Creating differentiable graph metrics to improve link prediction

Simone Colombo
Department of Computer Science
Vrije Universiteit Amsterdam
De Boelelaan 1105, 1081 HV Amsterdam
s2.colombo@student.vu.nl

Abstract

Relational Graph Convolutional Networks (R-GCNs) have been shown to be effective in modeling Multi-relational graphs for different base completion tasks; node classification and link prediction. In addition, the R-GCN provides a way of learning graph node embedding by successfully exploiting the graph connectivity structure as result of the underlying message passing mechanism. We further focus on enhancing the R-GCN link prediction power by utilizing its message passing mechanism with multi-task learning. We modify the R-GCN factorization method for link prediction, DistMult, by including the Potential Energy as additional feature. To the best of our knowledge, we are the first to introduce a new version of the R-GCN by including a knowledge base-specific metric in its decoder. Furthermore, we demonstrate that the vanilla version of RGCN lacks the ability to generalize for the knowledge base-specific metric we include in the factorization method, namely the Potential Energy.

1 Introduction

Nowadays, knowledge graphs are widely used in a variety of different fields (Abu-Salih 2021), and they have become the backbone of different AI-driven applications which are employed in diverse domains (Gao et al. 2020; Nickel et al. 2016; Futia and Vetrò 2020; Ji et al. 2021). Moreover, since knowledge graphs were first officially announced by Google (Singhal 2012), and then other companies followed the trend (R.J. Pittman 2017; Krishnan 2018; Socher et al. 2013a; Noy et al. 2019; Devarajan 2017), they continued to evolve and they are now in their proliferation phase; where the pace of research and development is skyrocketing (KGI 2021; KGD 2020; KGC 2021).

The rapid increase of interest in knowledge graphs led to the emergence of a broader landscape of research horizons in different fields (Abu-Salih 2021), leveraging the application of knowledge graphs for new tasks; recommendation systems (Nayyeri et al. 2020; Yao et al. 2019; Ayala-Gómez et al. 2018), literature surveys generation (Oelen et al. 2020), question answering (Jaradeh, Stocker, and Auer 2020) and knowledge discovery (Vahdati et al. 2018). Moreover, this build-up period led to the development of new graph-tailored neural network models, namely graph neural network (GNN) (Wu et al. 2021). Following previous research on knowledge graphs, we assume that the

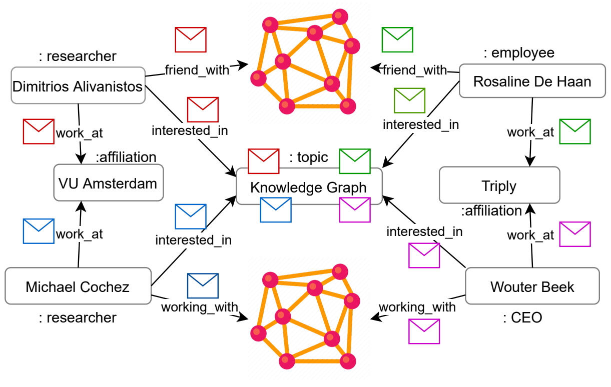


Figure 1: A 2-hop message passing mechanism: The nodes are entities, the edges are relations labeled with their types, the nodes are labeled with entity types (e.g., *:researcher*). The colored messages are the information gathered from the 2-hop neighbours to build the Knowledge Graph entity embedding. The topological structure of the entire graph is not learnt since the messages are from the 2-hop neighbours.

state-of-the-art GNN models, Graph Convolution Networks (GCNs) (Kipf and Welling 2017), use the node-related Message Passing mechanism (Gilmer et al. 2017) to create informative node embeddings. This approach considers individual knowledge bases as directed labeled multigraphs; where the entities are nodes and the triples are encoded with labeled edge between subject and object nodes. Furthermore, this approach ignores the overall graph topological structure (see Figure 1).

2 Problem Definition

We consider as our main statistical relational learning (SRL) task, inductive link prediction (recovering of missing triples). Thus, missing triples can be deemed to exists within the knowledge graph encoded through the neighborhood structure,i.e. knowing that *Michael Cochez* works with a person with whom *Wouter Beek* works too, implies that there is high probability that the triple (*Michael Cochez*, *friend_with*, *Wouter Beek*) must belong to the knowledge graph. Our link prediction model follows the work of Schlichtkrull et al. (2017) which utilizes the R-GNC architecture as main model. Furthermore, we enhance the R-GCN architecture with the Potential Energy (PE) (Kumar and Sharma 2020) of the knowledge bases to include the topological graph structure in our model. Our link prediction model can be regarded as an autoencoder made up of (1) encoder, the R-GCN, and an enhanced decoder version of (2) DistMult (Yang et al. 2015) by leveraging the Potential Energy. The model, including the R-GCN parameters, is learnt by using Multi-task learning (MTL). Hence, a multi-layer perceptron (MLP) is trained by optimizing L1 loss. Successively, the PE is fed as input to DistMult which maximises it. Finally, the eR-GCN (enhanced R-GCN) is optimized for cross-entropy loss.

2.1 Contributions

Our contributions are as follows.

- introducing a R-GCN architecture which considers both the neighborhood information and the overall topological structure of the knowledge graph to enhance the link prediction task,
- introducing the theoretical idea to enhance the message-passing neural networks (MPNNs) by leveraging the PE graph metric,
- showing that our enhanced R-GCN (eR-GCN) does not improve in performance due to the inability of the vanilla version of the RGCN to generalize knowledge for the Potential Energy.

3 Potential Energy Modeling

We introduce the 2 following notations: (1)let $\mathcal{L}$ be a bipartite graph $\mathcal{L} = (\mathcal{V}, \mathcal{U}, \mathcal{R})$, where $\mathcal{U}$ and $\mathcal{V}$ are two disjoint and independent sets of vertices and $\mathcal{R}$ is set of edges at timestamp T_1. The inductive link prediction aims to predict new link among nodes that can be observed at $T_2 \mid T_2 > T_1$. (2) Let $\mathcal{G}$ labeled multi-graph and $\mathcal{G} = (\mathcal{V}_1, \mathcal{E}, \mathcal{R}_1)$ with nodes (entities) $A_i \in \mathcal{V}_1$ and labeled edges (relations) $(A_i, r, A_j) \in \mathcal{E}$, where $r \in \mathcal{R}_1$ is a relation type.

3.1 Potential Energy for a Bipartite Graph

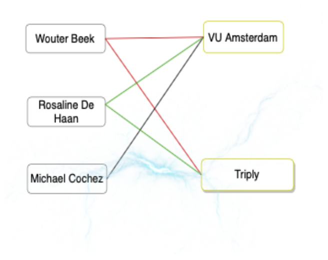
Defined in Kumar and Sharma (2020), a pair of nodes, $(A, B) \in \mathcal{V}$, is considered as an object and the product of degree of between the two nodes, $D_p^{(A,B)}$, as its mass. The force g between the nodes is represented by the sum of the clustering coefficient of their common neighbours, $S_{C_z}^{(A,B)}$, and the inverse of the shortest distance between, $I_{sd}^{(A,B)}$ the the pair of nodes (A, B) representing the distance between (A, B). The aforementioned terms as follows.

$$D_p^{(A,B)} = d_A d_B \tag{1}$$

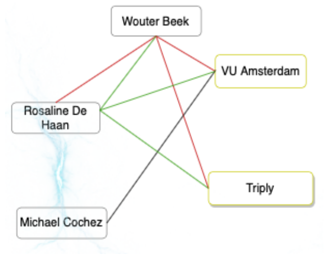
where d_A and d_B represent the degree of nodes A and B,

$$S_{C_z}^{(A,B)} = \sum_{z \in \Gamma(A) \cap \Gamma(B)} C_z \tag{2a}$$

$$C_z = \frac{2t_z}{d_z(d_z - 1)} \tag{2b}$$



(a) PE for Bipartite Graph



(b) PE for Non-bipartite graph

Figure 2: (a) Depiction of the potential energy, represented as a light blue lightning , for nodes A and B. in a Bipartite graph.(b)Depiction of the potential energy, represented as a light blue lightning , for nodes A and B in a Non-bipartite graph. The green and red links represent the triangles used in the calculation of the clustering coefficients for the common neighbours of node A (i.e. *VU Amsterdam*) and B (i.e. *Triply*)

where $\Gamma(A)$ and $\Gamma(B)$ represent the set of neighbours of A and B, t_z represents the number of triangles passing through node z, and d_z represents the degree of node z,

$$R_{sd}^{(A,B)} = \frac{1}{sd(A, B)} \tag{3}$$

where $sd(A, B)$ represents the shortest distance between the pair of nodes (A, B). Consequently, the PE is defined as follows

$$PE_{AB} = D_p^{(A,B)} S_{C_z}^{(A,B)} I_{sd}^{(A,B)} \tag{4}$$

One constraint is imposed for one edge case, if $z = \emptyset$, the value of C_z will be constant and 0.1. Another important remark is that in case a knowledge graph has disconnected components, it is necessary to isolate such components, and calculate the PE for the nodes in each component individually. That it to avoid having $lim_{R_{sd}^{(A,B)} \to \infty}$, while the $sd(A, B) \to 0$.

3.2 Potential Energy for a Non-Bipartite Graph

As previously mentioned,in order to calculate the Potential Energy of a knowledge base, we are interested in the information shared by the subject and the object nodes of such, i.e. $(A_1, B_1) \in \mathcal{V}_1$ forming the knowledge base $(A_i, r, A_j) \in \mathcal{E}$. Compared to a bipartite graph, in this second case, there is a link, $r \in \mathcal{R}_1$ between the subject and object of the knowledge base(see Figure and 2b for comparison). The PE is still calculated in the same way as previously shown in Formula 4, however, the conceptual interpretation of PE slightly changes. In a Bipartite graph, the PE is seen as an imaginary pulling force, or traction, between two disconnected nodes, i.e. $(A, B) \in \mathcal{U}$. In the case of a non-bipartite graph, besides the same way of calculating the graph metric, the PE is construed to be representative of the strength of the connection between the subject and the object nodes of the considered knowledge base. Hence, the higher the value of the PE, the higher the strength of the relation. This information can be useful during link prediction since it can be utilized to add the extra topological information of the graph. Additionally, it can also be considered as an attribute to characterize the knowledge base relation.

4 Neural Topological Modeling

Our model is primarily motivated as an extension of the R-GCN (Kipf and Welling 2017) enhanced with the PE metric.

4.1 Graph Convolutional Networks

Graph Convolutional Netoworks (GCNs) have the main goal to learn a function of signals/features on a structured graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ which takes as inputs (1st) a feature vector x_i for each node i; encoded in a $N \times D$ feature matrix, where and N is the number of nodes and D is the number of input features, and (2nd) the graph structure represented as adjacency matrix A. As output Z, at node-level, is an $N \times F$ feature matrix, where F number of output features. Every neural network layer follows a very simple propagation rule

$$H^{(l+1)} = f(H^{(l)}, A) \tag{5}$$

where $H^{(0)}$ is the node feature matrix X_i for node x_i, while, $H^{(L)}$ = Z and L is the number of layers (Kipf and Welling 2017). In specific terms, these and related models can be viewed as special cases of the node-related Message Passing mechanism introduced by Gilmer et al. (2017):

$$h_i^{(l+1)} = \sigma(\sum_{m \in \mathcal{M}} g_m(h_i^{(l)}, h_j^{(l)}), \tag{6}$$

where $h_i^{(l)} \in \mathcal{R}^{d^{(l)}}$ is the hidden state of node v_i in the l-th layer with $d^{(l)}$ being the dimensionality of l-th's representations. The incoming messages per node v_i, $\mathcal{M}_i$, are of the form $g_m(\cdot, \cdot)$ are accumulated and passed via a non-linear activation function σ, i.e. $ReLU(\cdot)$. $g_m(\cdot, \cdot)$ is typically picked to be neural-network like function or simply a linear transformation, i.e. Wh_i where W denotes the weight matrix as shown in (Kipf and Welling 2017).

4.2 Relational Graph Convolutional Networks

The Relational graph convolutional networks (RGCNs) are an extension of the GCNs that operate on large-scale relational data instead of local graph neighborhoods. The main difference resides in the definition of the propagation model for calculating the forward-pass update of an entity denoted by v_i

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right), \tag{7}$$

where $\mathcal{N}_i^r$ denotes the set of neighbor indices of node i under relation $r \in \mathcal{R}$. As explained by (Kipf and Welling 2017), 7 accumulates the transformed feature vectors of neighbors via a normalized sum, however, in addition to the standard GCNs, a relation-specific transformation is added. In general terms, a neural network update consist of evaluating 7 for each node in the graph.

R-GCN for Link Prediction The link prediction task deals with the prediction of new facts; formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, and given an incomplete set of edges, $\hat{\mathcal{E}}$, the objective is to assign scores to potential existing edges (s, r, o) in order to determine how likely they are to belong to $\mathcal{E}$, the complete set of edges of $\mathcal{G}$. The R-GCN tackles this problem by the introduction of an auto-encoder model, made up of an entity encoder and a decoder as scoring function. The encoder each entity $v_i \in \mathcal{V}$ to a real-valued vector $e_i \in \mathcal{R}^d$, meanwhile, the decoder, which in this case is the DistMult factorization, scores each predicted new triple as

$$f(s, r, o) = e_s^T R_r e_o, \tag{8}$$

where R_r is a diagonal matrix associated with each relation r, e_s^T is the transpose of the output of the encoder for the subject, meanwhile, e_o is the output of the encoder for the object. Then, the cross-entropy loss is used as main optimization objective.

4.3 Message-passing Neural Networks

Neural networks on graphs can be classified as message-passing neural networks (MPNNs). There are two main different variants: anonymous MPNNs whose Message Passing (MP) functions depend on only on the labels or the considered vertices; and degree-aware MPNNs whose message functions can use the information regarding the degrees of vertices (Geerts, Mazowiecki, and Pérez 2020). In our research, we treat degree-aware MPNNs; so, our R-GCN deploys an iterative neighbourhood aggregation of features combined with the vertex-degree information. It is important to remark that by using these kind of networks and by taking the computation rounds of the Weisfeiler-Lehman (WL) algorithm into consideration, the degree information may boost the distinguishing power of these MPNNs models (Geerts, Mazowiecki, and Pérez 2020). Nevertheless, the WL algorithm still misses to capture the connected components and cycles, topological features known for characterising graphs (Rieck, Bock, and Borgwardt 2019). Moreover, some promising results already showed that by learning the structural information of a KG, conjointly with the learning of the centrality and positional properties of the KG entities in one model, it improves the expressive power of some MPNNs models by being specifically beneficial for the link prediction task (Sadeghi et al. 2021).

4.4 Enhanced R-GCN

The R-GCN for link prediction provided by (Kipf and Welling 2017), as stated previously, can be seeing as a subclass of message passing neural networks (Gilmer et al. 2017), as such it can be enhanced by leveraging the graph topological information obtained with the PE. In order to leverage the topological information given by the PE we use Multi-task learning (MTL).

Multi-Task Learning We define Multi-task learning (MTL) as the improvement of learning a task by the addition of an external source of knowledge via the same learning protocol (see Figure 3). In our case, the task is inductive link prediction, the additional source or knowledge added to our

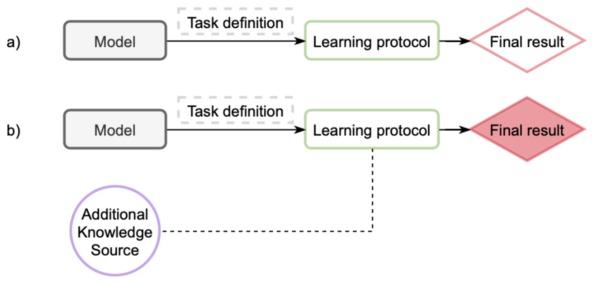


Figure 3: a) shows the standard ML workflow b) shows the Accelerated Learning ML workflow. The task definition remains the same, however, throughout the learning protocol, an additional knowledge source is leveraged to get improved final results.

model is the PE, and the learning protocol we adopt consists of the steps in Figure 4. In general terms, the learning protocol should include the model weight freezing (Brock et al. 2017) in order to include the additional knowledge source (see Figure 4 for an example). Similarly to Transfer Learning (TL) Torrey and Shavlik (2009), MTL helps by improving the overall model performance by having either a higher initial performance achievable, a steeper increase, thus, reducing the learning time, or by having higher asymptotic convergence, so, a higher final performance achievable.

Topological Modeling The PE is used as Source-Task Knowledge for TL with the R-GCN for link prediction. We use a Multilayer perceptron, stacked on top of the R-GCN, as a feedforward artificial neural network (ANN) to transfer the topological information to the R-GCN. In addition, we also add an initial Dense Layer, for dimensionality reduction, before the R-GCN encoder to create more informative node embeddings by including also textual information (see Figure 4). As our initial node representation, we use the linearly transformed BERT embeddings(see Appendix B) (Devlin et al. 2019).

The R-GCN decoder is enhanced with the output of the MLP as additional knowledge source regarding the topological graph structure as follows

$$f(s,r,o) = (1-\alpha)e_s^T R_r e_o + \alpha Z_{out}, \tag{9}$$

where Z_{out} is the resulting MLP output node embedding including the potential energy information and α is a scaling factor.

5 Link Prediction Use Case

We use the R-GCN as an effective encoder for relational data and our decoder model with a scoring function - DistMult augmented with the PE information - to score candidate triples for scholarly link prediction.

5.1 Data Preprocessing

We evaluate our model on one Scholarly Knowledge Graph, namely the IOS LD Connect Scholarly Knowledge Graph[1]

[1]http://ld.iospress.nl

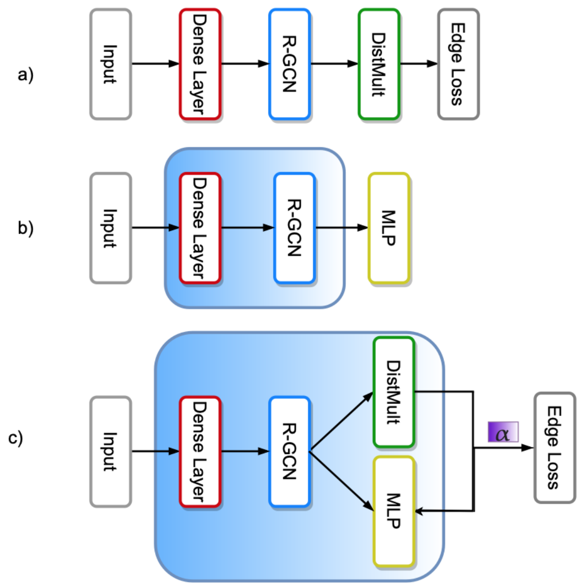


Figure 4: a) shows the R-GCN for link prediction with the additional Dense Layer for the creation of the textual embeddings, b) shows the R-GCN with the added MLP. After the initial R-GCN training, the parameter in the R-GCN for link prediction are frozen, shown by the light blue box around the Dense Layer, the R-GCN and DistMult, then the MLP is added and trained with the PE. c) Show the final training with the enhanced DistMult decoder optimized for α with the MLP perceptron output.

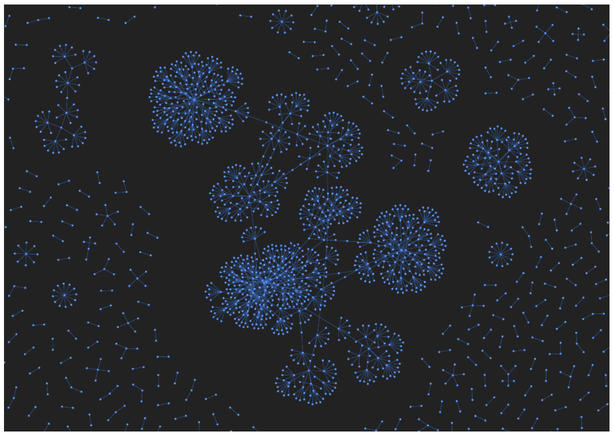


Figure 5: Visual representation of the IOS Press LD Connect Knowledge Graph

(see Figure 5). All the metadata about the papers are serialized and published as Linked Data by following the bibliographic ontology [2] and a SPARQL endpoint [3] and a deref-

[2]http://bibliontology.com/#sec-sioc-rdf
[3]http://ld.iospress.nl:3030

erence interface [4] are provided. We use the TriplyDB interface[5] to retrieve the relevant knowledge bases with the SPARQL service. As first data cleaning step, we remove all the Geometric triples which are not relevant for our use case. After this initial cleaning step, we proceed with gathering all the relevant triples for scholarly link prediction. We retrieve the following entities: authors, articles and affiliations, with their respective textual attributes; author's full name, article's titles, keywords and abstract, and affiliation names. Hence, we specifically select the instances of class *ontology:Organization*, *ontology:Contributor* and *ontology:Publication*. In addition to the selected entities with their respective textual features, we also want to retrieve the *owl:sameAs* relations between same *ontology:Contributor*s and the *ontology:publicationAuthorList* attribute for each entity of class *ontology:Article*. Therefore, we retrieve all the relevant data with a CONSTRUCT SPARQL query, see Appendices A.4 and A.5 for the code. We decide to retrieve the 2 extra pieces of information, *owl:sameAs* and *ontology:publicationAuthorList*, for the instances of *ontology:Contributor* and *ontology:Publication* respectively (see Appendix A.1 and tables 2 for the graph visualization and its statistics). The main use of this information is to identify and reduce the equivalence relations, *owl:sameAs*, in order to find the authors who collaborate in the same article; who belong to in the same author list. The merging of the data is necessary in order not to have duplicate authors in the dataset, meanwhile, the instances of *ontology:publicationAuthorList* are used to add the co-authorship relations during the data 'smushing' phase.As it is possible to see from Appendix A.2, applying the "smushing" identifies the instance of the *ontology:sameAs* relation and add a new relation by merging the two entities connected by the equivalence relation/ It is important to notice that the "smushing" does not remove the instances of *owl:sameAs* relations, but it omits them by using *owl:sameAs* quads that link the entities connected by the sameAs relations to the main URI for that entity created at the start of the "smushing" (see graph statistics in 3). It also applies the transferring of the relations to the main URI for one entity identifying the nodes connected to it with the equivalence relation. This is represented by the red dotted line in Appendix A.2. On the other hand, by running the CONSTRUCT SPARQL query we also add a new relation between the instances of *ontology:Contributor* and the *ontology:Organization*. This is performed in order to have extra information for the instances *ontology:Contributor*s. Therefore, different authors have different affiliations they worked with. The "smushing" of the data is performed with rdfpro (Corcoglioniti et al. 2014). More specifically, we run the following command on the terminal:*rdfpro -V read file_with_sameAs..ttl smush write withoutSameAs.ttl*. The "smushed" data is then further cleaned and processed by removing all the entities without any textual description, which would not provide any additional information in our specific case, and by adding the co-authorship relations between authors in the same author list. Further cleaning is applied by removing the author lists and the sameAs relations, which would not serve any use since they do not provide any extra relevant information. Moreover, it is important to notice that by using the CONSTRUCT SPARQL query we also add the relation *hasArticle* between each instance of *ontology:Contributor* and its respective *ontology: Publication*. The final data before feeding it to the R-GCN is shown in Appendix A.3 with its stats in Table 4.

The data format after the data preprocessing of the data is Turtle. The turtle format allows to write triples in a compact and natural way, with text abbreviations for frequently recurring patterns. An example of turtle triples is given in snippet of code 6.

```
@prefix ns0: <https://schema.org/> .
@prefix ns1: <http://ld.iospress.nl/artifact/>.
@prefix ns2: <http://ld.iospress.nl/Author:0.ID:> .

ns2:BD160230 ns0:fullName "Megha Tandon" ;
    ns0:hasAffiliation ns242:BD160230 ;
    ns0:hasArticle ns1:bd-v38-i3-4-BD160230 .
```

Figure 6: Example of the Turtle rdf compact format

Hence, the next step is to adapt the data retrieved with the SPARQL CONSTRUCT QUERY in order to extract the relevant information to feed to the R-GCN. For this task, we use the python RDFlib [6], which is very useful for our goal since it has built-in functions to analyze RDF graph characteristics (e.i. # of predicates, # of objects, # of subjects).

5.2 Results

Baseline Since our specific choice of data set, the IOS Press LD Connect, leads to comparison constrains, as a baseline of our experiments, we compare against the standard R-GCN version for link prediction (Kipf and Welling 2017) without the enhanced DistMult decoder. The train-val-test edge split for the edges in the IOS Press LD Connect graph is 65%(13330), 20%(2525) and 0.15%(1893) over the total amount of edges (17748). More specifically, we sample one negative edge for each positive edge for each split. Hence, we first evaluate the standard R-GCN for link prediction on the IOS Press LD Connect and we gather the train loss and test AUC and ROC values. Second, we continue the training for the R-GCN with the MTL mechanism and the MLP. Lastly, we re-evaluate the enhanced R-GCN with the new optimized DistMult decoder, and we gather the test AUC and ROC values for such model(see Table 1 for the comparison of the results). In addition, we tune the hypeparameters of our model with Optuna (Akiba et al. 2019)(see Appendix D for more details).

Experimental Design For the vanilla R-GCN, we report performance of a 3-layer model with 14, and 32 hidden units per layer, basis function decomposition, and trained with

[4]http://ld.iospress.nl/browse/#iospress-organization:Publisher.IOS_Press

[5]https://triplydb.com/SimoneColombo/CleanedLDConnect/sparql

[6]https://rdflib.readthedocs.io/en/stable/

Adam Kingma and Ba (2017) for 300 epochs using a learning rate of 0.01. For the eR-GCN, we run a 3-layer model equal to the vanilla R-GCN in terms of hidden units and optimizer parameters, but with the different factorization method, namely DistMult with the PE. We train and evaluate the R-GCN, the MLP and the eR-GCN for the scholarly link prediction task. We report the averages of the train loss, validation and test accuracies over 10 runs in the Tables 1 and 2. Our main goal is to prove that the PE actually improves the overall perfomance of the R-GCN for the link prediction task. Further details on (baseline) models and hyperparameter choices are provided in the Appendix D and C.

Model	train loss	val AUC/ROC score	test AUC/ROC score
R-GCN	0.3782 ±0.0252	0.8923 ±0.0191	0.8912 ±0.0173
eR-GCN	0.3665 ±0.0001	0.8997 ±0.0001	0.8993 ±0.0001
scaling factor (α)	≈ 0.0		

Table 1: Comparison between the train loss, validation and test AUC and ROC curves of the R-GCN and the eR-GCN with the MLP over 30 runs

Retrospective Experiment Successively, we continue by running a retrospective experiment to deepen our understanding on the performance of the eR-GCN. We decide not to use the MLP for the PE prediction (see Figure 7), but to directly feed the PE values for each knowledge base into the DistMult decoder and still optimize for the scaling factor α (the results for comparison can be seen in Table 2).

Model	train loss	val AUC/ROC score	test AUC/ROC score
R-GCN	0.3782 ±0.0252	0.8923 ±0.0191	0.8912 ±0.0173
eR-GCN	0.2014 ±0.0352	0.9186 ±0.0230	0.9209 ±0.0221
scaling factor (α)	0.2925 ±0.0097		

Table 2: Retrospective comparison between the train loss, validation and test AUC and ROC curves of the R-GCN and the eR-GCN by including the PE directly in the decoder without using the MLP over 30 runs

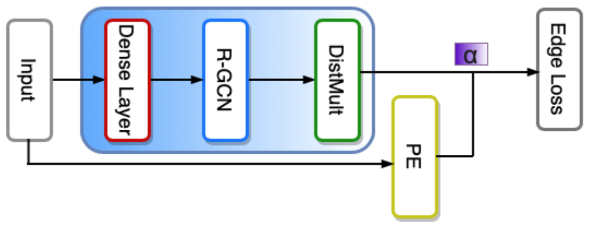


Figure 7: It shows the model used in the retrospective experiment. The eR-GCN is optimized for *alpha* and the PE is directly fed to the DistMult decoder without the use of the MLP. The light blue box around the Dense Layer, the R-GCN decoder and the DistMult decoder indicates that all the parameters of the final model are frozen besides the scaling factor α

6 Related Work

6.1 Topological modeling

Our encoder-decoder model approach to link prediction is based on DistMult in the decoder (Yang et al. 2015), enhanced by the Potential Energy as knowledge base graph metric (Kumar and Sharma 2020). Other alternatives factorization methods have been proposed (Bordes et al. 2013; Socher et al. 2013b; Trouillon et al. 2016), many of which are based on the tensor decomposition techniques (Kolda and Bader 2009). Link prediction methods can be split in 3 different categories; 1) predicting new auxiliary triples to add the the factorization method (Guu, Miller, and Liang 2015), 2) path-based methods have been used for predicting edges (Lin et al. 2015) and 3) both at the same time (Toutanova and Chen 2015). Our eR-GCN tends more towards the first direction, by extending the original decoder, DistMult, with the PE graph metric. The second direction, is relatively orthogonal to ours since our main focus is the addition of the new feature to the factorization method, however, the R-GNC encoder can be regarded as a computationally cheaper option than the path-based methods (Kipf and Welling 2017).

6.2 Neural Networks on graphs

Our eR-GCN is closely related to the R-GCN by Kipf and Welling (2017), and more specifically it focuses on the theoretical understanding of the concept of MPNNs (Gilmer et al. 2017), by enhancing it with the use of graph metrics. Previous research on graphs used graph metrics in Graph Neural Networks in two direction; 1) by enhancing the GCNs directly with the use of graph metrics (Li et al. 2020) and 2) by including graph information in the embeddings creation (Sadeghi et al. 2021). However, from the best of our knowledge, none of these directions considered the main GCNs underlying mechanism of Message Passing (MP) and the graph metric specifically related knowledge base structure, the PE.

7 Conclusions

We have introduced energized relational graph convolutional networks (eR-GCNs) and demonstrated its improvement with the Potential Energy as enriching factor in the DistMult decoder. We have also highlighted the downside of the generalization power of the Vanilla RGCN in modelling the knowledge base-specific graph metric. Moreover, enriching the factorization model with an PE metric proved especially valuable for the link prediction task, yielding a 0.3% improvement over the R-GCN baseline.

7.1 Future Work and Discussion

There are several ways in which our work could be extended. First, the vanilla structure of the R-GCN could be modified aiming to increase its generalization power towards knowledge base-specific graph metrics, as the PE. Hence, a potential research direction would be to analyze the depth of the R-GCN in correlation with its performance over the generalization power. Additionally, other graph metrics, e.g. node degrees, nodes shortest distance, etc., could be tested as additional values to the factorization method of the Vanilla

R-GCN, DistMult. This would aim to understand whether the R-GCN is not able to generalize for any external additional metric or whether it was a problem related to the PE specifically. To a wider spectrum, it could also be useful to understand the overall capacity of the R-GCN to generalize for external features added to the factorization method. Furthermore, the correlation between different factorization methods and the R-GCN ability to generalize new information could also be tested. Lastly, in order to further strengthen our findings it would be necessary to run more extensive experiments on data sets of different sizes for the task of link prediction. In turn, it would be possible to understand the effect of the data set size on the ability of the R-GCN to generalize new knowledge.

8 Acknowledgements

I would like to thanks my two mainsupervisors Michael Cochez and Dimitris Alivanistos for the guidance and the interesting discussion and the people at Triply for help throughout the data preprocessing phase and the useful comments.

References

[Abu-Salih 2021] Abu-Salih, B. 2021. Domain-specific knowledge graphs: A survey.

[Akiba et al. 2019] Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A next-generation hyperparameter optimization framework.

[Ayala-Gómez et al. 2018] Ayala-Gómez, F.; Daróczy, B.; Benczúr, A.; Mathioudakis, M.; and Gionis, A. 2018. Global citation recommendation using knowledge graphs. *J. Intell. Fuzzy Syst.* 34:3089–3100.

[Bordes et al. 2013] Bordes, A.; Usunier, N.; Garcia-Durán, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, 2787–2795. Red Hook, NY, USA: Curran Associates Inc.

[Brock et al. 2017] Brock, A.; Lim, T.; Ritchie, J. M.; and Weston, N. 2017. Freezeout: Accelerate training by progressively freezing layers.

[Bronstein et al. 2021] Bronstein, M. M.; Bruna, J.; Cohen, T.; and Veličković, P. 2021. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges.

[Corcoglioniti et al. 2014] Corcoglioniti, F.; Rospocher, M.; Amadori, M.; and Mostarda, M. 2014. Rdfpro: An extensible tool for building stream-oriented rdf processing pipelines. In *Proceedings of the 2014 International Conference on Developers - Volume 1268*, ISWC-DEV'14, 49–54. Aachen, DEU: CEUR-WS.org.

[Devarajan 2017] Devarajan, D. 2017. Happy birthday watson discovery. *IBM Cloud Blog*. https://www.ibm.com/cloud/blog/announcements/happy-birthday-watson-discovery.

[Devlin et al. 2019] Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

[Fey and Lenssen 2019] Fey, M., and Lenssen, J. E. 2019. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[Futia and Vetrò 2020] Futia, G., and Vetrò, A. 2020. On the integration of knowledge graphs into deep learning models for a more comprehensible ai—three challenges for future research. *Information* 11(2).

[Gao et al. 2020] Gao, Y.; Li, Y.-F.; Lin, Y.; Gao, H.; and Khan, L. 2020. Deep learning on knowledge graph for recommender system: A survey.

[Geerts, Mazowiecki, and Pérez 2020] Geerts, F.; Mazowiecki, F.; and Pérez, G. A. 2020. Let's agree to degree: Comparing graph convolutional networks in the message-passing framework.

[Gilmer et al. 2017] Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In Precup, D., and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 1263–1272. PMLR.

[Guu, Miller, and Liang 2015] Guu, K.; Miller, J.; and Liang, P. 2015. Traversing knowledge graphs in vector space.

[Jaradeh, Stocker, and Auer 2020] Jaradeh, M. Y.; Stocker, M.; and Auer, S. 2020. Question answering on scholarly knowledge graphs.

[Ji et al. 2021] Ji, S.; Pan, S.; Cambria, E.; Marttinen, P.; and Yu, P. S. 2021. A survey on knowledge graphs: Representation, acquisition and applications.

[KGC 2021] 2021. Knownledge graph conference. https://www.knowledgegraph.tech.

[KGD 2020] 2020. Workshop: Deep learning for knowledge graph. https://alammehwish.github.io/dl4kg_eswc_2020/.

[KGI 2021] 2021. International conference on knowledge graph. https://kmeducationhub.de/ieee-international-conference-big-knowledge-icbk/.

[Kingma and Ba 2017] Kingma, D. P., and Ba, J. 2017. Adam: A method for stochastic optimization.

[Kipf and Welling 2017] Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks.

[Kolda and Bader 2009] Kolda, T. G., and Bader, B. W. 2009. Tensor decompositions and applications. *SIAM Rev.* 51(3):455–500.

[Krishnan 2018] Krishnan, A. 2018. Making search easier: How amazon's product graph is helping customers find products more easily. *Amazon Blog*. https://blog.aboutamazon.com/innovation/making-search-easier.

[Kumar and Sharma 2020] Kumar, P., and Sharma, D. 2020. A potential energy and mutual information based link prediction approach for bipartite networks. *Scientific Reports* 10.

[Li et al. 2020] Li, X.; Wei, W.; Feng, X.; Liu, X.; and Zheng,

Z. 2020. Representation learning of graphs using graph convolutional multilayer networks based on motifs.

[Lin et al. 2015] Lin, Y.; Liu, Z.; Luan, H.; Sun, M.; Rao, S.; and Liu, S. 2015. Modeling relation paths for representation learning of knowledge bases.

[Nayyeri et al. 2020] Nayyeri, M.; Vahdati, S.; Zhou, X.; Yazdi, H. S.; and Lehmann, J. 2020. Embedding-based recommendations on scholarly knowledge graphs. In *European Semantic Web Conference*, 255–270. Springer.

[Nickel et al. 2016] Nickel, M.; Murphy, K.; Tresp, V.; and Gabrilovich, E. 2016. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE* 104(1):11–33.

[Noy et al. 2019] Noy, N.; Gao, Y.; Jain, A.; Narayanan, A.; Patterson, A.; and Taylor, J. 2019. Industry-scale knowledge graphs: Lessons and challenges. *Commun. ACM* 62(8):36–43.

[Oelen et al. 2020] Oelen, A.; Jaradeh, M. Y.; Stocker, M.; and Auer, S. 2020. Generate fair literature surveys with scholarly knowledge graphs. *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*.

[Rieck, Bock, and Borgwardt 2019] Rieck, B.; Bock, C.; and Borgwardt, K. 2019. A persistent weisfeiler-lehman procedure for graph classification. In Chaudhuri, K., and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 5448–5458. PMLR.

[R.J. Pittman 2017] R.J. Pittman, Amit Srivastava, S. H. 2017. Cracking the code on conversational commerce. *eBay Blog*. https://blog.aboutamazon.com/innovation/making-search-easier.

[Sadeghi et al. 2021] Sadeghi, A.; Collarana, D.; Graux, D.; and Lehmann, J. 2021. Embedding knowledge graphs attentive to positional and centrality qualities.

[Schlichtkrull et al. 2017] Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; van den Berg, R.; Titov, I.; and Welling, M. 2017. Modeling relational data with graph convolutional networks.

[Singhal 2012] Singhal, A. 2012. Introducing the knowledge graph: things, not strings. *Google Blog*. https://blog.google/products/search/introducing-knowledge-graph-things-not/.

[Socher et al. 2013a] Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. Y. 2013a. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, 926–934. Red Hook, NY, USA: Curran Associates Inc.

[Socher et al. 2013b] Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. Y. 2013b. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, 926–934. Red Hook, NY, USA: Curran Associates Inc.

[Torrey and Shavlik 2009] Torrey, L. A., and Shavlik, J. 2009. Chapter 11 transfer learning.

[Toutanova and Chen 2015] Toutanova, K., and Chen, D. 2015. Observed versus latent features for knowledge base and text inference.

[Trouillon et al. 2016] Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, E.; and Bouchard, G. 2016. Complex embeddings for simple link prediction. In Balcan, M. F., and Weinberger, K. Q., eds., *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, 2071–2080. New York, New York, USA: PMLR.

[Vahdati et al. 2018] Vahdati, S.; Palma, G.; Nath, R. J.; Lange, C.; Auer, S.; and Vidal, M.-E. 2018. Unveiling scholarly communities over knowledge graphs.

[Wu et al. 2021] Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2021. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32(1):4–24.

[Yang et al. 2015] Yang, B.; tau Yih, W.; He, X.; Gao, J.; and Deng, L. 2015. Embedding entities and relations for learning and inference in knowledge bases.

[Yao et al. 2019] Yao, S.; Wang, R.; Sun, S.; Bu, D.; and Liu, J. 2019. Joint embedding learning of educational knowledge graphs.

A Further details about the dataset preprocessing

Tables 1 represent the IOS Press Data set after removing the Geometric triples.

IOS PRESS ENTITIES		
Class Name	**# of Instances**	%
ontology:Organization	325.487	≈42%
ontology:Contributor	318.116	≈41%
ontology:Publication	92.000	≈11%
ontology:GeocodedLocation	42.023	≈5%
ontology:Issue	8.843	≈0,8%
ontology:Volume	2.433	≈0,3%
ontology:Journal	128	<0,1%
ontology:Category	9	<0,1%
ontology:Publisher	8	<0,1%

IOS PRESS FEATURES		
Class Name	**Relation Type**	**# of Feature Instances**
ontology:Organization	ontology:OrganizationName	167.321
ontology:Contributor	ontology:ContributorFullName	318.116
ontology:Publication	ontology:publicationIncludesKeywords	63.668
ontology:Publication	ontology:publicationAbstract	75.322
ontology:Publication	ontology:publicationTitle	92.000

IOS PRESS EXTRA INFO		
Class Name	**Relation Type**	**# of Feature Instances**
ontology:Contributor	owl:sameAs	105.271
ontology:Publication	ontology:publicationAuthorList	84.044

Table 1: Entities statistics and features for the IOS Press LD Connect knowledge graph before preprocessing
.

A sample visual representation of the data we want to retrieve can be seen in Figure A.1.

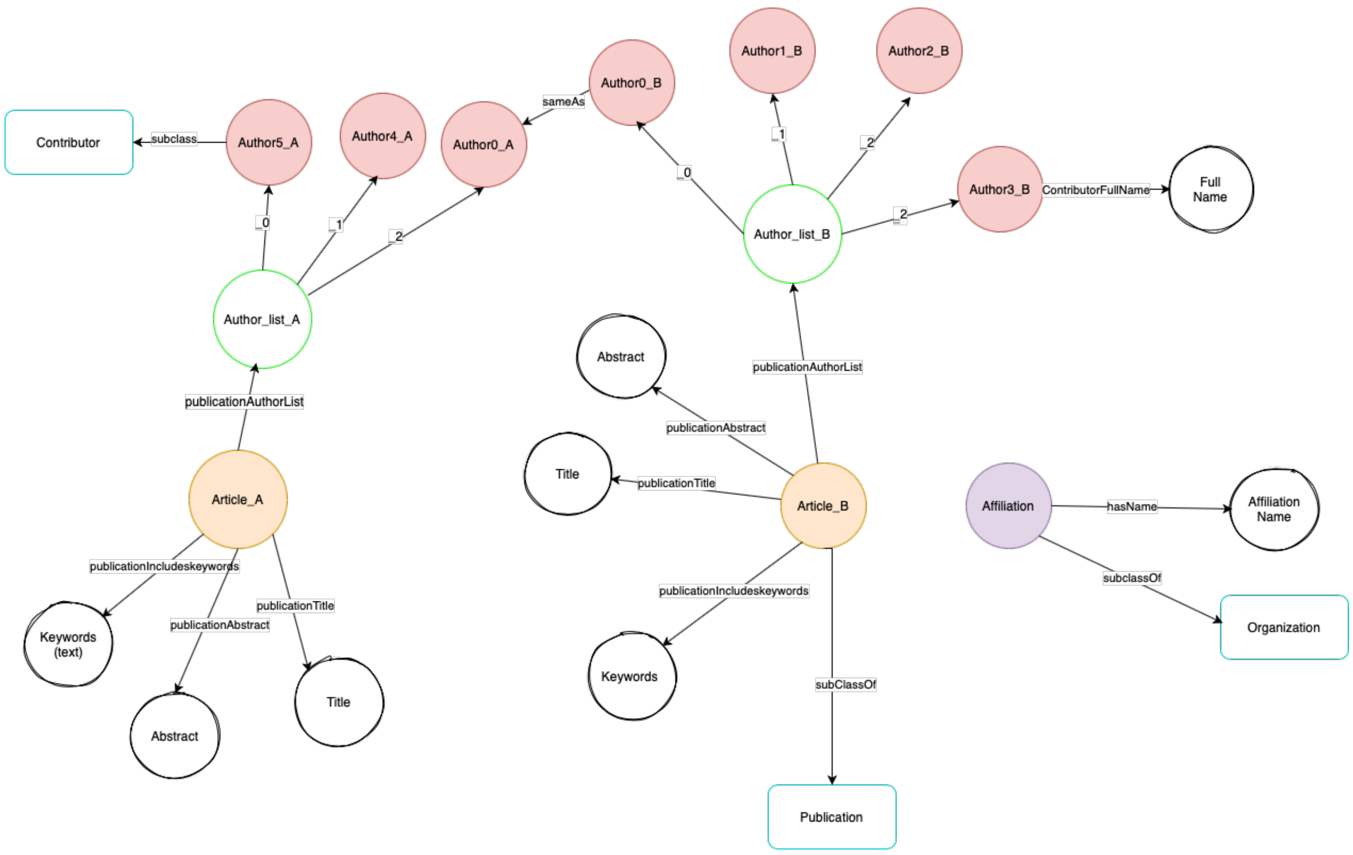


Figure A.1: IOS PRESS data after initial cleaning and with the selected entities and features

Overall, Table 2 summarizes the entity and feature statistics after the initial preprocessing.

IOS PRESS ENTITIES		
Class Name	**# of Instances**	%
ontology:Organization	2946	≈49%
ontology:Contributor	2031	≈33%
ontology:Publication	1077	≈18%

IOS PRESS FEATURES		
Class Name	**Relation Type**	**# of Feature Instances**
ontology:Contributor	ontology:OrganizationName	2946
ontology:Organization	ontology:ContributorFullName	2031
ontology:Publication	ontology:publicationIncludesKeywords	1077
ontology:Publication	ontology:publicationAbstract	1077
ontology:Publication	ontology:publicationTitle	1077

IOS PRESS EXTRA INFO		
Class Name	**Relation Type**	**# of Feature Instances**
ontology:Contributor	owl:sameAs	2031
ontology:Publication	ontology:publicationAuthorList	7300

Table 2: Entity and feature statistics for the IOS Press LD Connect knowledge graph after the initial preprocessing

.

Among initial data set, we specifically select the instances of class *ontology:Organization*, *ontology:Contributor* and *ontol-*

ogy:Publication. The visual representation of the data after the "smushing" is shown in Figure A.2.

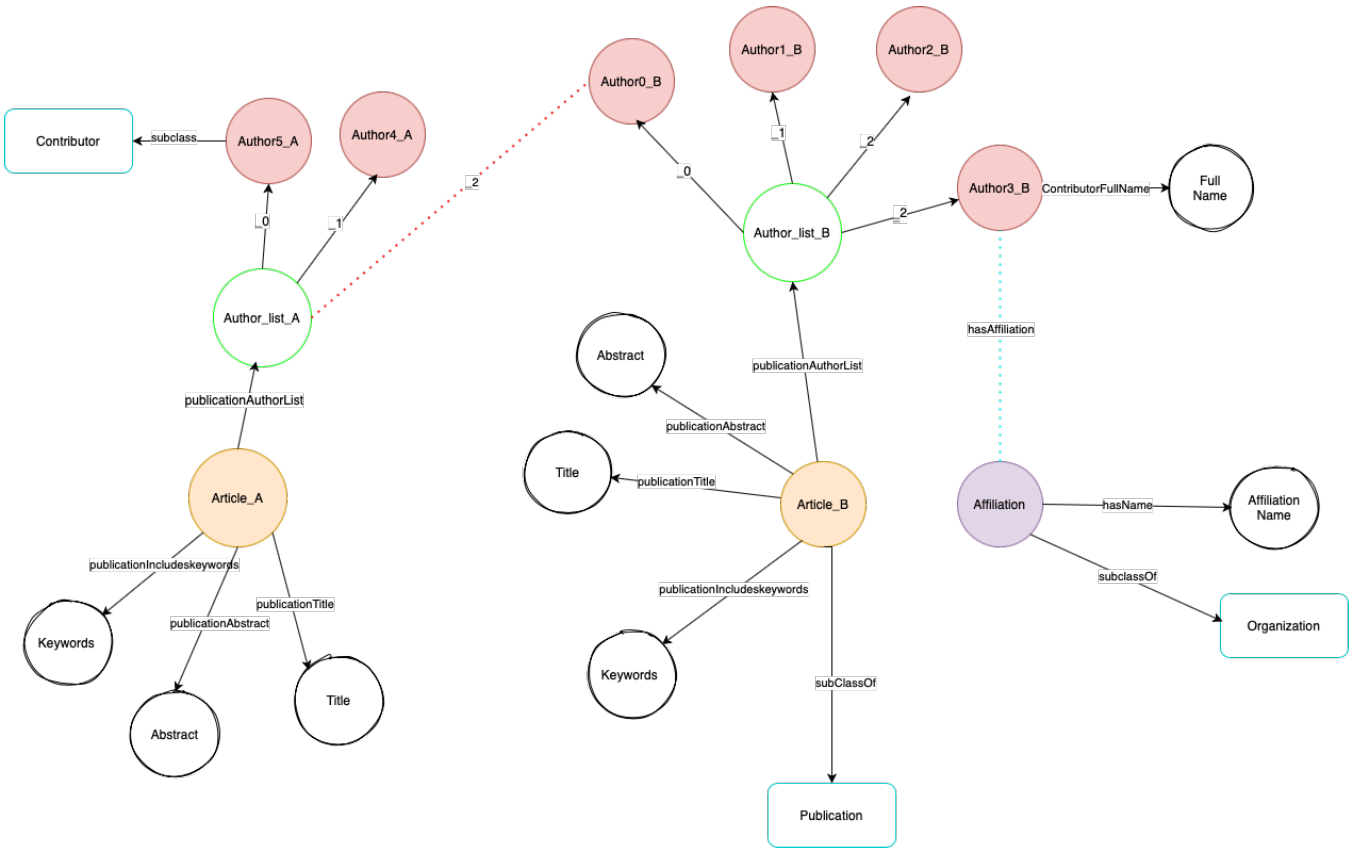


Figure A.2: IOS PRESS data after CONSTRUCT SPARQL query and the "smushing"

The respective entity and feature statistics for the data set after the smushing are shown in Table 3

IOS PRESS ENTITIES AFTER SMUSHING		
Class Name	**# of Instances**	%
ontology:Organization	2946	≈50%
ontology:Contributor	1907	≈32%
ontology:Publication	1077	≈18%

IOS PRESS FEATURES		
Class Name	**Relation Type**	**# of Feature Instances**
ontology:Organization	ontology:OrganizationName	2946
ontology:Contributor	ontology:ContributorFullName	1907
ontology:Publication	ontology:publicationIncludesKeywords	1077
ontology:Publication	ontology:publicationAbstract	1077
ontology:Publication	ontology:publicationTitle	1077

IOS PRESS EXTRA INFO		
Class Name	**Relation Type**	**# of Feature Instances**
ontology:Contributor	owl:sameAs	1907

Table 3: Entity and feature statistics for the IOS Press LD Connect knowledge graph after smushing

.

The final data for the RGCN is shown in Figure A.3 and the entity and feature statistics can be found in Table 4.

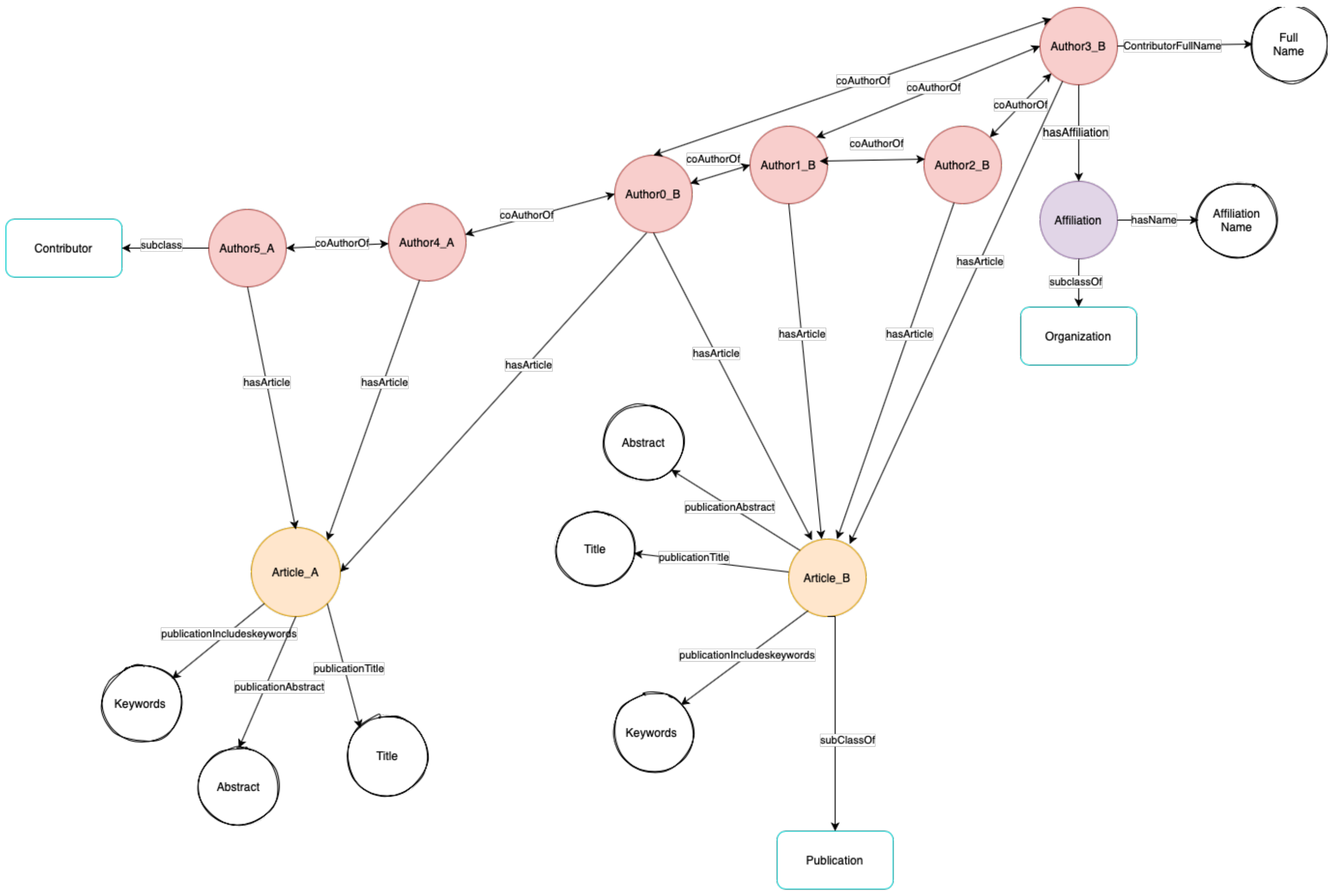


Figure A.3: IOS PRESS data after final preprocessing ready to feed to the RGCN

IOS PRESS ENTITIES AFTER SMUSHING		
Class Name	**# of Instances**	%
ontology:Organization	2946	≈50%
ontology:Contributor	1907	≈32%
ontology:Publication	1077	≈18%

IOS PRESS FEATURES		
Class Name	**Relation Type**	**# of Feature Instances**
ontology:Organization	ontology:OrganizationName	2946
ontology:Contributor	ontology:ContributorFullName	1907
ontology:Publication	ontology:publicationIncludesKeywords	1077
ontology:Publication	ontology:publicationAbstract	1077
ontology:Publication	ontology:publicationTitle	1077

IOS PRESS EXTRA INFO		
Class Name	**Relation Type**	**# of Feature Instances**
ontology:Contributor	ontology:coAuthorOf	1569
ontology:Contributor	ontology:hasArticle	1907

Table 4: Entity and feature statistics for the IOS Press data after the final preprocessing
.

All the statistics above are retrieved by the use of the TriplyDB interface [7].

SPARQL Construct Query

```
PREFIX sd: <http://www.w3.org/ns/sparql-service-description#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX ios: <http://ld.iospress.nl/rdf/ontology/>
PREFIX sdo: <https://schema.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

CONSTRUCT {

  ?article sdo:hasKeywords ?pubKeywords.
  ?article sdo:hasPubTitle ?pubTitle.
  ?article sdo:hasPubAbstract ?pubAbstract.
  ?article sdo:hasPubDate ?pubDate.
  ?author sdo:hasArticle ?article.
  ?author sdo:inAuthorList ?pubAuthorList.
  ?author sdo:hasAffiliation ?authorAffiliation.
  ?authorAffiliation sdo:hasName ?label.
  ?author sdo:fullName ?fullName.
  ?author owl:sameAs ?author_.
  ?author_ sdo:hasAffiliation ?authorAffiliation_.

}
WHERE {
  ?article rdf:type ios:Article.
  ?article ios:publicationIncludesKeyword ?pubKeywords.
  ?article ios:publicationDate ?pubDate.
  ?article ios:publicationTitle ?pubTitle.
  ?article ios:publicationAbstract ?pubAbstract.
  ?article ios:publicationAuthorList ?pubAuthorList.
  ?pubAuthorList ?p ?author.
  ?author ios:contributorAffiliation ?authorAffiliation.
  ?author ios:contributorFullName ?fullName.
  ?author owl:sameAs ?author_.
  ?author_ ios:contributorAffiliation ?authorAffiliation_.
  ?authorAffiliation ios:organizationName ?label.
  filter (?pubDate >= "2017-11-01"^^xsd:dateTime).
}
```

Figure A.4: SPARQL Construct Query used to retrieve all the data from the IOS Press LD Connect Knowledge Graph

[7] https://triplydb.com/ios-press/ld-connect/insights/classFrequency

SPARQL Construct Query with JS

It is important to remark that we specifically use the TriplyDB in combination with the javascript service [8] in order to return more than 10K triples; otherwise not possible due to a Virtuoso limit with the CONSTRUCT SPARQL queries. The query A.5 shows how we use the TriplyDB service with JS.

```
"use strict";
var __importDefault = (this && this.__importDefault) || function (mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};
Object.defineProperty(exports, "__esModule", { value: true });
exports.run = void 0;
require("source-map-support/register");
const triplydb_1 = __importDefault(require("@triply/triplydb"));
async function run() {
    const app = triplydb_1.default.get({ token: process.env.TRIPLY_API_TOKEN });
    // get organization that contains the to query Dataset.
    const user = await app.getUser("SimoneColombo");
    // Get query information
    const query = await user.getQuery("IOS-Press-K-Gfor-RGCN");
    // for construct and describe queries
    const results = query.results().statements();
    // saving the results to file
    await results.toFile(`./more_than_10K.ttl`);
}
exports.run = run;
run().catch((e) => {
    console.error(e);
    process.exit(1);
});
process.on("uncaughtException", function (err) {
    console.error("Uncaught exception", err);
    process.exit(1);
});
process.on("unhandledRejection", (reason, p) => {
    console.error("Unhandled Rejection at: Promise", p, "reason:", reason);
    process.exit(1);
});
```

Figure A.5: SPARQL CONSTRUCT Query call with NodeJS by using the Triply interface in order to retrieve more than 10K triples

B Further information about the use of the pretrained BERT model

BERT uses an attention mechanism to learn contextual relations between words, or sub-words, in a text; this is called Transformer. In its simplest form, the vanilla form, a Transformer includes two separate parts — an encoder that reads the text input and a decoder that outputs a prediction for the task. Since BERT's task is to generate a language model, the only necessary part is the decoder. Differently from directional models, that read the text input sequentially, the Transformer encoder reads the whole text sequence at once. Hence it is considered bidirectional, even if it would be more accurate to define it as non-directional. This essential characteristic of BERT allows it to learn the context of a word based on all of its surrounding words, hence, it is regarded as one of the best model to encode textual information (Devlin et al. 2019). The structure of the R-GCN model we propose in our research has an initial dense layer used for the dimensionality reduction of the BERT input embeddings. The dense layer is added in order to check whether the BERT embeddings can be reduced in dimension while still retaining their encoded information. We detach the computational graph for the BERT-embedding creation from the R-GCN computational graph. We found this step essential for the R-GCN for link prediction with textual information since without this detachment the backward pass for the loss retains the entire computational graph. Nevertheless, in order reduce memory usage, PyTorch,

[8]https://triply.cc/docs/triplydb-js

during the '.backward()' call, deletes all the intermediary results when they are not needed anymore. In our specific case, the weight matrices for the embeddings can be calculated only once, without having the need to free or delete them, hence we detach the BERT computational graph. In more details, the error we obtained was the following: "*RuntimeError: Trying to backward through the graph a second time, but the buffers have already been freed. Specify retain_graph=True when calling backward the first time*" [9], and we solved it with '.detach()' (see B.6)

```
def get_bert_embedding(list_of_strings):
  tensor_length = len(list_of_strings)
  result_embedding = torch.zeros(tensor_length, 768)
  for sentence in list_of_strings:
    input_ids = torch.tensor(tokenizer.encode(sentence[:512])).unsqueeze(0)  #batch size 1
    outputs = model(input_ids)
    embeddings_of_last_layer = outputs[0]
    cls_embeddings = embeddings_of_last_layer[0]
    result_embedding[list_of_strings.index(sentence)] = torch.sum(cls_embeddings, dim=0)
  return torch.sum(result_embedding, dim=0)

def create_entities_embeddings(rdf_graph, entities_dict, embedding_dimension):
  entities_IDs_to_embeddings = torch.zeros(len(entities_dict), embedding_dimension)
  #for loop through all unique entities and their IDs
  for rdf_entity, entity_ID in entities_dict.items():
    entity_textual_information = []
    #for loop through all the triples in the graph which have as entity the
    #entity for entity_ID and where the entity e2 is a textual attribute for e1
    for e1,r,e2 in rdf_graph.triples((rdf_entity, None, None)):
      #check for literal attribute for entity e1
      if type(e2) == rdf.term.Literal:
      #create embedding for entity embeddings
        entity_textual_information.append(e2)
    if len(entity_textual_information) > 0:
      entities_IDs_to_embeddings[entity_ID]  =  get_bert_embedding(entity_textual_information)

  return entities_IDs_to_embeddings.detach()
```

Figure B.6: Code snippet for the creation of the BERT embeddings for textual entities

C Further details about the R-GCN implementation

By using the rdflib Python library, we read the triples from the turtle file and we iterate through them in order to build the sets for subjects, relations and objects. Successively, based on the aforementioned sets, we create the necessary arguments to feed as input to the R-GCN. In order to implement the RGCN we use PyTorch Geometric (Fey and Lenssen 2019). PyTorch Geometric is a geometric deep learning extension library for PyTorch. It provides various built-in methods for deep learning on graphs, also called geometric deep learning (Bronstein et al. 2021), and it also provides a set of common benchmark datasets to use. The PyTorch geometric implementation of the RGCN for link prediction [10] is based on the paper by (Schlichtkrull et al. 2017), and it runs in a full-batch fashion on CPU. Our goal is to use the RGCN for link prediction on GPU with the additional necessary embeddings for the textual information for each unique entity[11]. We achieve the former by running our program on the SURF Sara Lisa computer cluster and by moving all the tensors on GPU, and the latter by the use the BERT bidirectional transfomer (as explained in B). After the embeddings creation, we start generating the necessary arguments for the R-GCN. The R-GCN initialization module can be seen in the code snippet C.7

[9]https://discuss.pytorch.org/t/runtimeerror-trying-to-backward-through-the-graph-a-second-time-but-the-buffers-have-already-been-freed-specify-retain-graph-true-when-calling-backward-the-first-time/6795

[10]https://github.com/pyg-team/pytorch_geometric/blob/master/examples/rgcn_link_pred.py

[11]Our implementation of the R-GCN for link prediction in pytorch geometric can be found at https://github.com/simoneVU/EnhancedRGCN

```
def __init__(self, in_channels: Union[int, Tuple[int, int]],
                out_channels: int,
                num_relations: int,
                num_bases: Optional[int] = None,
                num_blocks: Optional[int] = None,
                aggr: str = 'mean',
                root_weight: bool = True,
                bias: bool = True, **kwargs):

       super(RGCNConv, self).__init__(aggr=aggr, node_dim=0, **kwargs)
```

Figure C.7: PyTorch Geometric R-GCN initialization module

The R-GCN has 3 non-optional arguments; in_channels, out_channels and num_relations,and 2 optional ones; the num_blocks and the num_bases. The non-optional arguments represent the dimension of the input embedding for each node, the dimension of the output embedding for each node and the total number of relations in the KG respectively. The optional arguments can be set to a certain number of bases or blocks in order to use either tensor or block decomposition to create the R-GCN embeddings. In addition, the aggregation function for the message passing is set to the 'mean' and both the root_weight and the bias are initialized.

D OPTUNA hyperparameters autotuning

Hyperparameters Autotuning						
trial nr.	**in_channels**	**out_channels**	**hidden_channels**	**learning_rate**	**num_bases**	**validation accuracy**
0	192	16	8	6.0532e-06	6	0.7133
1	32	24	14	4.2222e-05	3	0.8082
2	32	4	12	1.2788e-06	2	0.9023
3	192	8	8	0.0001	5	0.9193
4	96	12	14	5.6398e-05	6	0.8335
5	224	16	14	4.0599e-06	6	0.8565
6	256	4	10	0.0006	4	0.8758
9	192	16	12	0.0001	5	0.8723
16	192	12	4	8.7679e-05	6	0.8659
30	224	8	6	4.5198e-06	6	0.8919
41	192	16	12	0.0001	6	0.8825
44	224	8	12	0.0001	6	0.9131
47	192	8	14	0.0001	6	0.9170
54	224	12	16	0.0002	6	0.9146
59	64	4	14	9.8440e-05	5	0.9209
62	64	8	14	0.0001	2	**0.9242**
67	96	12	16	0.0004	6	0.8965

Table 5: Hyperparameters autotuning with Optuna for the baseline R-GCN model

We tune the hyperparameters of the vanilla RGCN with Optuna Akiba et al. (2019). We run one study of 100 trials and we set the input channels, output channels, hidden channels, learning rate, number of bases as hyperparameters to tune. The values reported in Table are the non-pruned trials from Optuna. We find out that the best validation values are given by trial 62. The most important hyperparameters to tune to improve the validation accuracy are the output number of channels and the learning rate. The former defines the expressiveness of the output embedding, whereas, the latter shows how efficiently the network can reach a local/global optimum in the search space. The importance on the hyperparameters is shown in the bar chart D.8.

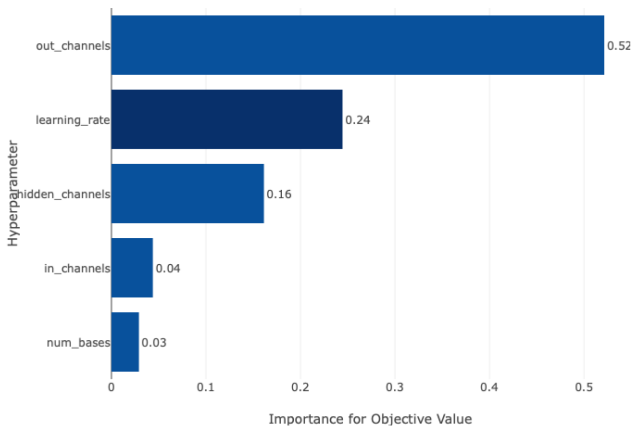
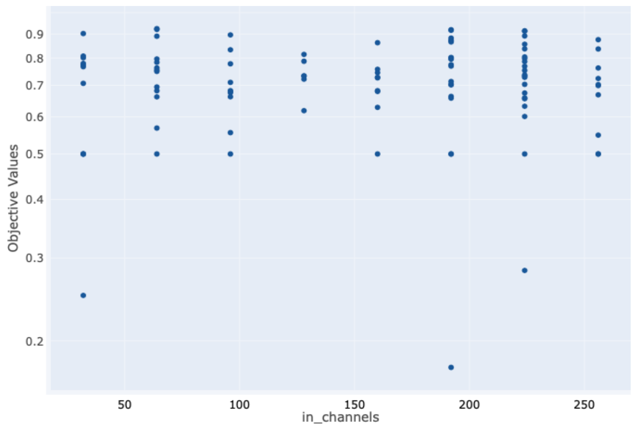
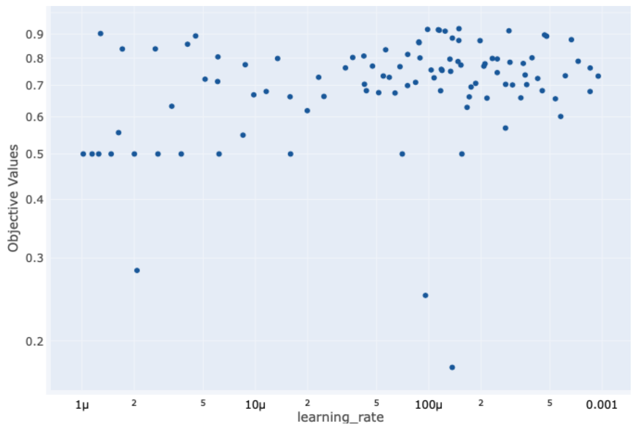


Figure D.8: Hyperparameters and their importance relative to the objective value

Figures D.9b and D.9a show the different parameter values for the learning rate and number of input channel respectively. As it is possible to see the best combination given from learning rate equal to 0.0001 and number of input channels equal to 64.



(a) Input number of channels against objective value



(b) Learning rate against objective value

Figure D.9: In (a)is shown the comparison between different number of input channel for the R-GCN against the objective value, whereas, in (b) is shown the comparison between learning for the R-GCN Adam optimizer against the objective value.

Further experimental details on Link Prediction

The link prediction task is performed with the main aim to show a contribution of the PE for the overall performance of the R-GCN. The following line plots show what explained in the results. Figure D.10 show the values for the training loss dna the validation for the run of one experiment for the vanilla R-GCN. Furthermore, Figure D.11 shows that the scaling factor, α, is positively correlated to the validation accuracy when the PE is used directly to the DistMult decoder without the use of the MLP. Additionally, we also observe that the validation accuracy for the eR-GCN without the MLP reaches a higher value compared to the one for the vanilla R-GCN and α equal to zero.

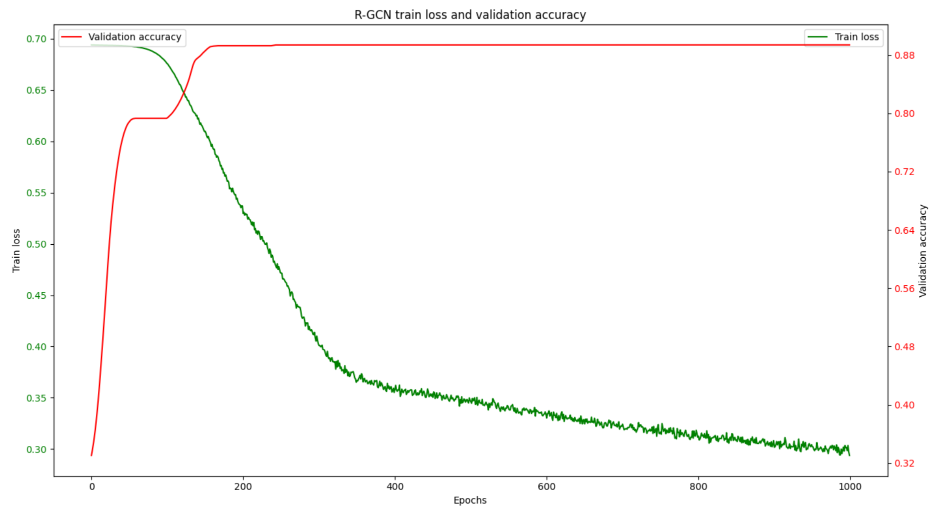


Figure D.10: R-GCN train loss and validation accuracy comparison

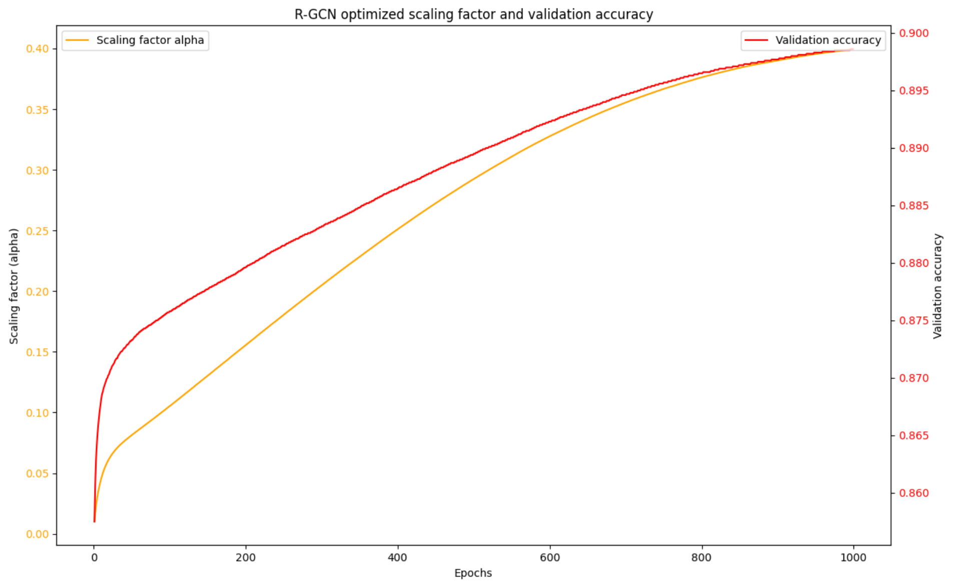


Figure D.11: Scaling factor against Validation accuracy for the eR-GCN without MLP