \epsilon N(u)\}\right)\right) \tag{2.1}$$

$$= UPDATE^{(k)}\left(h_u^{(k)}, m_{N(u)}^{(k)}\right) \tag{2.2}$$

where UPDATE and AGGREGATE are arbitrary differentiable function such as neural networks and $m_{N(u)}$ defines the aggregated 'message' from local neighbourhood $N(u)$. The UPDATE function combines the previous embeddings $h_u^{(k-1)}$ and the message $m_{N(u)}^{(k)}$ of node u to obtain the new embeddings. The AGGREGATE function takes a set of neighbourhood embedding. The GNN designed this way are permutation equivariant.

The equations 2.1 and 2.2 define in an abstract manner the framework which defines GNN layer. A more implementable version of the equations is to instantiate the UPDATE and AGGREGATES with Weights and summation of hidden states.

The basic GNN message passing can be defined as follows:

$$h_u^{(k)} = \sigma\left(W_{self}^k h_u^{(k-1)} + W_{neigh}^k \sum_{v \epsilon N(u)} h_v^{k-1} + b^{(k)}\right) \tag{2.3}$$

where W_{self}^k is the trainable weights on the node, W_{neigh}^k are the weights applied over the neighbourhood. σ denotes the element-wise non-linearity while $b^{(k)}$ denotes the bias term. Other sophisticated methods which will be defined in our approach will apply a modified and more effective version of the UPDATE and AGGREGATE functions.

3
Related Work

3.1 Knowledge Graphs in Manufacturing

Research on using knowledge graphs in manufacturing has significantly increased in recent times. Most Knowledge Graphs were modelled by domain experts in the form of an ontology and infused with data about manufacturing. The primary objective of these Knowledge Graphs is to integrate data from various sources and build Heterogeneous models (BUCHGEHER et al., 2021). It integrates contextual data, such as equipment configuration, with operational data, such as process data and events (DUAN and CHIANG, 2016). AGGOUR et al. (2019) explore the application of Knowledge graphs in the field of Additive Manufacturing. They propose the utilization of a federated multi-modal big data storage platform to construct Knowledge Graphs, which serve as the underlying structure for a data analysis platform. HE and JIANG (2019) define Knowledge graphs storing manufacturing knowledge and production problems. A platform constructed on top of the Knowledge graph is used to link production problems to related manufacturing knowledge. In the case of KATTEPUR and P (2019), an autonomous system that utilizes knowledge graph queries for robotic action planning. The system leverages the knowledge graph to enable the robot to autonomously generate action plans based on its understanding of the environment and task requirements. A similar kind of automation is proposed by NAYAK et al. (2020), where knowledge graphs are utilized to generate test cases by leveraging the semantic relationships and dependencies between software artefacts. The knowledge graph represents the underlying domain knowledge and captures the relationships between various elements, such as requirements, design specifications, and code components.

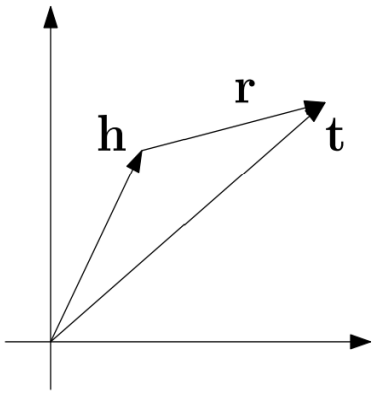
In all of the methods mentioned above, knowledge graphs are used to depict foundational knowledge and are incorporated into compact vector representations that can be used to extract rules BORDES et al. (2013a), WANG et al. (2017) for easier integration into various downstream tasks and learning systems PORTISCH et al. (2022). While knowledge graphs often contain factual and conceptual knowledge, such as terminology, classification, and generalizations KRATHWOHL (2002), NOY et al. (2019), research on procedural knowledge, which encompasses techniques, their application, and contextual understanding, has been limited. This type of knowledge is heuristic in nature and is typically acquired through years of experience and expertise. NORDSIECK et al. (2021) and NORDSIECK et al. (2022a) provide the basis for data-based extraction of procedural knowledge from production data. NORDSIECK et al. (2021) showcases a methodology of dealing with (re-)parameterization in the manufacturing process to produce products which satisfy certain quality criteria. NORDSIECK et al. (2019) further presents the idea of modelling patterns for procedural knowledge graphs capable of representing knowledge at different levels of abstraction.

3.2 Knowledge Graph Representations

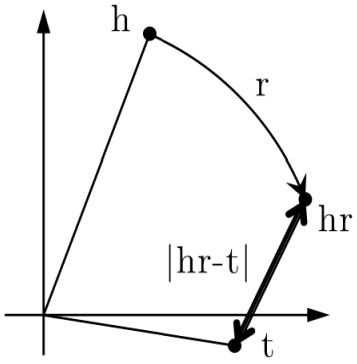
While Knowledge Graphs (KGs) are efficient in representing structured data, their symbolic nature often poses challenges when it comes to manipulation. To address this issue, a fundamental approach is to embed the entities and relations of a KG into continuous vector spaces. This embedding process aims to streamline manipulation tasks while still maintaining the intrinsic structure of the KG intact. The embedding of the entities and relations are in themselves used in different kinds of task such as Knowledge Graph completion (BORDES et al., 2013b) (WANG et al., 2014), relation extraction(WESTON et al., 2013), entity classification(NICKEL et al., 2011) and entity resolution(GLOROT et al., 2013). Knowledge Graph embedding workflow typically has three major steps : (i) representation of relations and entities, (ii) implementing a scoring function, and (iii) learning the representations. Different embedding techniques have been defined by varying approaches to each of these steps.

Translational Models

Translational models are a popular approach in Knowledge Graph Embeddings (KGE) that aim to represent entities and relations in a continuous vector space. These models capture the translation-based assumption, which states that the relationship between entities in a KG can be modelled as a translation operation. TransE (BORDES et al., 2013b) is a widely used translational model that represents relations as translation vectors between entity embeddings. The model assumes that the relation vector added to the head entity embedding is close to the tail entity embedding.



(a) TransE (WANG et al., 2017)



(b) RotatE models r as a rotation in complex plane (WANG et al., 2017)

Other extensions (WANG et al., 2014), (LIN et al., 2015) extend TransE model to accommodate complex relations using relation-specific hyperplanes or projection matrices. In BoxE (ABBOUD et al., 2020), each relation is represented as a hyperrectangle (or box) in the embedding space. The head and tail entities are also represented as points in the same space. The model assumes that the relation boxes enclose the points representing valid entity pairs for that relation. In the case of RotatE (SUN et al., 2019), each entity and relation in the knowledge graph is represented as a complex-valued vector. The key idea behind RotatE is to model the relation between entities as a rotation operation in the complex plane. The rotation is applied by multiplying the head entity embedding by a complex-valued rotation vector associated with the relation, aiming to align it with the tail entity embedding.

Semantic Matching Models

Semantic Matching models are another popular approach in Knowledge Graph Embeddings (KGE) that aim to capture the interactions between entities and relations using semantic similarity. These models enable more expressive representation of complex relationships in a KG. RESCA L(NICKEL et al., 2011) represents a relation r as a full-rank $d \times d$ matrix M and entities as d- dimensional vectors e. DistMult (YANG et al., 2014) is a simplified version of RESCAL that assumes diagonal matrices for relations. It models the interactions between entities and relations using a bilinear product that results in a diagonal matrix.

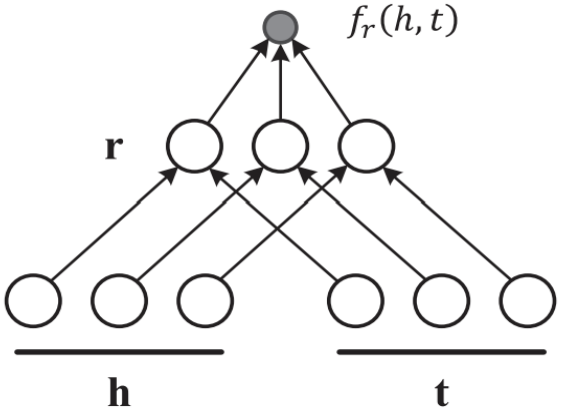


Figure 3.2: DistMult (WANG et al., 2017)

ComplEx (TROUILLON et al., 2016) extends DistMult by utilizing complex-valued embeddings to capture more expressive interactions between entities and relations. It represents both entities and relations as complex-valued vectors and computes the interactions using the conjugate dot product. ComplEx can effectively model asymmetric and symmetric relations as well as the compositionality of relations.

Neural network based models

Neural network-based models for Knowledge Graph (KG) embeddings utilize neural network architectures to learn continuous vector representations of entities and relations in KGs. These models leverage the expressive power of neural networks to capture complex patterns and interac-

tions in the graph structure. Linear / bilinear models can also be modeled using neural networks. Generally entities and relations are fed to neural network which computes a semantic matching score. One such example is ConvE DETTMERS et al. (2018) which employs a 2D convolutional neural network architecture to capture the local neighborhood information of entities and relations. It operates on the 2D matrix representation of triples (head entity, relation, tail entity) and uses convolutional filters to learn feature maps. ConvE combines these feature maps to generate entity and relation embeddings. SimplE is a model that incorporates both forward and backward embeddings for relations in KGs. It uses two sets of embedding vectors to represent relations in both directions. By considering the embeddings of head and tail entities along with these relation embeddings, SimplE captures different types of relation patterns.

Other Models

A few Knowledge Graph Embedding models use language modelling approached to learn representations of vertices in graph and are usually used over homogeneous graph methods. Models like DeepWalk (PEROZZI et al., 2014) and Node2vec (GROVER and LESKOVEC, 2016) which perform random walks on a neighbourhood and use language models embed these paths. These approach mainly preserving the network neighbourhood of nodes without distinctions between the type of nodes or relations.

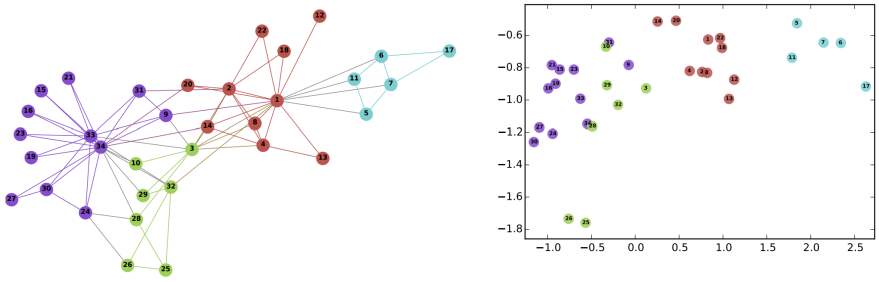


Figure 3.3: DeepWalk captures the structure of the graph in Two Dimensional Embedding space, from (PEROZZI et al., 2014)

RDF2Vec(RISTOSKI and PAULHEIM, 2016) is one such approach that aims to learn continuous vector representations (embeddings) of entities and relations in a Knowledge Graph (KG) using the RDF (Resource Descrip-

tion Framework) graph structure. It leverages techniques from the field of word embeddings to capture semantic relationships and similarities between entities and relations in the KG.The RDF2Vec approach follows a similar idea to word2vec, a popular method for learning word embeddings from large text corpora. In word2vec, word embeddings are learned based on the co-occurrence patterns of words in sentences. Similarly, RDF2Vec learns embeddings by considering the co-occurrence patterns of entities and relations in the RDF graph.

4
Methods

4.1 AIPE Dataset

AIPE Dataset consists of procedural data from the Fused-diffusion process explained earlier in Chapter 1. Similar to most manufacturing use cases, procedural knowledge is implicit in the parameterization of the machinery for producing quality products. The Expert operator sets parameters for the additive manufacturing technique to print models by squeezing the melted raw material through a nozzle to deposit it layer-wise. As such, the knowledge can be described as a relationship between the process input parameters set for the manufacturing process and then the quality characteristics of the product obtained. This process usually involves how and in what order the expert has tweaked the process parameter to obtain the required best product mitigating the quality defects in each iteration. In total, there are 46 process parameters and 13 quality features representing various production defects. These process parameters have distinct ranges, while the quality features are of two types: boolean values and ordinal features ascending from zero. The dataset consists of approximately 500 such process iterations of process parameters and associated quality characteristics.

4.1.1 AIPE Dataset Representation

NORDSIECK et al. (2022a) propose different methods of describing operator knowledge in the form of knowledge graphs. These methods provide different levels of abstraction in the representations. Lower levels of abstraction contain just interacting entities such as process parameters and

quality characteristics, while higher abstraction adds more information, like including parameter ranges and iteration dependencies.

The different levels of abstraction for representing the operator knowledge can be described as:

Unquantified rules: It is the highest level of abstraction which represents the relation 'implies' between a quality characteristic and a process parameter .i.e the process parameter p needs to be adjusted implied by the quality characteristics q. The triples in this knowledge graph can be represented by $r_\eta = (q, < implies >, p)$, $p \epsilon P$, $q \epsilon Q$ where P is the set of all process parameters and Q is set of all quality characteristics.

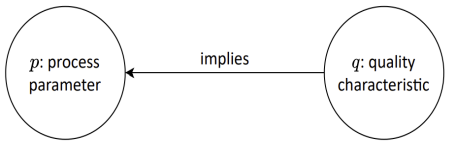


Figure 4.1: Graphical representation of r_η from NORDSIECK et al. (2022a)

Quantified Conclusion: This representation would add more information as to how much the process parameter p would need to be adjusted to attain better quality from the current quality characteristics q.

A fact in this knowledge graph could be represented be with a edge weight $v \epsilon \mathbb{R}$ quantifying the degree of adjustment or could be represented as a chained binary representation like $r_{\hat{\rho},ch} = (q, < implies >, (v, < quantifies >, p))$. The chained binary representation can be modelled

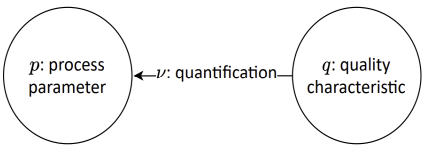


Figure 4.2: Graphical representation of $r_{\hat{\rho},rel}$ from NORDSIECK et al. (2022a)

two different ways. One way is to treat the quantification as an entity but this would impact the semantic nature of the p and q. This causes the relation to become a ternary relation with three entities used to represent a single relationship as shown in the Figure 4.3.

This representation essentially cause indirection in which either p or q are not directly connected to the quantification. The alternative would be to

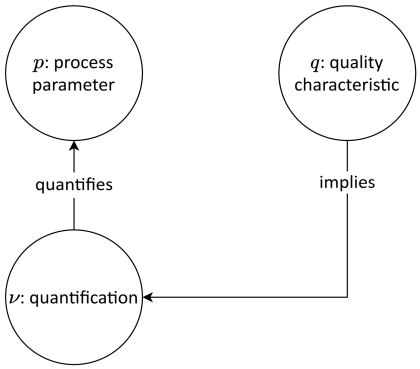


Figure 4.3: Graphical representation of $r_{\hat{\rho},ch,e}$ from NORDSIECK et al. (2022a)

maintain the direct unquantified connection but include connection to the quantification ν from both process parameter and quality characteristics as the modified representation $r_{\hat{\rho},ch,e,\eta}$ as shown in the Figure 4.4.

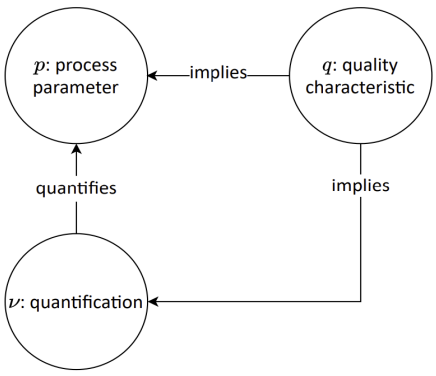


Figure 4.4: Graphical representation of $r_{\hat{\rho},ch,e,\eta}$ from NORDSIECK et al. (2022a)

Another representation method is to represent the n-ary representation as proposed by (NOY et al., 2006). This involves introducing an additional node which signifies the relation between p and q and then connecting the quality characteristic, process parameter and quantification to the relation entity as shown in figure 4.5.

:

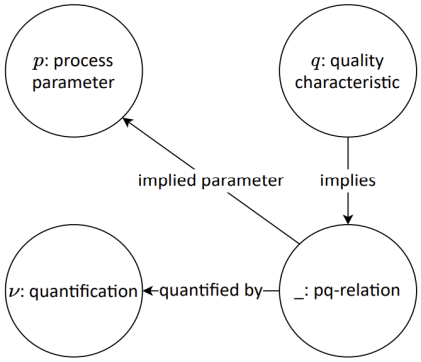


Figure 4.5: Graphical representation of $r_{\hat{\rho},rei,e,\eta}$ from NORDSIECK et al. (2022a)

4.2 Graph Convolution Networks

Convolution is a concept which essentially belongs to the image domain. A filter or kernel, a matrix with random weights, performs element-wise multiplication with the input pixel values. This filter slides over the entire image using a user-defined protocol, obtaining information from all the pixels in the image and thereby learning its weights. This process is called convolution (LI et al., 2021).

A convolution layer may contain many such filters defined by the user to convolve over the image to learn its weight to capture patterns in the image corpus. Since the same filter or kernel is used for all the different parts of the image, the concept of weight or parameter sharing is implemented.

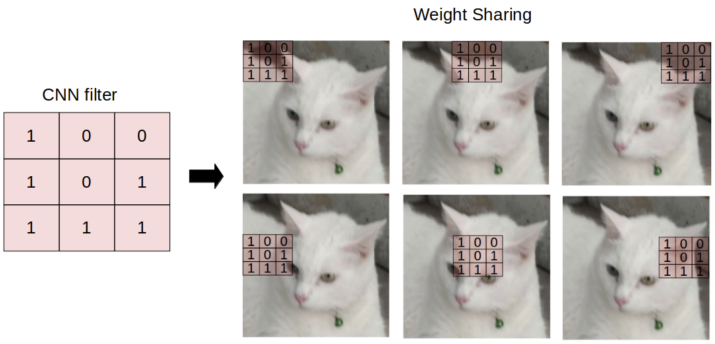


Figure 4.6: The process of convolution from MAYACHITA (2020)

Similarly, it is also a common practise to interpret images as a connected graph where each pixel is a node connected to its neighbouring pixels through an edge. Each node can have additional information such as the RGB values embedded in it.

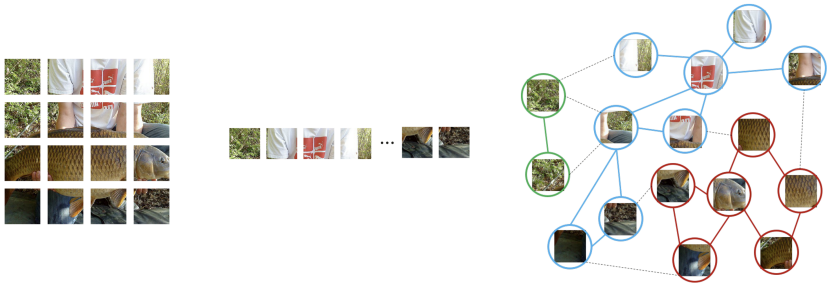


Figure 4.7: Interpreting image as graph from HAN et al.

HAN et al. shows how the graph interpretation enables you to connect nodes not only using euclidean adjacency but also using the similarity of images. It also allows you to segment the image by classifying nodes belonging to the same object.

Corollary to the two definitions of images as graph and convolution for images, a convolution operation can be used over the neighbourhood in a graph structure. The use of convolution operation over graphs is called graph convolution. Like a Convolution neural network (CNN), a graph convolution network (GCN) model learns features for each node, gathering information from the neighbouring nodes using the convolution operation. Weights or parameters are shared across different neighbourhoods in the graphs. Although similar in concept, GCN has few features that suit the permutation equivariance and non-euclidean nature of the graphs since the number of edges and nodes in a neighbourhood may or may not vary or be in a particular order, unlike an image (ZHANG et al., 2019).

Let us consider a Graph $G = (V, E)$ where V and E are a set of Vertices (nodes) and Edges respectively. The input to GCN can be formalised as follows :

- **Input feature matrix X:** feature vectors of length D for node $n \epsilon V$. Therefore, the input feature matrix X is of dimensions $V \times D$.

- **Adjacency Matrix A:** representing the connection in the graph structure in the matrix form.

A simple form of the neural network layer wise propagation rule for GCN can be defined as follows

$$H^{(k+1)} = f(H^{(k)}, A) = \sigma\left(AH^{(k)}W^k\right) \tag{4.1}$$

where $H^{(0)} = X$, $H^K = Z$, Z is the output(embedding matrix) , K is the number of layers, $W^{(k)}$ is the weights in the k-th layer and σ is a non-linear activation functin like ReLU.

There are a few limitations to the above-mentioned simplistic formulation: multiplication with the Adjacency matrix(A) means summing up the node features of the neighbourhood, but the node features already existing in the node are forgotten. We need to consider a self-loop condition which can be fixed by adding an identity matrix to A.

Another limitation is the need to normalize the feature vectors since the Adjacency matrices are not typically normalized. Therefore using $D^{-1}A$, where D is the Diagonal node degree matrix, fixes the problem by taking an average of neighbourhood node features. As proposed by KIPF and WELLING (2016a), a symmetric normalization method is used instead of simple averaging. The new propagation rule now obtained after changes are as follows :

$$H^{(k+1)} = f(H^{(k)}, A) = \sigma\left(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^{(k)}W^k\right) \tag{4.2}$$

with $\hat{A} = A + I$, where I is the identity matrix and $\hat{D}$ is the Diagonal Node degree matrix of $\hat{A}$.

The same matrix form of the propagation rule can be formalized in vector form as :

$$h_u^{(k+1)} = \sigma\left(\sum_{v \epsilon N(u)} \frac{1}{c_{uv}} h_v^k W^{(k)}\right) \tag{4.3}$$

where v indexes the nodes in the neighbourhood of u $N(u)$,$c_{u,v}$ is constant obtained from $\hat{D}^{-1/2}\hat{A}\hat{D}$ for node u and v.

With this configuration of the GCN update rule, we obtain a very stable model in practice owing to the normalization method and an appropriate

orthogonal weight initialization such as Glorot initialization. It outputs meaningful embeddings which reflect the local neighbourhood structure whose similarity or dissimilarity can be observed in their distances in the embedding space.

In our thesis, we will use the Graph convolution network architecture as an example of a homogeneous graph neural network that does not differentiate between edge types. We will also include multiple variants of differing numbers of layers to increase the hops done to aggregate the embedding values during the training process.

4.3 Relational Graph Convolution Network (RGCN)

As discussed in the last section, GCN does not discriminate different edge types between connecting nodes. One of the crucial aspects of Knowledge Graph Embedding is to handle the multi-relational nature of it. Remember the figure 2.3, nodes $'WashingtonDC'$ and $'UnitedStates'$ had two different relations between them - $'capital'$ and $'country'$. In this case, the GCN would be unable to discriminate between the two relations and would not have any embeddings to signify the relations. Relational Graph Convolution solves this problem by learning a relation feature vector for each relation type in the Knowledge graph (SCHLICHTKRULL et al., 2018). While it distinguishes between the two relations $'capital'$ and $'country'$, it also learns meaningful embeddings, which would imply that some relations of the same type are closer to each other than others of a different type in the embedding space.

For the purpose of defining the RGCN architecture(THANAPALASINGAM et al., 2022), we expand the definition of the Graph which was used for GCN and add another element R. We consider Graph $G = (V, E, R)$ where V is the set of all nodes or vertices, E is the set of all edges and R is the set of all edge labels.

The layer-wise propagation rule for the Relational Graph Convolution Network would extend the graph convolutions to include directions of the

edges and message passing for different relations differently. Therefore 4.2 would be modified as follows :

$$H^{(k+1)} = \sigma\left(\sum_{r=1}^{R} \hat{D}_r^{-1/2} \hat{A}_r \hat{D}_r^{-1/2} H^{(k)} W_r^k\right) \tag{4.4}$$

with $\hat{A}_r = A_r + I$, where I is the identity matrix, A_r is the Adjacency matrix for relation $r \epsilon R$, $\hat{D}_r$ is the Diagonal Node degree matrix of $\hat{A}_r$ and W_r^k is the weights for relation r.

The same matrix form of the propagation rule can be formalized in vector form as :

$$h_u^{(k+1)} = \sigma\left(\sum_{r}^{R} \sum_{v \epsilon N^r(u)} \frac{1}{c_{u,r}} h_v^k W_r^k + W_0^k h_u^{(k)}\right) \tag{4.5}$$

where v indexes the nodes in the neighbourhood of u $N^r(u)$,$c_{u,r}$ is constant obtained from $\hat{D}_r^{-1/2} \hat{A}_r \hat{D}_r$ for node u and v, W_0^k are weights for self loop edge (THANAPALASINGAM et al., 2022).

Relational Graph convolution networks are an example of a heterogeneous graph neural network we will use in our thesis. Although the most common and vital relation in our use case is the 'implies' relation between the process parameter and quality characteristics, other relation types like 'quantify' or 'quantified by' exist in the knowledge graph. We would like to see if using RGCN is helpful in the representations with more than one relationship type.

4.4 Graph Attention Networks

Attention is a concept introduced in the domain of Language models by VASWANI et al. (2017). The idea that some parts of the text sequence are more important for predicting a word than others gave rise to the concept of attention. Attention provides coefficients to weigh the importance of words in a text sequence. This idea gave rise to a compelling model called Transformers. One of the benefits of using the attention-based mechanism was its ability to handle inputs of different sizes. When attention is computed from a single sequence of text, it is commonly known as self-attention.

Graph attention networks(GANs) (VELIČKOVIĆ et al., 2017) combine the concept of GNNs and attention. The attention mechanism allows us to

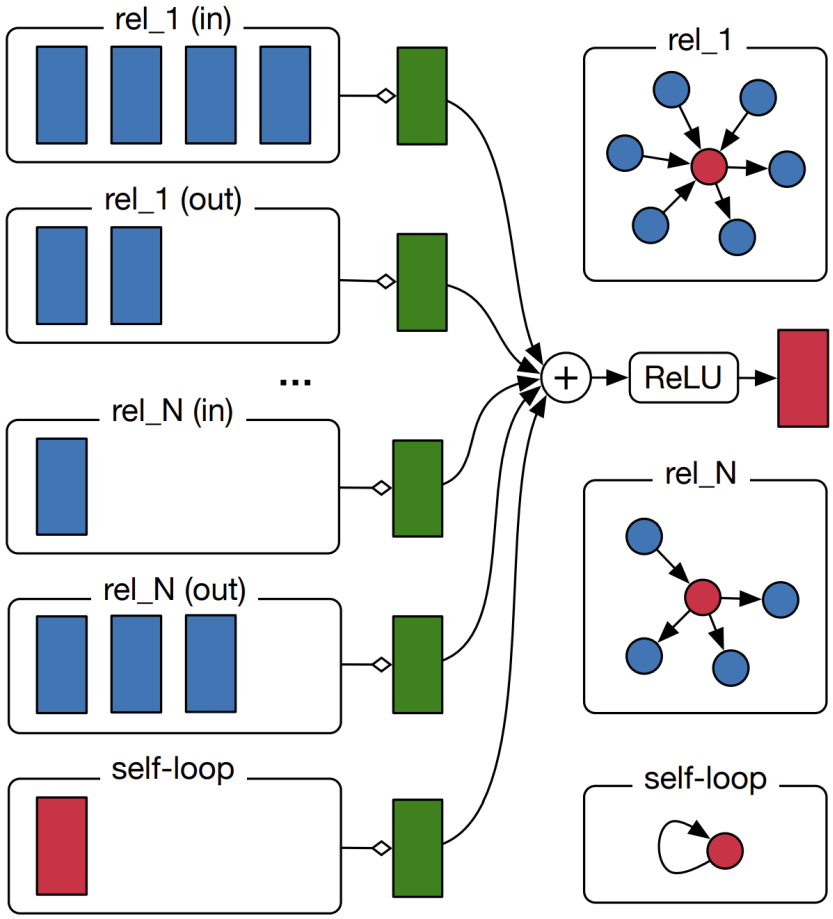


Figure 4.8: Update graph node embedding using RGCN model from SCHLICHTKRULL et al. (2018)

learn an embedding using GNNs but with attention to specific neighbours who might be more important than others. This model is highly parallelizable owing to the simplicity of the self-attention mechanism and can be applied to varying degrees of neighbours of a node.

Let us now consider one of the hidden layer of the GAT network. The input to our layer is a set of node features, $h = \vec{h}_1, \vec{h}_2, ...\vec{h}_N$, $\vec{h}_i \epsilon \mathbb{R}^F$, where N is the number of nodes and F is the number of features in each node. Outputs of the layer can be formulized as $h' = \vec{h'}_1, \vec{h'}_2, ...\vec{h'}_N$, $\vec{h'}_i \epsilon \mathbb{R}^{F'}$,. F' signifies that the output features may potentially be of a different cardinality.

Now the shared self-attention operation is performed over these nodes can be :

$$E_{i,j} = a(W\vec{h}_i, W\vec{h}_j) \tag{4.6}$$

where E is the logit computed by attention mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow R$ and indicates the importance of node j's features to node i. The attention mechanism can be further restricted to include the neighbourhood of node i where j ϵ N_i, Therefore the attention function would look like:

$$\alpha_{i,j} = softmax_j(E_{i,j}) = \frac{exp(E_{i,j})}{\sum_{k\epsilon N_i} exp(E_{i,k})} \tag{4.7}$$

This step would be called masked attention where the attention mechanism is restricted to the neighbourhood thus injecting the graph structure into the computation. The above equation can be now fully expanded by combining the 4.6 and 4.7 as follows (VELIČKOVIĆ et al., 2017):

$$\alpha_{i,j} = softmax_j(E_{i,j}) = \frac{exp(LeakyReLu(\vec{a}^T[W\vec{h}_i||W\vec{h}_j]))}{\sum_{k\epsilon N_i} exp(LeakyReLu(\vec{a}^T[W\vec{h}_i||W\vec{h}_k]))} \tag{4.8}$$

where attention a is represent as a single layer feed forward network and || is a concatenation operation.

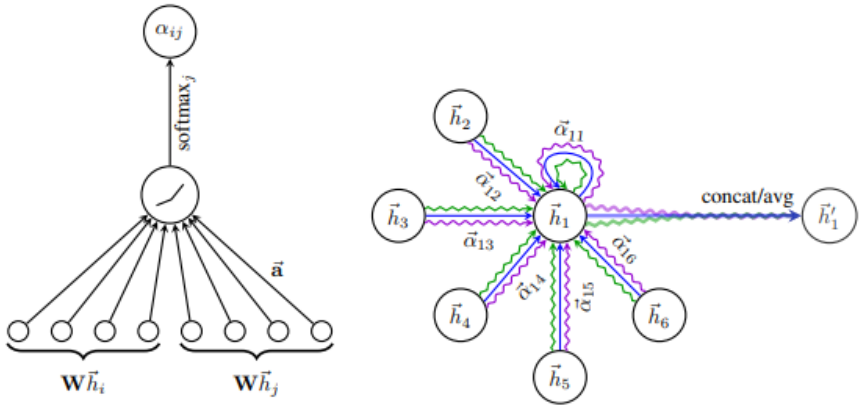


Figure 4.9: **Left:** attention mechanism used. **Right:** computation of attention coefficient α and aggregating it to obtain the final output node features from VELIČKOVIĆ et al. (2017)

The normalized attention coefficient obtain from 4.8 is used to perform a weighted linear aggregation of node features in the neighbourhood to obtain the final feature of the node after applying the non-linearity.

$$\vec{h'}_i = \sigma\left(\sum_{j \epsilon N_i} \alpha_{i,j} W \vec{h}_j\right) \tag{4.9}$$

For a more stable learning process for the self attention mechanism, we can used the multi-head attention technique suggested by VASWANI et al. (2017). Using K independent self attention mechanism and concatenating the resulting output features would transform the equation 4.9 as follows:

$$\vec{h'}_i = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \epsilon N_i} \alpha^k_{i,j} W^k \vec{h}_j\right) \tag{4.10}$$

4.4.1 Extending GAT to Relational GAT

An extension of the Graph attention networks is to accommodate the multi-relational nature of Knowledge Graphs (BUSBRIDGE et al., 2019). Consider the same formal definitions as GAT. The input to our layer is a set of node features, $h = \vec{h}_1, \vec{h}_2, ...\vec{h}_N$, $\vec{h}_i \epsilon \mathbb{R}^F$, where N is the number of nodes and F is the number of features in each node. Outputs of the layer can be formulized as $h' = \vec{h'}_1, \vec{h'}_2, ...\vec{h'}_N$, $\vec{h'}_i \epsilon \mathbb{R}^{F'}$,. F' signifies that the output features may potentially be of a different cardinality. Here,we also include the set of relation R as input to the layer.

To differentiate between the hidden features influenced by the various relation, we introduce an intermediate representation where $\vec{g}_i^{(r)} \epsilon \mathbb{R}^{F\prime}$ for relation $r \epsilon R$. Therefore the update rule to the intermediate relation based representation in the form matrices can be defined as follows:

$$G^{(r)} = HW^{(r)} \epsilon \mathbb{R}^{N \times F'} \tag{4.11}$$

where $G^{(r)} = [\vec{g}_1^{(r)} \vec{g}_2^{(r)} ... \vec{g}_N^{(r)}]$ is the intermediate representation feature matrix and $W^{(r)} \epsilon \mathbb{R}^{F \times F'}$ is the learnable weights for the relation r and H = $\vec{h}_1 \vec{h}_2 ....\vec{h}_N$ is node feature matrix. The attention mechanism which was defined in the Equation 4.6 can be now modified to include the relational aspect as follows :

$$E_{i,j}^{(r)} = a(g_i^{(r)}, g_j^{(r)}) \tag{4.12}$$

$E_{i,j}^{(r)}$ is the logit output of the self attention mechanism for the relation r. This is further normalized using a softmax over the neighbourhood node features to obtain the attention coefficient α independently for each of the relations r .

$$\alpha_{i,j}^{(r)} = softmax_{j,r}(E_{i,j}^{(r)}) = \frac{exp(E_{i,j}^{(r)})}{\sum_{k \epsilon N_i} exp(E_{i,k}^{(r)})} \tag{4.13}$$

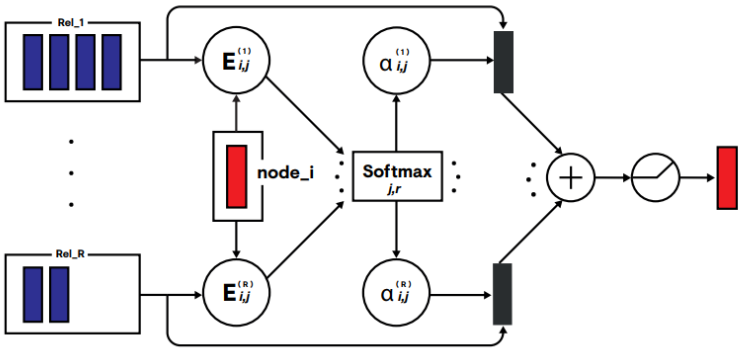


Figure 4.10: RGAT model BUSBRIDGE et al. (2019)

where N_i is the neighbourhood of node i, $\alpha_{(i,j)}^{(r)}$ is the attention coefficient for node i and j with respect to relation r. Combining the attention mechanism for each relation r and multi head attention mechanism explored in Equation 4.10, the propagation rule can be modified as follows:

$$\vec{h'}_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{r \epsilon R} \sum_{j \epsilon N_i} \alpha_{(i,j)}^{(r,k)} \vec{g}_j^{(r,k)}\right) \tag{4.14}$$

where K is the number of attention heads. The propagation rule aggregates all the relation specific features in the neighbourhood. This model is called Relational Graph attention network(RGAT) (BUSBRIDGE et al., 2019).

RGAT is another example of a heterogeneous network other than the RGCN model. In our thesis, we will use RGAT as a comparison against

the RGCN model to understand if the relative importance of the relation types has any impact on the performances of the model. Similar to GCN and RGCN, we would use multiple variants with differing layers to understand if the model performance increases with propagation depth.

4.5 Encoder-Decoder Model for Neighbourhood Reconstruction

The goal of learning node embeddings is to encode the node information as a dense vector which would represent the node's position in the knowledge graph and the other nodes connected to it in the neighbourhood. The task is to translate the relative positions of the graph nodes in the knowledge graph to the distance between them in the embedding space.

For the purposes of training the node, we employ an Encoder-Decoder model where an encoder maps the node information into a low-dimensional vector and a decoder, depending on the task, can be used to decode a neighbourhood or a node label. The decoding neighbourhood task is the link prediction task in which the similarity of a pair of nodes in the embedding space is used to decode if an edge exists between the pair in the original graph(HAMILTON, 2020).

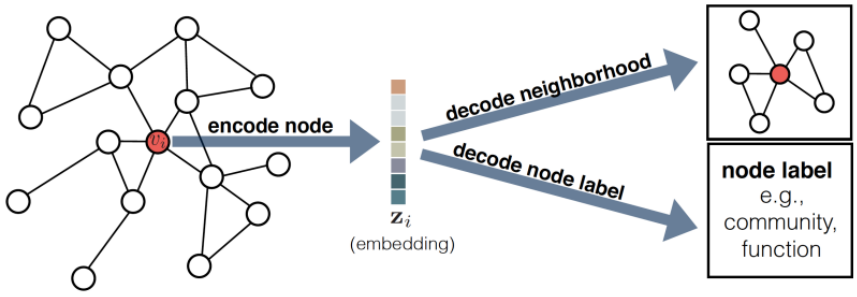


Figure 4.11: Encoder encodes the node v_i to obtain an embedding z_i which can be used by the decoder to either decode neighbourhood or node label

Now let us formalize the encoder function. An encoder maps the node $v \epsilon V$ to embeddings $z_v \epsilon \mathbb{R}^d$. In the simplest of terms the encoder can be defined as follows:

$$ENC(v) = Z[v] \tag{4.15}$$

where $Z \epsilon \mathbb{R}^{|V| \times d}$ is an embedding matrix and $Z[v]$ denotes the embedding vector for node v.

In the case of the decoder, we will only consider the neighbourhood reconstruction task for our thesis. A decoder receives the embeddings as input and tries to predict the neighbouring nodes. A typical decoder model is pairwise, i.e. it takes in a pair of node embeddings and tries to predict the relationship or similarity. Therefore a pairwise decoder would be applied for a pair of embeddings (z_u, z_v) to predict the existence of an edge between the nodes u and v. This can be formally defined as follows:

$$DEC(ENC(u), ENC(v)) = DEC(z_u, z_v) \tag{4.16}$$

The goal is to minimize the reconstruction loss in this encoder -decoder model so that

$$DEC(z_u, z_v) \approx S[u, v] \tag{4.17}$$

where $S[u, v]$ is a graph based similarity measure. The reconstruction loss L over the set of training node pairs (u, v) can be defined as :

$$L = \sum_{(u,v) \epsilon D} l(DEC(z_u, z_v), S[u, v]) \tag{4.18}$$

where is the loss function capturing the difference between the embedding based similarity $DEC(z_u, z_v)$ and the graph based true values $S[u, v]$.

Inner Product Decoder

KIPF and WELLING (2016b) defines Inner Product Decoder for unsupervised learning model -Variational Graph autoencoder where the dot product between a pair of embeddings is defined as a measure of similarity between the nodes.

$$DEC(z_u, z_v) = z_u^T z_v \tag{4.19}$$

Decoder for Multi Relational Data

The Inner product Decoder defined earlier does not take into account the multi-relational nature of the input knowledge graph. The earliest

method which realises learning multi relational embeddings is called RESCAL (NICKEL et al., 2011). The decoder is defined formally as :

$$DEC(u, r, v) = z_u^T \mathbf{R}_r z_v \tag{4.20}$$

where $\mathbf{R}_r \epsilon \mathbb{R}^{d \times d}$ is a learnable matrix specific to a relation $r \epsilon R$. This decoder is not generally used because it is computationally expensive and this can make it undesirable in the case of large graphs.

Popular Translation Embedding models like TransE, RotatE can themselves serve as decoder for a Graph neural network based encoders. The scoring function of these translational models are used as decoder function. For example, a TransE decoder as :

$$DEC(u, r, v) = -||z_u + r_r z_v|| \tag{4.21}$$

where r_r is the relational vector.

Similarly other semantic models like DistMult can also be used as decoder. DistMult is an updated version of RESCAL where we learn relation vectors rather than matrices which makes it much more efficient that RESCAL. The decoder function of DistMult can be defined as :

$$DEC(z_u, r, z_v) = \sum_{i=1}^{d} z_u[i] \times r_r[i] \times z_v[i] \tag{4.22}$$

where d is the embedding dimension.

This thesis uses the Inner product and DistMult decoders for our Graph neural network-based models. Since the GCN model is not a relational model, the inner product decoder is used, while the DistMult model will be used with the RGCN and RGAT model during node embeddings training.

We will also use a negative sampling technique to facilitate efficient training and generalizability. The most common approach for negative sampling is to use a uniform distribution over all nodes in the graph to generate 'negative' samples, i.e., triples which do not exist in the graph. The loss function defined for this case would be a binary cross-entropy loss as:

$$L = \sum_{(u,r,v)\epsilon E} \left(-log(\sigma(DEC(z_u, r, z_v))) - \sum_{(v_n)\epsilon P_(n,u)} [log(\sigma(-DEC(z_u, r, z_v)))] \right) \tag{4.23}$$

where $P_{n,u}$ is a small set of nodes obtained from the negative sampling distribution.

5
Experiments and Evaluation

5.1 Experimental Setup

For the purposes of the training process of learning the node embeddings in the knowledge graph, we employ the Knowledge completion or link prediction task for this purpose. A typical Knowledge completion task involves training the embedding model on a set of nodes V and a set of edges E_{train}, which is a subset of all the edges E in the knowledge graph. This task aims to use the trained embeddings to predict the edges that were left out of the training process.

We compare our Graph Neural Network model with some of the state-of-the-art Translation, Semantic Matching, and Random walk-based models. The translational model used in the experiment are TransE (BORDES et al., 2013a), RotatE (SUN et al., 2019)and BoxE (ABBOUD et al., 2020). For the Semantic matching models, we have the DistMult(YANG et al., 2014), ComplEx (TROUILLON et al., 2016) models. KRISTIADI et al. (2019). In the case of Random walk-based model, we use the RDF2Vec model for learning our node embeddings.

For the simplicity and ease of comparison, we chose to train 46-dimensional embeddings(similar number to the number of process parameters). We use the pyKEEN library (ALI et al., 2021) for the implementing the Translation and semantic model for our experiment. The RDF2Vec implementation in the pyRDF2Vec (VANDEWIELE et al., 2022) was used with our other embedding models. We used a default configuration of Adam optimizer, learning rate of 4×10^{-4} for the link prediction task. The number of epochs—TransE: 400, ComplEx: 1000, RotatE: 700, DistMult: 200, BoxE:

1500 and RDF2Vec: 1000—was chosen individually for each embedding method by inspecting its convergence on the train set.

GNN experimental Setup

As discussed earlier in chapter 4, we evaluate three popular Graph neural network models-based embeddings in the context of procedural knowledge graph in the manufacturing scenario. For the experiments, Graph neural networks are set up in the following manner:

- **Graph Convolution Networks(GCN)** (KIPF and WELLING, 2016a): Graph neural networks variant that does not differentiate between edge types. As explained earlier, the Decoder used for the GCN training is an Inner Product Decoder.

- **Relational Graph Convolution Networks(RGCN)** (SCHLICHTKRULL et al., 2018): Multi-Relational Graph neural networks variant which differentiates between different edge types. We use the DistMult-based decoder for RGCN because it can handle the multi-relational nature of our input embeddings

- **Relational Graph Attention Networks(RGAT)** (BUSBRIDGE et al., 2019): Multi-Relational Graph neural networks variant which has an attention mechanism for each relation. Similar to RGCN, we will use a DistMult decoder.

Please note that we use three different variations of these networks with different layers-1,2 and 3. As explained in the conceptual review of GNN and GCN, the layers in a GNN model signify the number of hops taken from the source node in the neighbourhood for the aggregation and updation of the GNN message passing process (see Figure 2.9). Each variation will be sub-scripted with the layer number to establish the number of layers used in these versions like GCN_1 or $RGCN_2$ etc. Pytorch Geometric library was used to implement these GNN models (FEY and LENSSEN, 2019).We explain the purpose of this variant when we discuss the evaluation results in the context of the research questions.

5.2 Evaluation

5.2.1 Subgraph Embeddings and Matches@K

Metrics which are commonly used to evaluate the embeddings in a link prediction setting are AMRI and hits@K. However, these are not suited for predicting parameters for the next iteration in procedural knowledge graphs. NORDSIECK et al. (2022b) proposes that for a downstream task such as choosing the fitting parameters for the process, the parameters adjusted for a quality characteristic should be similar in the embedding space and the knowledge graph. The Metric proposed is analogous to hist@k and is based on the overlap between the k closest quality characteristics in the graph and embedding space.

A sum-based aggregation of a subgraph of the Knowledge representation considering different quality characteristics as the focal node is calculated. Let us consider individual node embedding vectors as z for node v, with $v \epsilon S_n$, where S_n is the subgraph corresponding to a quality characteristic q. The Subgraph S_n is determined for each quality characteristic depending on the knowledge graph representation, i.e., the number of hops required to cover process parameters connected to a quality characteristic in each representation. Therefore for an unquantified rules representation as seen in Figure 4.1, only a single hop would fetch all process parameters associated with a quality characteristic, but the propagation depth would be 2 in the case of the representation in Figure 4.4.

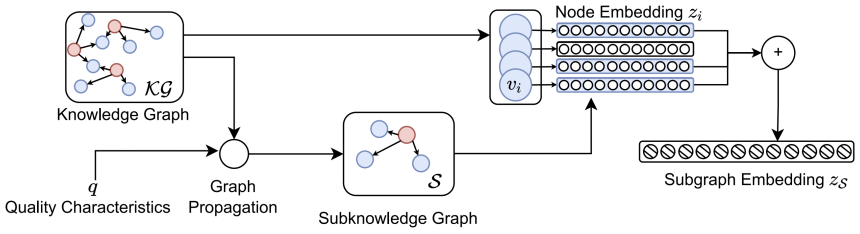


Figure 5.1: Subgraph is determined by the quality characteristic q as the starting node and node embedding in the subgraph are aggregated to obtain a subgraph embedding from NORDSIECK et al. (2021)

The aggregation on the Subgraph can be done as two alternatives- one with the head node(quality characteristics q) included or not included. If

the head node is not included, the Subgraph aggregation can be calculated as follows:

$$z_S^{\bar{h}} = \sum_{v_i, v_j \epsilon S} z_j \otimes D(z_i, z_j) \tag{5.1}$$

where v_i, v_j are the pairs of head and tail nodes resulting from graph propagation and $D(z_i, z_j)$ is the distance between node embeddings of v_i and v_j. However the head node can also be included and therefore the embedding of head node z_0 is added to obtain the following equation:

$$z_S^{h} = z_0 + \sum_{v_i, v_j \epsilon S} z_j \otimes D(z_i, z_j) \tag{5.2}$$

Subgraph embeddings are used to identify the closest quality characteristics in the embedding space for a given quality characteristic, and a set of quality characteristics ranked by similarity is obtained. Similarly, the number of parameters shared between two quality characteristics in the Knowledge graph determines their closeness in the Knowledge graph. These two ordered sets of quality characteristics nodes are matched for the top k values, and the values obtained are divided by the k value to obtain the metric in the range of [0,1]. The choice of k is essential and highly domain and dataset dependent. In our case, k=3 is an ideal case to measure the performance of each of these models. With higher k values, the similarity measure plateaus to a constant value for all embedding types, which is not ideal.

Calvo for Comparison

In order to compare the Matches@k results from various embedding models, we need to perform some significance tests to ascertain if the models' performance can be differentiated from the values obtained. Null hypothesis significance test (NHST) has been usually used as a method for model comparison but has been determined to be flawed in its interpretation (BENAVOLI et al., 2017). One of the assumptions that two models are necessarily equivalent if they are not different statistically is patently false. Such assumptions based on other parameters, such as p-value and confidence intervals whose values are arbitrarily chosen in the literature, make

it unreliable. CALVO et al. (2019) proposes the use of a Bayesian Plackett-Luce model, which is a probabilistic model to rank algorithms or, in our case, the embedding methods, i.e, the probability of each algorithm being the best when compared against others over a significant number of sample values(metric values from different iterations). For the purposes of this experiment, we use the Calvo models implemented in the cmpbayes library[1].

5.3 Evaluation Results

5.3.1 RQ1 : How do Graph Neural Network perform over a procedural knowledge Graph when compared against other state of the art baseline models?

For evaluating the Graph neural network models against the different translational, semantic matching and random walk models, we conducted the 30 iterations of training over different knowledge graph representations of the procedural data as explained in the Dataset section of Chapter 4. As discussed earlier, we use the Matches@k metric as the surrogate for our downstream process built over these embeddings. This metric has configurable options such as the inclusion of head, distance measure used in the embedding space and k values which we have defaulted here to 3.

Table 5.1 shows the Matches@k results for different embedding models and configurations for aggregation. For the simple notion of comparing each type of GNN, such as GCN, RGCN, and RGAT, we use the values of their best models instead of individual variants with different layers.

In the case of r_η, which is the representation with the highest level abstraction, RDF2Vec($\bar{h}, euc$) outperforms all the other models. It is important to note that this representation is homogeneous because it has only one type of edge between the nodes, and it is evident that Multi-relational models do not perform well enough. Although the Calvo plot in Figure 5.2, which provides a comparison for all the models, shows that RGCN has a higher chance of performing the best, the margin looks pretty slim considering the probability being so dispersed.

[1] `https://github.com/dpaetzel/cmpbayes`

Table 5.1: Subgraph embedding aggregation(Matches@k) results for k=3. Please note that the best method results are highlighted in bold

R.	H.	Dist.	TransE	ComplEx	RotatE	DistMult	BoxE	rdf2vec	GCN	RGCN	RGAT
r_{η}	h	euc	0.48	0.61	0.78	0.79	**0.81**	0.38	0.69	0.80	0.77
		jac	0.55	0.53	0.75	0.78	0.7	0.42	0.65	0.76	**0.78**
	$\bar{h}$	euc	0.52	0.5	0.61	0.73	0.72	**0.82**	0.73	0.69	0.70
		jac	0.53	0.51	0.63	0.69	0.73	0.51	0.69	0.70	**0.74**
$r_{\hat{\rho},rel}$	h	euc	0.48	0.62	0.56	0.53	0.69	0.52	0.70	**0.83**	0.78
		jac	0.55	0.51	0.56	0.56	0.5	0.51	0.68	0.78	**0.79**
	$\bar{h}$	euc	0.51	0.5	0.52	0.54	0.59	0.7	**0.73**	0.70	0.69
		jac	0.52	0.51	0.53	0.53	0.63	0.5	0.68	0.72	**0.74**
$r_{\hat{\rho},ch,e}$	h	euc	0.36	0.38	0.41	0.47	0.51	**0.67**	0.46	0.48	0.55
		jac	0.38	0.36	0.39	0.49	0.45	0.45	0.47	0.46	**0.57**
	$\bar{h}$	euc	0.38	0.34	0.4	0.47	0.49	**0.71**	0.50	0.46	0.53
		jac	0.4	0.34	0.38	0.5	0.49	0.52	0.48	0.45	**0.55**
$r_{\hat{\rho},ch,e,\eta}$	h	euc	0.44	0.47	0.55	0.6	**0.67**	0.6	0.53	0.62	0.64
		jac	0.49	0.43	0.53	0.62	0.62	0.51	0.53	0.61	**0.63**
	$\bar{h}$	euc	0.45	0.43	0.52	0.61	**0.63**	0.59	0.54	0.58	0.62
		jac	0.47	0.42	0.53	0.61	0.61	0.57	0.51	0.60	**0.64**
$r_{\hat{\rho},rei,e,\eta}$	h	euc	0.3	0.28	0.27	0.37	0.43	**0.64**	0.49	0.49	0.54
		jac	0.32	0.29	0.3	0.4	0.41	0.38	0.52	0.47	**0.60**
	$\bar{h}$	euc	0.3	0.28	0.27	0.37	0.41	**0.7**	0.49	0.43	0.53
		jac	0.33	0.29	0.3	0.39	0.45	0.43	0.50	0.47	**0.56**

For the representation $r_{\hat{\rho},rel}$, our graph neural network models perform the best, especially RGCN, which performs much better than all other models. This representation includes quantification of the process parameter adjustment as an edge attribute over the edges. This would also mean that the weight-agnostic models, like the translation models, do not account for the quantification in their training process, which is reflected in the results. The Matches@k results for all the models can again be compared using the Calvo plot (refer Figure **??**), clearly showing an overwhelming win for Graph neural network models.

In the case of representation $r_{\hat{\rho},ch,e}$, RDF2Vec performs phenomenally well compared to other models, almost 14% better than other configurations. The only reason we could discern from this upset for GNNs is the lack of a direct edge between the quality characteristics and process parameters due to the introduction of a new additional quantification. A random walks-based algorithm overcomes this lack of connection. Again in the context of model comparison, RDF2Vec seems clearly superior in terms of its performance (refer Figure 5.4).

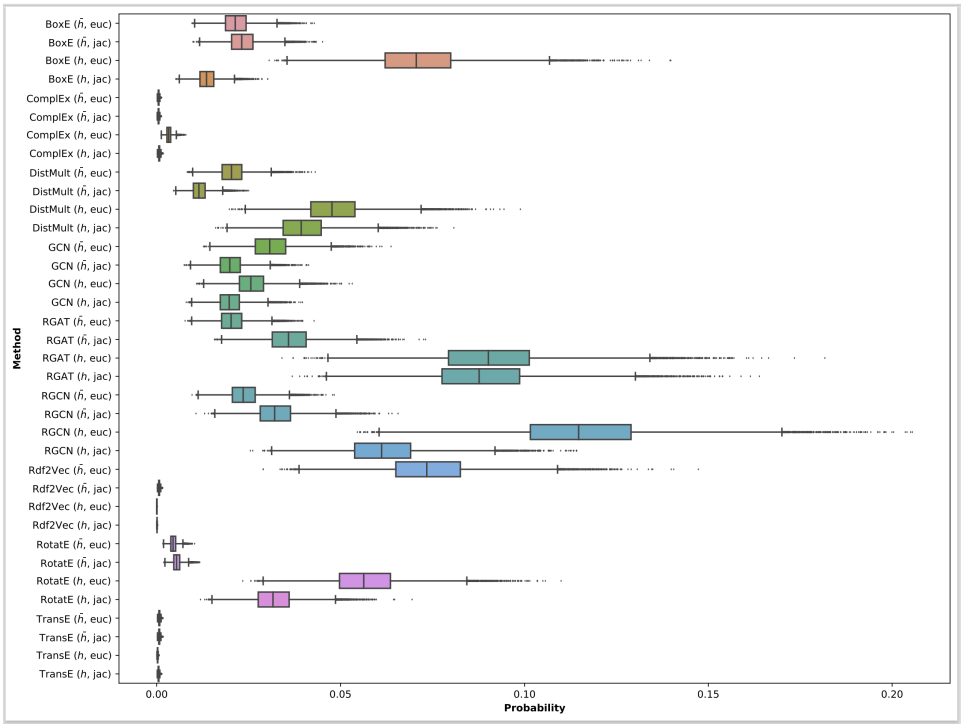


Figure 5.2: Calvo Plot for comparing all the embedding methods with their aggregation for representation r_{η}

For representation $r_{\hat{\rho},ch,e,\eta}$, BoxE performs slightly better than RGAT. Both these models can handle multi relational data well, BoxE specifically has known to have better performance with nodes with higher arity than just binary relations, which is part of this representation. BoxE implements n-ary relations as hyperplane and the GNNs seem to be lacking in that respect.

5.3.2 Indirections and Reifications

For the reification-based representation $r_{\hat{\rho},rei,e,\eta}$, the GNN suffers against RDF2Vec, which uses the random walk to overcome the indirections caused by reification. Reification upsets triples structure which forms the crux of most embedding models. In both these representations discussed above, triples no longer contain all the information essential to the quality characteristic and process parameters relationship. This indirection does not bode well for GNN models as well. The Calvo plots for representations

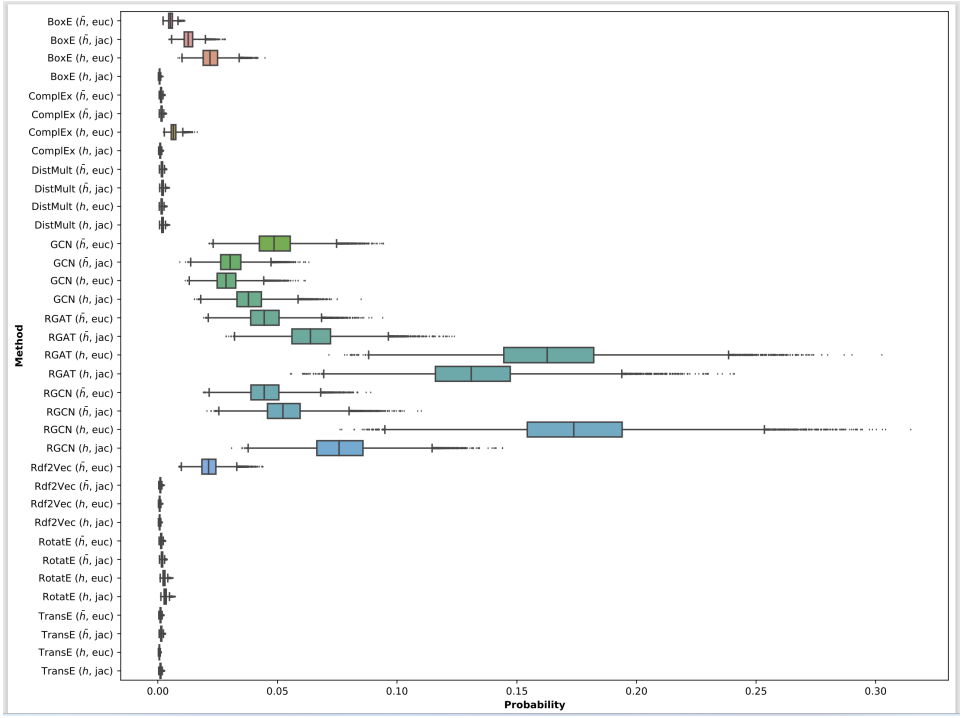


Figure 5.3: Calvo Plot for comparing all the embedding methods with their aggregation for representation $r_{\hat{\rho},rel}$

$r_{\hat{\rho},ch,e,\eta}$, $r_{\hat{\rho},rei,e,\eta}$ clearly shows this dynamics of the model as explained before as well. The differences between the RDF2Vec model performance and other models become more apparent in the Calvo plots.

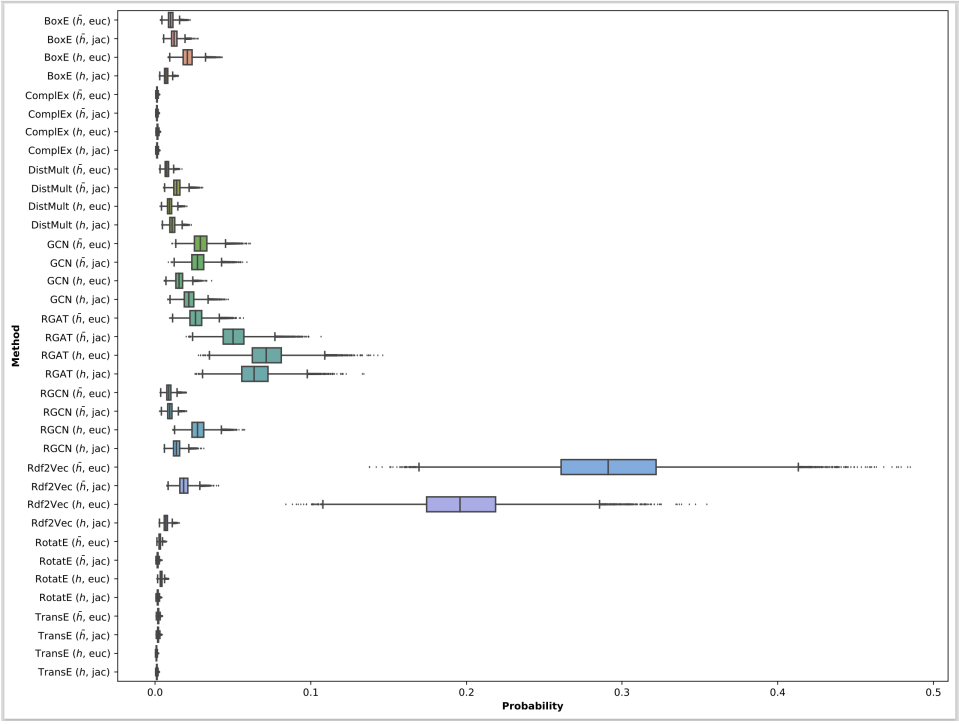


Figure 5.4: Calvo Plot for comparing all the embedding methods with their aggregation for representation $r_{\hat{\rho},ch,e}$

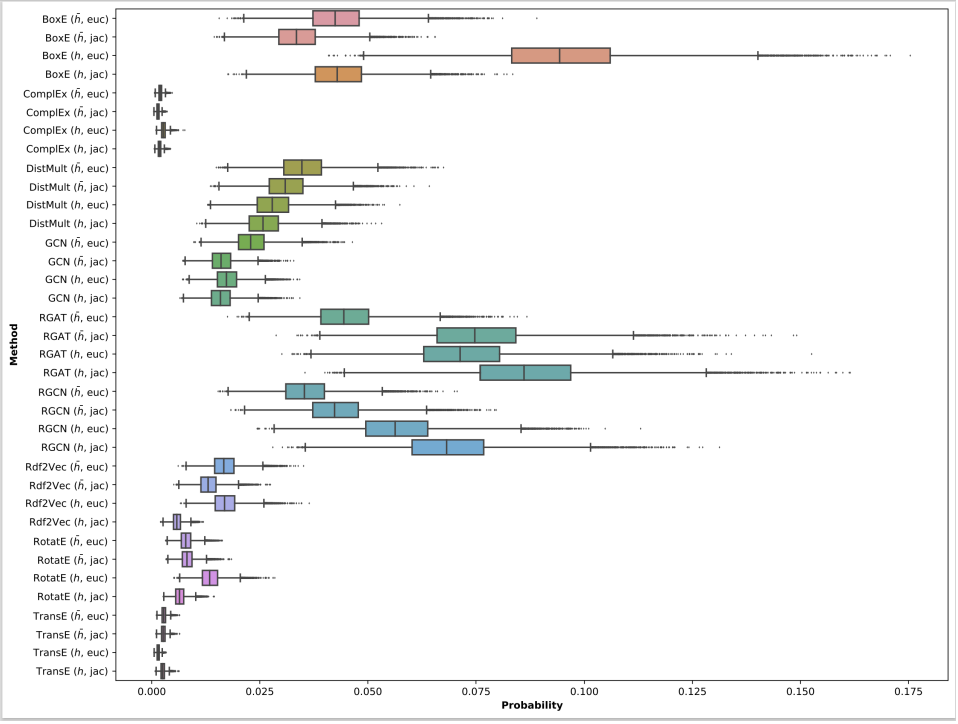


Figure 5.5: Calvo Plot for comparing all the embedding methods with their aggregation for representation $r_{\hat{\rho},ch,e,\eta}$

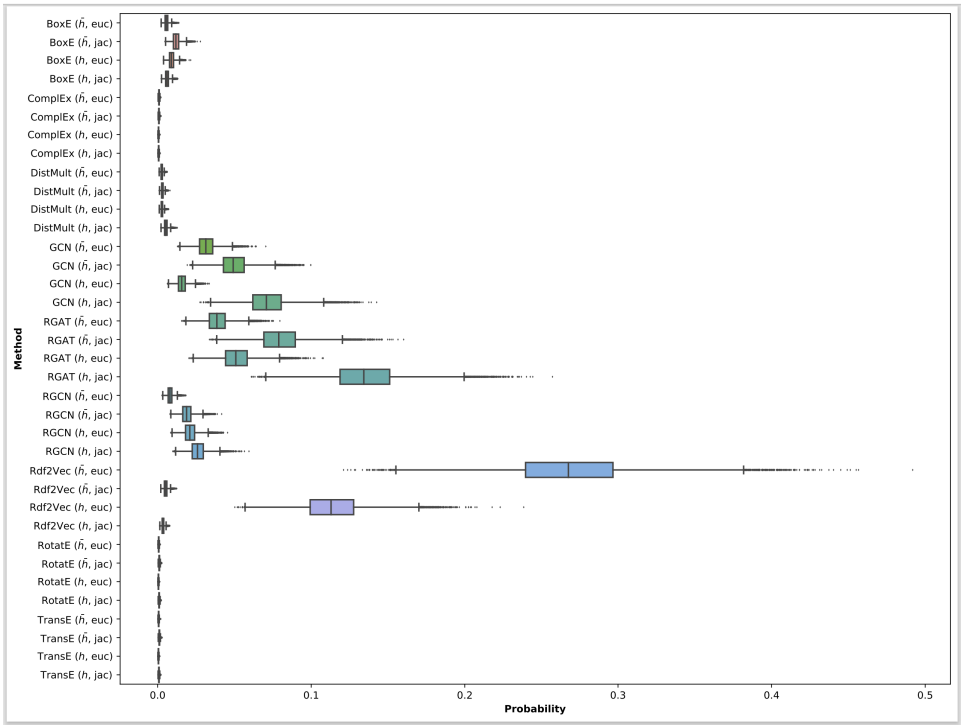


Figure 5.6: Calvo Plot for comparing all the embedding methods with their aggregation for representation $r_{\hat{\rho},rei,e,\eta}$

5.3.3 RQ2 : How does the GNN model configurations affect performance against different representations?

For this research question, we will analyze the results of GCN, RGCN, and RGAT variants against different representations and aggregation configurations. As explained in the experimental setup, we implemented three different versions-1 layer, 2 layer and 3-layer versions of each GNN model to compare their results in handling different models.

The primary motivation behind this task is to understand that the number of layers in a GNN measures the propagation depth from the head node. This is analogous to the receptive field concept of convolution concerning image processing. As the number of layers increases, the message-passing process aggregates more data farther from the head node with each layer.

Table 5.2: Subgraph embedding aggregation(Matches@k) results for k=3. for GCN, RGCN, RGAT with its variants.Please note that the best method results are highlighted in bold

R.	H	Dist.	$RGCN_1$	$RGCN_2$	$RGCN_3$	$RGAT_1$	$RGAT_2$	$RGAT_3$	GCN_1	GCN_2	GCN_3
r_η	h	euc	**0.8**	0.78	0.79	0.77	0.72	0.76	0.64	0.68	0.69
		jac	0.74	0.76	0.6	0.73	0.74	**0.78**	0.61	0.63	0.66
	$\bar{h}$	euc	0.63	0.66	0.69	0.66	0.65	0.7	0.57	0.66	**0.73**
		jac	0.65	0.69	0.7	0.68	0.68	**0.74**	0.59	0.64	0.69
$r_{\hat{\rho},rel}$	h	euc	0.8	**0.84**	0.77	0.78	0.78	0.74	0.63	0.66	0.7
		jac	0.72	0.78	0.57	0.73	0.77	**0.79**	0.58	0.63	0.68
	$\bar{h}$	euc	0.62	0.69	0.69	0.66	0.69	0.67	0.57	0.65	**0.73**
		jac	0.63	0.72	0.72	0.68	0.7	**0.74**	0.57	0.64	0.68
$r_{\hat{\rho},ch,e}$	h	euc	0.48	0.47	0.48	0.49	0.55	**0.54**	0.46	0.44	0.46
		jac	0.45	0.46	0.44	0.49	0.52	**0.57**	0.46	0.47	0.47
	$\bar{h}$	euc	0.41	0.41	0.46	0.41	0.48	**0.53**	0.5	0.47	0.45
		jac	0.42	0.42	0.45	0.46	0.5	**0.55**	0.47	0.48	0.47
$r_{\hat{\rho},ch,e,\eta}$	h	euc	0.57	0.61	0.62	0.58	**0.64**	0.61	0.52	0.49	0.53
		jac	0.6	0.61	0.61	0.6	**0.63**	0.63	0.5	0.49	0.53
	$\bar{h}$	euc	0.56	0.58	0.58	0.56	**0.62**	0.61	0.54	0.53	0.54
		jac	0.57	0.6	0.6	0.57	0.63	**0.64**	0.5	0.5	0.51
$r_{\hat{\rho},rei,e,\eta}$	h	euc	0.39	0.39	0.49	0.46	0.48	**0.54**	0.39	0.41	0.49
		jac	0.42	0.43	0.47	0.49	**0.55**	0.6	0.47	0.49	0.52
	$\bar{h}$	euc	0.35	0.37	0.43	0.4	0.46	**0.53**	0.41	0.45	0.49
		jac	0.38	0.39	0.47	0.45	0.53	**0.56**	0.47	0.47	0.5

It is evident from the values in the table that the Performance of GNN models vary with the complexity of the model and representations. Simpler models like $RGCN_1$ and GCN do perform well on representation with higher abstraction. Out of all the GNN model variant, $RGAT_3$ performs the

best in most cases. As a complex model with higher number learnable parameters $RGAT_3$ is able to better capture the complexities of a procedural graph with higher arity and reification. The performance of $RGCN_3$ and $RGAT_3$ is better in comparison to simpler models with low abstraction representations like $r_{\hat{\rho},ch,e}$, $r_{\hat{\rho},ch,e,\eta}$, $r_{\hat{\rho},rei,e,\eta}$.

Also with the chained binary representation in the $r_{\hat{\rho},ch,e}$, smaller models do not capture relationship between quality characteristics and process parameters where there is no directed edge between them.

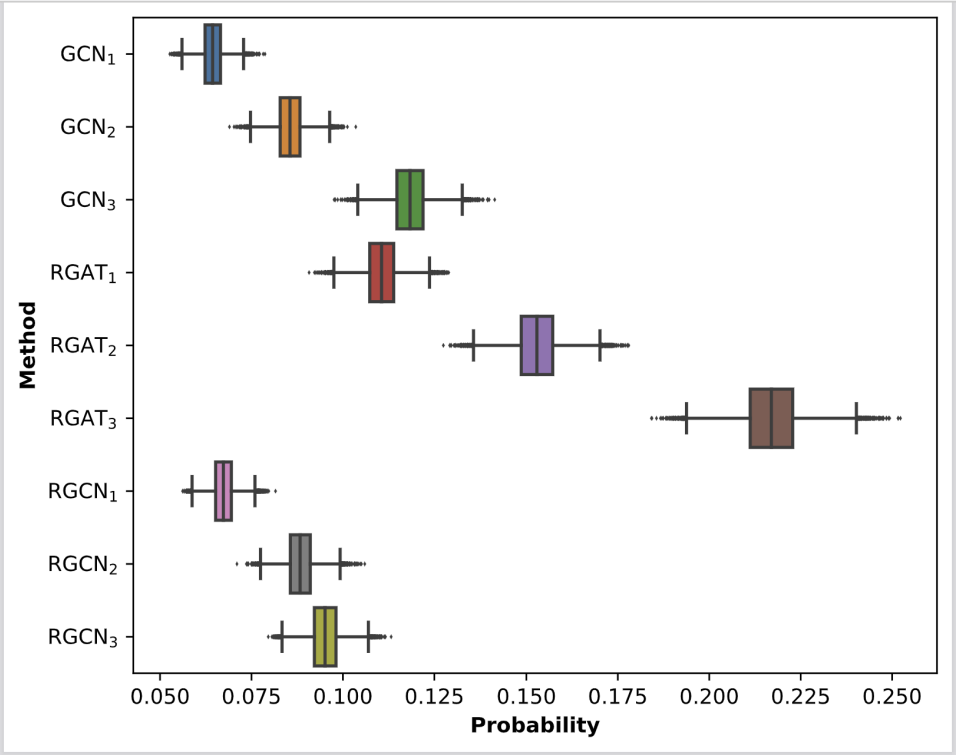


Figure 5.7: Calvo Plot for comparing all the GCN variants over all representations. Subscript represents the number of layers of convolution in these models

5.3.4 RQ3 : Among the GNN-based models, how does Multi-relational models perform against homogeneous models?

This question can be answered by looking at the Matches@k results from Table 5.2. It illustrates that the GCN model works well with high abstraction representation though the multi-relational models perform well too. The additional relational parameters learned by the RGCN though not directly useful in the case of r_{η}, still facilitate better performance than the GCN.

With the increasing complexity of the models, RGCN and RGAT perform much better than the GCN model, but the 3 -layer variant of GCN still gives comparable performance. This observation is much more visible in the case of the Calvo plot in Figure 5.7. In this plot, the probability of RGAT is the highest overall representation GCN takes the second spot over consistent performance.

We hypothesize that the higher metric values which occur RGCN and RGAT over GCN might be attributed to additional parameters rather than the multi-relational nature of the data itself. At this point, the lack of data, indirection and n-ary relations play a more significant role in the overall performance than multi-relationality. This conclusion can be drawn by looking at the values of RGCN_3 and GCN_3 models over the $r_{\hat{\rho},rei,e,\eta}$ representation. Both models perform almost the same, even though one is multi-relational while the other is relation-type agnostic.

6

Conclusions and Future Work

6.1 Conclusions

The primary objective of this thesis has been to evaluate a popular Graph neural network model over the different representations of a procedural knowledge graph obtained from a manufacturing process. As part of the project, we perform the experiments on a practical manufacturing scenario- the AIPE project, which is a dataset obtained from the Fused Diffusion Model 3D printing use-case. Different representations with differing levels of abstraction were used to determine how embedding model performance varied with the increasing complexity of the input data.

In the context of this thesis, we looked at the basic concepts behind the Graph neural network Architectures in Chapter 2. It touched upon the concepts of edge, node representation as matrices and adjacency matrix as graph properties which are input to GNNs. Then we explained the basic concepts of updation and aggregation, which are used in GNN message passing while learning the embeddings.

For comparison of our GNN models, we chose the state-of-the-art baseline Translation, semantic matching and Random walk-based models from the literature. We discussed these models in detail in Chapter 3. In the same chapter, we explored how knowledge graph research has taken over the manufacturing and production industry in the wake of Industry 4.0.

Chapter 4 offered a deeper look at the concepts behind Graph convolution, its origin and then a more profound conceptual background on the Graph Convolution Networks and its variant, which we have used in our thesis.

This chapter also explained the Encoder-Decoder Model architecture used in training GNN models to learn node embeddings.

Finally, we evaluated our research questions with various experiments in Chapter 5. We defined an alternate metric- Matches@k for our current scenario using the Subgraph embedding aggregation concept proposed by (NORDSIECK et al., 2019). Different translation models such as TransE, RotatE and BoxE and Semantic Models like DistMult, ComplexE, and a random walk-based model RDF2Vec were compared with the GNN models -GCN, RGCN, and RGAT. We also explored the idea of propagation depth being a factor in the case of model performance when different representations with differing complexities are compared.

Here we draw some important conclusions from our experiments. First, GCN and its variants perform well with the representation with edge weight against edge agnostic methods.

Second, GCN and its variants do not perform better with knowledge graphs containing higher arity or reified relationships, which causes indirections in the knowledge graph. Overall the GNN models performed consistently over all the representations despite the upsets at individual configurations, while other model performances varied erratically with the configurations and representations.

Third, the propagation depth of GNN models based on the number of layers affects the performance concerning more complex representations. The simpler GCN model and $RGCN_1$ variant performed well with representation with the highest abstraction. With the lower abstraction, $RGAT_3$ with 3 layers could better capture the complexities of chained binary relations and indirections than simpler GCN models.

Fourth, homogeneous models perform poorly than multi-relational models in most cases in procedural graphs owing to their complex nature. A method like RDF2Vec, which performs random walks over the neighborhood, works well in this kind of knowledge graph structure. This is primarily because of scarce edge attributes or node attributes and a need for more data for a deep learning model to capture the essence of the graph neighborhood accurately.

6.2 Limitations and Future Work

6.2.1 Dataset

One of the primary limitations of our work is the lack of procedural data. The acquisition of data in the context of modelling and manufacturing is very expensive. During our experiments, we augmented the data using the negative sampling technique where we trained the model on edges that do not exist in the original graph with edge probability between the edges as zero.

Apart from our Local closed world Assumption negative sampling method, a future work could look to impute possible process parameter values and re-parameterization values by simulating the manufacturing procedure or defining a probability distribution for each of the input variables and augmenting values by sampling from this distribution. In essence, create methods to synthetically generate more data using the domain knowledge of the manufacturing process.

6.2.2 n-ary relations

Another major limitation of our work was learning the graph structure from the n-ary relations. In the relationship with lower abstractions and higher complexity, our major problem was the inability of our current GNN models used here in our experiments to train over and learn representations from relations which were not binary. Ternary relations in the representations led to a disconnect between the process parameters and quality characteristics when edges were defined only as binary. In a complex setting where more than two relations define a single fact, GNN fails to recognize them as efficiently as other embedding models like BoxE, which uses the hyperplanes to learn relation embedding and can handle n-ary relations.

An approach to implement n-ary relations in a graph neural network context would be to create flexible architectures to take a differing number of source nodes when considering a relation or a fact. Few examples in the literature, such as the Graph LSTM architecture used in the case of (WANG et al., 2018) and Neuinfer (GUAN et al., 2020) architecture with additional

convolution layer for other entities of relations apart from a primary binary relation. Other architectures would include using hypergraph graph neural networks, each hyperplane used to represent n-ary relations as binary relations in each hyperplane later concatenated across to obtain a single n-ray embedding.

6.2.3 Indirections

Complex processes, such as manufacturing processes, have dependencies between various nodes, which are essential to be viewed as a whole. With iterative relations, which have dependencies between each iteration, the representation of this structure exponentially becomes complicated and cannot be seen as binary relations or triples. This case might be similar to n-ary relations, but now the focus of the problem is neighbourhood structures. In the case of low abstraction representation, the graph neighbourhood reconstruction can have reified relations, n-ary relations, graph loops and isolated neighbourhood structures.

Graph neural network works by learning and updating their weights and embedding for node and edges through aggregating learnt weight from propagating in the neighbourhood. In indirections caused by the abovementioned complex representation, GNN message passing architecture, as established in the case of GNN models like GCN, RGCN, and RGAT, may not capture the neighbourhood by using the existing approach's convolution-based neighbourhood aggregation mechanism. A more complex message-passing technique to better capture underlying indirections can be designed.

The simplest method which worked in our representations was using a Random walk-based algorithm like RDF2Vec; these representations capture the neighbourhood structure more quickly than a more sophisticated method like GNN.

A

List of Figures

B

List of Tables

C

Bibliography

[ABBOUD et al. 2020] R. Abboud, I. Ceylan, T. Lukasiewicz and T. Salvatori. **Boxe: A box embedding model for knowledge base completion**. Advances in Neural Information Processing Systems, Vol. 33:9649–9661, 2020.

[AGGOUR et al. 2019] K. S. Aggour, V. S. Kumar, P. Cuddihy, J. W. Williams, V. Gupta, L. Dial, T. Hanlon, J. Gambone and J. Vinciquerra. **Federated Multimodal Big Data Storage Analytics Platform for Additive Manufacturing**. In: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 1729–1738.

[ALI et al. 2021] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp and J. Lehmann. **PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings**. J. Mach. Learn. Res., Vol. 22(82):1–6, 2021.

[BENAVOLI et al. 2017] A. Benavoli, G. Corani, J. Demšar and M. Zaffalon. **Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis**. Journal of Machine Learning Research, Vol. 18(77):1–36, 2017.

[BENGIO et al. 2000] Y. Bengio, R. Ducharme and P. Vincent. **A neural probabilistic language model**. Advances in neural information processing systems, Vol. 13, 2000.

[BISWAS 2023] R. Biswas. **Embedding Based Link Prediction for Knowledge Graph Completion**. Ph.D. thesis, Karlsruher Institut für Technologie (KIT), 2023.

[BIZER et al. 2009] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak and S. Hellmann. **Dbpedia-a crystallization point for the web of data**. Journal of web semantics, Vol. 7(3):154–165, 2009.

[BOLLACKER et al. 2008] K. Bollacker, C. Evans, P. Paritosh, T. Sturge and J. Taylor. **Freebase: a collaboratively created graph database for structuring human knowledge**. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 1247–1250.

[BORDES et al. 2013] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston and O. Yakhnenko. **Translating embeddings for modeling multi-relational data**. Advances in Neural Information Processing Systems, Vol. 26, 2013a.

[BORDES et al. 2013b] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston and O. Yakhnenko. **Translating Embeddings for Modeling Multi-relational Data**. In: C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Weinberger, Eds., Advances in Neural Information Processing Systems, Vol. 26. 2013, Curran Associates, Inc.

[BUCHGEHER et al. 2021] G. Buchgeher, D. Gabauer, J. Martinez-Gil and L. Ehrlinger. **Knowledge graphs in manufacturing and production: A systematic literature review**. IEEE Access, Vol. 9:55537–55554, 2021.

[BUSBRIDGE et al.] **Relational Graph Attention Networks**.

[CALVO et al. 2019] B. Calvo, O. M. Shir, J. Ceberio, C. Doerr, H. Wang, T. Bäck and J. A. Lozano. **Bayesian performance analysis for black-box optimization benchmarking**. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019, pp. 1789–1797.

[DETTMERS et al. 2018] T. Dettmers, P. Minervini, P. Stenetorp and S. Riedel. **Convolutional 2d knowledge graph embeddings**. In: Proceedings of the AAAI conference on artificial intelligence, 2018, Vol. 32.

[DIMITROV et al. 2020] D. Dimitrov, E. Baran, P. Fafalios, R. Yu, X. Zhu, M. Zloch and S. Dietze. **TweetsCOV19 - A Knowledge Base of Semantically Annotated Tweets about the COVID-19 Pandemic**. In: Proceedings of the 29th ACM International Conference on Information Knowledge Management, CIKM '20, p. 2991–2998. 2020, Association for Computing Machinery, New York, NY, USA.

[DUAN and CHIANG 2016] W. Duan and Y.-Y. Chiang. **Building Knowledge Graph from Public Data for Predictive Analysis: A Case Study on Predicting Technology Future in Space and Time**. In: Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, BigSpatial '16, p. 7–13. 2016, Association for Computing Machinery, New York, NY, USA.

[EHRLINGER and WÖSS 2016] L. Ehrlinger and W. Wöß. **Towards a definition of knowledge graphs.** SEMANTiCS (Posters, Demos, SuCCESS), Vol. 48(1-4):2, 2016.

[FEY and LENSSEN 2019] M. Fey and J. E. Lenssen. **Fast Graph Representation Learning with PyTorch Geometric**. 2019, In: ICLR Workshop on Representation Learning on Graphs and Manifolds.

[GILMER et al. 2017] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals and G. E. Dahl. **Neural message passing for quantum chemistry**. In: International conference on machine learning, pp. 1263–1272. 2017, PMLR.

[GLOROT et al.] **A Semantic Matching Energy Function for Learning with Multi-relational Data**.

[GROVER and LESKOVEC 2016] A. Grover and J. Leskovec. **node2vec: Scalable feature learning for networks**. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[GRUBER 1989] T. R. Gruber. **Automated Knowledge Acquisition for Strategic Knowledge**. Machine Learning, Vol. 4(3-4):293–336, 1989.

[GUAN et al. 2020] S. Guan, X. Jin, J. Guo, Y. Wang and X. Cheng. **Neuinfer: Knowledge inference on n-ary facts**. In: Proceedings of the 58th annual meeting of the association for computational linguistics, 2020, pp. 6141–6151.

[HAMILTON 2020] Hamilton. **Graph Representation Learning**. Synthesis Lectures on Artifical Intelligence and Machine Learning, Vol. 14.3:1–159, 2020.

[HAMILTON et al. 2017] W. L. Hamilton, R. Ying and J. Leskovec. **Representation learning on graphs: Methods and applications**. Bulletin of

the IEEE Computer Society Technical Committee on Data Engineering, Vol. 40(3):52–74, 2017.

[HAN et al.] **Vision GNN: An Image is Worth Graph of Nodes**.

[HE and JIANG 2019] L. He and P. Jiang. **Manufacturing knowledge graph: a connectivism to answer production problems query with knowledge reuse**. IEEE Access, Vol. 7:101231–101244, 2019.

[HOFFART et al. 2013] J. Hoffart, F. M. Suchanek, K. Berberich and G. Weikum. **YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia**. Artificial intelligence, Vol. 194:28–61, 2013.

[HOGAN et al. 2021] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. Ngonga Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab and A. Zimmermann. **Knowledge graphs**. ACM Computing Surveys (CSUR), Vol. 54(4):1–37, 2021.

[JI et al. 2021] S. Ji, S. Pan, E. Cambria, P. Marttinen and S. Y. Philip. **A survey on knowledge graphs: Representation, acquisition, and applications**. IEEE Transactions on Neural Networks and Learning Systems, 2021.

[KATTEPUR and P 2019] A. Kattepur and B. P. **RoboPlanner: Autonomous Robotic Action Planning via Knowledge Graph Queries**. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, p. 953–956. 2019, Association for Computing Machinery, New York, NY, USA.

[KIPF and WELLING 2016] T. N. Kipf and M. Welling. **Semi-supervised classification with graph convolutional networks**. arXiv preprint arXiv:1609.02907, 2016a.

[KIPF and WELLING] **Variational Graph Auto-Encoders**.

[KRATHWOHL 2002] D. R. Krathwohl. **A revision of Bloom's taxonomy: An overview**. Theory into practice, Vol. 41(4):212–218, 2002.

[KRISHNAN] **Making search easier: How Amazon's Product Graph is helping customers find products more easily**. Amazon Blog.

[KRISTIADI et al. 2019] A. Kristiadi, M. A. Khan, D. Lukovnikov, J. Lehmann and A. Fischer. **Incorporating literals into knowledge graph embeddings**. In: International Semantic Web Conference, 2019, pp. 347–363.

[LI et al. 2021] Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou. **A survey of convolutional neural networks: analysis, applications, and prospects**. IEEE transactions on neural networks and learning systems, 2021.

[LIN et al. 2015] Y. Lin, Z. Liu, M. Sun, Y. Liu and X. Zhu. **Learning entity and relation embeddings for knowledge graph completion**. 2015, In: Twenty-ninth AAAI conference on artificial intelligence.

[MANNING and SCHUTZE 1999] C. Manning and H. Schutze. **Foundations of statistical natural language processing**. MIT press, 1999.

[MAYACHITA] **Understanding graph convolutional networks for node classification**.

[MIKOLOV et al. 2013] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean. **Distributed representations of words and phrases and their compositionality**. Advances in Neural Information Processing Systems, Vol. 26, 2013.

[NAVIGLI and PONZETTO 2012] R. Navigli and S. P. Ponzetto. **BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network**. Artificial Intelligence, Vol. 193:217–250, 2012.

[NAYAK et al. 2020] A. Nayak, V. Kesri and R. K. Dubey. **Knowledge Graph Based Automated Generation of Test Cases in Software Engineering**. In: Proceedings of the 7th ACM IKDD CoDS and 25th COMAD, CoDS COMAD 2020, p. 289–295. 2020, Association for Computing Machinery, New York, NY, USA.

[NICKEL et al. 2011] M. Nickel, V. Tresp and H.-P. Kriegel. **A three-way model for collective learning on multi-relational data**. 2011, In: Icml.

[NORDSIECK et al. 2019] R. Nordsieck, M. Heider, A. Angerer and J. Hähner. **Towards Automated Parameter Optimisation of Machinery by Persisting Expert Knowledge**. In: Oleg Gusikhin, Kurosh Madani and Janan Zaytoon, Eds., Proceedings of the 16th International Conference

on Informatics in Control, Automation and Robotics, ICINCO 2019 - Volume 1, Prague, Czech Republic, July 29-31, 2019, pp. 406–413. 2019, SCITEPRESS.

[NORDSIECK et al. 2022a] R. Nordsieck, M. Heider, A. Hoffmann and J. Hähner. **Reliability-Based Aggregation of Heterogeneous Knowledge to Assist Operators in Manufacturing**. In: 2022 IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 131–138.

[NORDSIECK et al. 2022b] R. Nordsieck, M. Heider, A. Hummel and J. Hähner. **A Closer Look at Sum-based Embeddings for Knowledge Graphs Containing Procedural Knowledge**. 2022, In: M. Alam, D. Buscaldi, M. Cochez, F. Osborne and Reforgiato Recupero Diego, Eds., Proceedings of the Workshop on Deep Learning for Knowledge Graphs (DL4KG 2022) co-located with the 21th International Semantic Web Conference (ISWC 2022).

[NORDSIECK et al. 2021] R. Nordsieck, M. Heider, A. Winschel and J. Hähner. **Knowledge extraction via decentralized knowledge graph aggregation**. In: 2021 IEEE 15th International Conference on Semantic Computing (ICSC), 2021, pp. 92–99.

[NOY et al. 2019] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson and J. Taylor. **Industry-scale knowledge graphs: Lessons and Challenges**. Communications of the ACM, Vol. 62(8):36–43, 2019.

[NOY et al. 2006] N. Noy, A. Rector, P. Hayes and C. Welty. **Defining n-ary relations on the semantic web**. W3C working group note, 2006.

[PALMONARI and MINERVINI 2020] M. Palmonari and P. Minervini. **Knowledge graph embeddings and explainable AI**. Knowledge Graphs for Explainable Artificial Intelligence: Foundations, Applications and Challenges, Vol. 47:49, 2020.

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

[PORTISCH et al. 2022] J. Portisch, N. Heist and H. Paulheim. **Knowledge graph embedding for data mining vs. knowledge graph embed-**

ding for link prediction–two sides of the same coin? Semantic Web, Vol. 13(3):399–422, 2022.

[RAIMOND et al. 2014] Y. Raimond, T. Ferne, M. Smethurst and G. Adams. **The BBC world service archive prototype**. Journal of web semantics, Vol. 27:2–9, 2014.

[RISTOSKI and PAULHEIM 2016] P. Ristoski and H. Paulheim. **Rdf2vec: Rdf graph embeddings for data mining**. In: International Semantic Web Conference, 2016, pp. 498–514.

[RUSSELL 2016] S. Russell. **Artificial Intelligence: A Modern Approach, eBook, Global Edition**. Pearson Education, Limited, 2016.

[SANCHEZ-LENGELING et al. 2021] B. Sanchez-Lengeling, E. Reif, A. Pearce and A. B. Wiltschko. **A Gentle Introduction to Graph Neural Networks**. Distill, 2021. Https://distill.pub/2021/gnn-intro.

[SCARSELLI et al. 2008] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini. **The graph neural network model**. IEEE transactions on neural networks, Vol. 20(1):61–80, 2008.

[SCHLICHTKRULL et al. 2018] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. den van Berg, I. Titov and M. Welling. **Modeling relational data with graph convolutional networks**. In: European semantic web conference, 2018, pp. 593–607.

[SHRIVASTAVA] **Bring rich knowledge of people, places, things and local businesses to your apps**. Bing Blogs.

[SUN et al. 2019] Z. Sun, Z.-H. Deng, J.-Y. Nie and J. Tang. **Rotate: Knowledge graph embedding by relational rotation in complex space**. arXiv preprint arXiv:1902.10197, 2019.

[THANAPALASINGAM et al. 2022] T. Thanapalasingam, L. van Berkel, P. Bloem and P. Groth. **Relational graph convolutional networks: a closer look**. PeerJ Computer Science, Vol. 8:e1073, 2022.

[TROUILLON et al. 2016] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier and G. Bouchard. **Complex embeddings for simple link prediction**. In: International conference on machine learning, 2016, pp. 2071–2080.

[VANDEWIELE et al. 2022] G. Vandewiele, B. Steenwinckel, T. Agozzino and F. Ongenae. **pyRDF2Vec: A Python Implementation and Extension of RDF2Vec**. arXiv preprint arXiv:2205.02283, 2022.

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

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

[VRANDEČIĆ 2012] D. Vrandečić. **Wikidata: A New Platform for Collaborative Data Collection**. In: Proceedings of the 21st International Conference on World Wide Web, WWW '12 Companion, p. 1063–1064. 2012, Association for Computing Machinery, New York, NY, USA.

[WANG et al.] **DOLORES: Deep Contextualized Knowledge Graph Embeddings**.

[WANG et al. 2017] Q. Wang, Z. Mao, B. Wang and L. Guo. **Knowledge Graph Embedding: A Survey of Approaches and Applications**. IEEE Transactions on Knowledge and Data Engineering, Vol. 29(12):2724–2743, 2017.

[WANG et al. 2014] Z. Wang, J. Zhang, J. Feng and Z. Chen. **Knowledge graph embedding by translating on hyperplanes**. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2014, Vol. 28.

[WESTON et al. 2013] J. Weston, A. Bordes, O. Yakhnenko and N. Usunier. **Connecting Language and Knowledge Bases with Embedding Models for Relation Extraction**. CoRR, Vol. abs/1307.7973, 2013.

[YANG et al. 2014] B. Yang, W.-t. Yih, X. He, J. Gao and L. Deng. **Embedding entities and relations for learning and inference in knowledge bases**. arXiv preprint arXiv:1412.6575, 2014.

[ZHANG et al. 2019] S. Zhang, H. Tong, J. Xu and R. Maciejewski. **Graph convolutional networks: a comprehensive review**. Computational Social Networks, Vol. 6(1):1–23, 2019.

[ZHANG et al. 2020] W. Zhang, H. Cao, F. Hao, L. Yang, M. Ahmad and Y. Li. **The chinese knowledge graph on domain-tourism**. In: Advanced Multimedia and Ubiquitous Engineering: MUE/FutureTech 2019 13, pp. 20–27. 2020, Springer.

Declaration of Academic Integrity

I hereby declare that I have written the present work myself and did not use any sources or tools other than the ones indicated.

Datum: .....

(Signature)