International Journal of Software Engineering and Knowledge Engineering

RR-GCN : EXPLORING UNTRAINED RANDOM EMBEDDINGS FOR RELATIONAL GRAPHS

SANDEEP RAMACHANDRA*
sandeep.ramachandra@ugent.be

VIC DEGRAEVE
GILLES VANDEWIELE
BRAM STEENWINCKEL
SOFIE VAN HOECKE
FEMKE ONGENAE
IDLab, Ghent University - imec,
9000 Ghent, Flanders, Belgium

Received (31 October 2024)
Revised (11 February 2025)
Accepted (16 March 2025)

The inception of the Relational Graph Convolutional Network (R-GCN) marked a milestone in the Semantic Web domain as a widely cited method that generalizes end-to-end hierarchical representation learning to Knowledge Graphs (KGs). R-GCNs generate representations for nodes of interest by repeatedly aggregating parameterized, relation-specific transformations of their neighbors. However, in this work it is posited that the R-GCN's main contribution lies in this "message passing" *paradigm*, rather than the learned weights. To prove this, the "Random Relational Graph Convolutional Network" (RR-GCN) is introduced, which leaves *all* parameters untrained and thus constructs node embeddings by aggregating *randomly* transformed *random* representations from neighbors. Additionally, the advantage offered by learnable parameters for RR-GCN without completely losing the advantages of random transformations is explored. It is empirically shown that RR-GCNs can compete with fully trained R-GCNs in node classification.

Keywords: Representational Learning; Knowledge graph embedding; Graph Convolutional Network; Message Passing

1. Introduction

Knowledge Graphs (KGs) are the ideal data structure to represent both expert knowledge and observational data. With the recent advances in Machine Learning (ML) methodologies [1, 2], KGs can be used to reveal new insights about the modeled domain. As these models typically operate on Euclidean data, the integration

*Corresponding author, OrcID: *https://orcid.org/0000-0001-5505-4362*

of KGs into their decision making processes involves a non-trivial transformation from information represented as a variable number of nodes and edges to fixed size numerical vectors, i.e., node embeddings. Graph Convolutional Networks (GCNs) embed the structural information of each node in a KG by iteratively recalculating node embeddings using aggregations of the transformed neighbor node representations. These transformed representations are referred to as "messages" and the collection and aggregation of these messages is called "message passing". An extension of the GCN, called the Relational GCN (R-GCN), similarly embeds the graph's structural information, but also includes the semantic information stored in the unique (named) edges. For the R-GCN, the messages are generated using learned relation-specific transformations [3]. The R-GCN is used as a foundation for all newer variations in relational GCN model architectures. The newer variants make incremental changes to R-GCN with the message passing aspect retained mostly the same with R-GCN serving as a baseline for comparison. R-GCN would use enormous memory for the trainable node embeddings and relational transforms which would not fit in memory of most single GPUs for large KGs, necessitating using CPU for training the model on multiple GPUs with model parallelism for training. This would take a long time for training with CPUs being slow without the parallelism offered by GPUs. Multiple GPUs would help speed up the training but for the really large KGs, message passing which is performed in one matrix multiplication of node embedding and relational transform matrix would not fit in memory, needing matrix multiplication to be modified to be read from disk in patches of matrices which would again slow the training. All of this complexity in case of GPU training and slow training in case of CPU training for training R-GCN is mostly for comparison purposes.

In this work, this paper is the first to evaluate *fully random* R-GCNs for KG embeddings and explores unsupervised data augmentation for the random embeddings to add additional information regarding local structure of KG. These "Random Relational Graph Convolutional Networks" (RR-GCNs) are initialized with random weights for each node type and random transformations for each relation in the KG. Fixed random weights are chosen because the chief benefit of R-GCN for learning the structural and semantic information of KGs is gained from the message passing. In message passing, the weights for each node type uniquely identifies each node in an abstract latent space and the relational transformations of these weights encodes information about the structure of the KG. In this paper, we hypothesise that the weights themselves do not need to be trained and can be random fixed weights. These random weights will still uniquely identify the node in a latent space and the transformations would still encode the information about the structure of the KG. Fixing the weights makes it computationally inexpensive while performing competitively with R-GCN. The lack of training for RR-GCN removes the backpropagation steps and the necessity to calculate and store gradients for each trainable parameter (which is a huge time gain in the case of R-GCNs) which would speed up the model considerably [4]. To put this intuitively, using RR-GCN would be running at

inference speeds even during training.

Further information about the neighborhood can be derived from unsupervised processing of the weights of neighbors. In the paper, we propose Proportion of Positive Values (PPV) [5, 6] as one such augmentation of information. PPV adds information about the neighborhood of each node based on the neighboring node's features.

Training the node and relation weights determine the important relations and the important nodes for the task. This identification would be missing in untrained RR-GCN. To address this shortcoming, we propose a low parameter training (node and relational attenuation) for the node and relation weights which can bridge the divide between R-GCN and RR-GCN. This method loses the time and memory benefits gained from the untrained RR-GCN but can overcome performance deficits for certain datasets.

These RR-GCN configurations are evaluated on 9 node classification datasets and compared against the default R-GCN. We show that fixed random transformations can capture a surprising amount of information. The results show that RR-GCNs produce embeddings on par with or better than those produced by a trained R-GCN. The effectiveness of these random transformations illustrates that, for KGs, the R-GCN's message passing and aggregation *paradigm* is more significant than the actual parameters, which have to be obtained through an expensive training procedure. This makes our RR-GCNs an interesting baseline model when developing trained embedding methods and opens up avenues for further research on more efficient, and more powerful, message passing parametrization for KGs.

The paper is structured as follows. Section 2 provides the background and related works. Section 3 discusses the technique of training and the various models and their variations being used in the work. Section 4 introduces the task and the datasets and reports the results of the trained models, the comparison against established state of the art results for each task, and in the following Section 5, the discussions of the implications of the result. Finally, Section 6 puts forth the closing remarks on the work.

2. Related works

In this section, some of the existing research related to the RR-GCNs will be covered. First part covers different kinds of KG representation learning models. Then the R-GCN as the point of reference for the architecture of the RR-GCN is discussed. The final section goes over the existing work that was done on using random representations within the context of machine learning.

2.1. *Representation Learning for KGs*

Techniques to embed substructures in KGs can be categorised into four groups. A first category compromises techniques that extract generic properties from neighborhoods of substructures of interest. These can either be feature-based [7, 8] or

exploit similarities with other substructures of interest, i.e., kernel functions [9, 10]. A second category consists of (algebraic) embedding spaces learned using tensor factorisation or through negative sampling [11, 12, 13, 14, 15]. A third category adapts existing natural language processing (NLP) techniques, such as Word2Vec [16], to graph structures [17]. A fourth and final category contains the message passing architectures that are trained end-to-end to aggregate relevant information around the substructures of interest [18, 19, 20, 21]. R-GCNs belong to this final category.

2.2. *R-GCN*

In general, Graph Neural Networks (GNNs) (of which GCNs and R-GCNs are particular subcategories) enable graph ML by learning how to update a node's representation based on its neighbors. These GNNs work analogously to classical neural networks: they start from initial node features which are passed through multiple GNN layers to refine and abstract the representations by mixing in information from one additional hop with every layer.

A single GNN layer operates in three steps: every node (1) generates a "message" based on its current representation and sends it along its outgoing edges, (2) aggregates incoming messages, and (3) updates its representation based on the aggregated messages and its own previous representation. The definitions of the message, aggregation and update functions differentiate several subtypes of GNNs. The most prominent of these subtypes, the Graph Convolutional Network (GCN) [22], uses a learned transformation matrix as its update function, aggregates by averaging messages (including the node's own message) and passes the aggregations through an activation function to yield new node representations (see Eq. (1)).

$$h_i^{(l+1)} = \phi\left(\sum_{j \in \mathcal{N}(i) \cup i} \frac{1}{|\mathcal{N}(i)| + 1} \; W^{(l)} h_j^{(l)}\right) \tag{1}$$

Here, ϕ denotes the activation function used, $h_i^{(l)}$ is the representation for node i at layer l, $\mathcal{N}(i)$ is the set of neighbours for node i, and $W^{(l)}$ is the learned transformation matrix in layer l. This formulation assumes unweighted edges.

As suggested in Section 1, R-GCNs [18] extend GCNs to support typed edges. Since KGs are multi-relational, this extension is particularly interesting for hybrid ML. R-GCNs are almost identical to standard GCNs but learn a different transformation matrix $\mathbf{W_r^{(l)}}$ for every relation $r \in \mathcal{R}$. A separate transformation $W_0^{(l)}$ takes care of self-loops (see Eq. (2)). Note that $\mathcal{R}$ contains two copies of every relation in the original KG, as R-GCNs also take outgoing (i.e. inverse) edges into account, with different learned transformations.

$$h_i^{(l+1)} = \phi\left(W_0^{(l)} h_i^{(l)} + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|} \; W_r^{(l)} h_j^{(l)}\right) \tag{2}$$

Weight sharing between relations using Basis function decomposition and Block diagonal decomposition has been proposed to reduce both overfitting and the number

of required parameters [18].

GCN and R-GCN tend to overfit to the training data due to the size of the models and adding additional layers smooths over the whole model. This makes the models shallow [23]. There have been many modifications to the base GCN model to improve their expressiveness in deeper neural networks [23, 24, 25].

Self attention [26] has been used to great effect in large language models. The application of self attention in the domain of graph neural networks has been similarly inspired. There have been variations on GCNs [27, 28, 29, 30, 31] and R-GCNs [32]. These have mixed bag of results and do not address the large memory costs of using graphs. Zhang et al. [33] deals with this and proposes multiple techniques to reduce the memory usage of both regular GCN and self attention based Graph Attention Networks.

Random fixed weights have been applied to graph convolutions without needing training [34]. This approach can provide better than or equal performance to GCN while taking a fraction of the training time. The paper covers the mathematical proof for stability of training using random weights in GCNs. This approach is similar to the random weights used in this paper but in the author's opinion, does not show the full capabilities gained by using random weights nor does the paper explore the semantic information available in knowledge graphs.

2.3. *Learning from random representations*

Time series classification papers on ROCKET [5, 6] have explored the increase in performance of models by applying random convolutions and aggregations to the raw inputs to extract features. These transformations are not learned and are quick to compute which aids in their scalability. The initial version of the algorithm relied on maxpooling and the Proportion of Positive Values (PPV) pooling, which captures the proportion of the input time series for which the output of the convolution is positive. The later versions rely solely on the PPV pooling and so the same can be adopted for the KG domain.

The idea of freezing, or not training, layers in neural networks is not novel either. Reservoir Computing [35] is a paradigm in recurrent neural networks where inputs are first passed through any black-box non-linear system called a "reservoir", which could be an untrained neural network. As such, only the final layer that maps the learned representations to the target output is trained. The paradigm of random modeling often introduces a trade-off between efficiency and effectiveness. As many random, albeit less effective, transformations can often be applied very efficiently, they allow to scale to higher numbers. An example of a technique where quantity matters over quality is ExtraTrees [36], where a large number of decision trees with random splits yield good results.

In a blog post detailing the inner workings of (non-relational) GCNs [37], Thomas Kipf hinted at the discriminative power of representations resulting from untrained transformations, but this hypothesis was never formally evaluated, let alone for

KGs. In a recent paper [38], random transformations were evaluated in the context of link prediction in a neuromorphic computing setting. In this research, the authors backpropagated through random weights to train initial node embeddings. Other research demonstrated that keeping the initial node features random and frozen, and only training the message passing transformations results in good performance for unirelational graphs [39, 40]. For KGs, it is even possible to deterministically generate meaningful node features, avoiding the need to learn initial embeddings in the first place [41].

3. Methodology

This section focuses on the modifications to the R-GCN algorithm for adding random transformations.

3.1. *Random R-GCN (RR-GCN)*

This subsection discusses in depth on the random transformations used in message passing of the RR-GCN and the variations that can be introduced such as PPV and attenuation and consolidates the approaches in an algorithm.

3.1.1. *Random Transformations*

The proposed algorithm closely resembles the message passing paradigm of R-GCN (see Eq. (2)). The most important difference is that all the transformation matrices are randomly initialized using the Glorot uniform initialization function [42] and are then fixed to freeze the network. Eq. (3) shows the equation for updating the weights of nodes in RR-GCN.

$$\begin{aligned} h_i^{(1)} &= \phi\left(N(S_0)\eta(s_i) + \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|}\ N(S_r)\eta(s_j) \right) \\ h_i^{(l+1)} &= \phi\left(N(S_0)h_i^{(l)} + \sum_{r\in\mathcal{R}} \sum_{j\in\mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|}\ N(S_r)h_j^{(l)} \right) \end{aligned} \tag{3}$$

Both N and η are Glorot and gaussian(normal) functions for relation and node weights which take in seeds $S_{(i)}$ and $s_{(i)}$ respectively. From comparing the equations in Eq. (3), h_i^0 is randomly initialized from a normal distribution and is then frozen for all subsequent calls to the model.

In the original paper on the R-GCN [18], the node features that are fed to the first R-GCN layer are one-hot encoded. Using one-hot node features effectively assigns a separate initial embedding matrix $W_r^{(0)}$ per relation type in the first message passing layer. This is indeed desirable for R-GCNs, since the output node representation dimensionality is usually very small (10 to 16 in the original paper [18]). Starting with small, randomly initialized, node representations limits the level of detail in the initial node characterizations and leads to worse performance for some datasets [43].

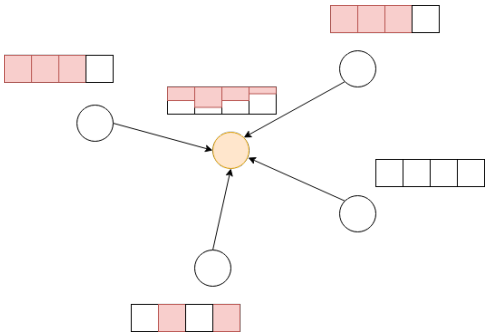

Fig. 1. The PPV is calculated by the proportion of positive values in the neighborhood of the central node. The red nodes depict negative values.

As RR-GCNs use much larger node representations, the first message passing layer takes random features $h_i^{(0)} \in \mathbb{R}^d$, with d the embedding size–which are not trained–and saves them as a single seed. In the experiments, this did not impact performance when compared to using one-hot encoded inputs.

3.1.2. *Proportion of Positive Values (PPV)*

The performance of ROCKET's random feature extraction for time series depends strongly on their use of PPV [5] pooling, as is also apparent from their second version which abandons max-pooling and uses *only* PPV features [6]. A graph-adapted variant of PPV that aims to encapsulate additional information about a node's neighborhood is proposed. Given a matrix of node representations H, PPV is defined as the proportion of strictly positive values in a 1-hop neighborhood per representation dimension (see Eq. (4) and Fig. 1).

$$P = \text{PPV}(H, \mathcal{N}) = \sum_{j \in \mathcal{N}(i)} \frac{1}{|\mathcal{N}(i)|} \mathbb{1}[h_j > 0] \tag{4}$$

Here, h_i is the representation for the i^{th} node (i^{th} row in H), $\mathcal{N}(i)$ is the set of neighbors for node i, and $\mathbb{1}[\cdot]$ is the indicator function (subset of elements are taken as one and rest of the elements as zero). The resulting matrix P, with the same dimensionality as H, houses additional node features that capture information about the feature diversity in every node's neighborhood. When the PPV hyperparameter is enabled, the PPV is calculated and concatenated to the embedding of layer H to produce the layer's output. In the first layer, the PPV uses directly the first layer's embedding as input. For each subsequent layer, the PPV from the previous layer is processed using RR-GCN to compute the new PPV.

The PPV function does not differentiate neighbors based on relation types as it is applied as a "post-processing" step to representations that are the result of relation-specific transformations.

3.1.3. *Attenuation*

The experiments with the fixed weights and transformations show that a large part of the GCN's performance can be traced to the message passing paradigm [34]. Similar performance can be expected for R-GCNs. However, the representations for each node H do not move much in the embedding space for different tasks since no training is involved. R-GCNs, which have learned embeddings for each task, do not scale well with memory or training time as will be discussed in section **??**.

A trade off between performance and scalability can be made by attenuating the fixed weights using a trainable parameter. The parameter can be used to attenuate both fixed node representations H, either as a single parameter for all nodes or per node parameter, and fixed relation-specific transformations W_r, as seen in Table 1.

Attenuation hyperparameter	Equation	Comment
Relation attenuation	$W_r^* = \beta_r.W_r$	β_r is the trainable relation attenuation parameter per relation and scales the relational transformation matrix
Single node attenuation	$H^* = \alpha.H$	α is a one trainable parameter node attenuation for all nodes in the KG.
Per node attenuation	$h_i^* = \alpha_i.h_i$	α_i is a trainable parameter for node i, for every node in the KG

Table 1. Equations describing the effects of the different attenuation hyperparameters. H and W_r are the node embeddings and relation transforms from the random seeded generators before message passing

H^* contains the new node representations used in the message passing procedure, W_r^* contains the new transformations for each relation. The hyperparameters choices for attenuation are grouped as follows: relational attenuation or no relation attenuation and, single node attenuation, per node attenuation or no node attenuation. The relational attenuation and node attenuation can be applied independent of the choice of the other. The attenuation works as a scaling parameter for the fixed weights and transformations which would allow for a better output by increasing/decreasing the weight of the relevant/irrelevant nodes and relations respectively. This approach trades the training speed and some of the memory improvements of RR-GCN with performance since the attained node representations are a better reflection of the task the model is being trained for.

Relation attenuation uses a single learnable parameter per relation to scale the random transformation matrix (as opposed to the R-GCN which has trainable trans-

formation matrices). Node attenuation can be implemented as a single parameter for all nodes, or with single parameter per node in the KG. This will still have a smaller memory footprint than the R-GCN since there is only one parameter per node whereas an R-GCN would have learnable parameters of embedding dimension size for each node.

These attenuations perform the selection of important KG features (either important relations or important nodes for relation attenuation and node attenuation respectively) similar in function to learnable parameters in R-GCN. These parameters give higher absolute importance to significant entities (node or relation) and closer to zero for less significant entities. This should improve the performance but will need end-to-end training similar to R-GCN.

3.1.4. *Algorithm*

Given a KG $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with entities $v_i \in \mathcal{V}$, edges $(v_i, r, v_j) \in \mathcal{E}$ and relation types $r \in \mathcal{R}$, each entity is assigned a Euclidean vector. These vectors should encapsulate each entity's semantics, as contained in the triples of the KG, such that they can be used as features for machine learning models. Note that entities can also be "literals" and have associated values. Both R-GCNs and the proposed RR-GCNs, however, treat literals as ordinary nodes.

Algorithm 1 Generating node embeddings $\in \mathbb{R}^{|\mathcal{V}| \times e}$ using RR-GCN layers with embedding size e, seed s, and number of layers n

procedure EMBED-RRGCN-PPV(e, s, n)
  $H \leftarrow$ NORMAL(shape $= |\mathcal{V}| \times e, \sigma^2 = \frac{1}{e}$, seed = s) ▷ Random feature matrix
  $S \leftarrow$ RANDOM(shape $= |\mathcal{R}| + 1$, seed $= s$) ▷ Random relation seeds
  $H \leftarrow max$(RRGCN-CONV(S, e, H), 0) ▷ ReLU of RRGCN-CONV
  $P \leftarrow$ PPV($H, \mathcal{N}$) ▷ See Eq. (4)
  for all 2 *to* n *layers* **do**
    $H \leftarrow max$(RRGCN-CONV(S, e, H), 0) ▷ ReLU of RRGCN-CONV
    $P \leftarrow$ RRGCN-CONV(S, e, P) ▷ No ReLU for PPV
    $P \leftarrow$ PPV($P, \mathcal{N}$)
  end for
  return $H||P$ ▷ Concatenation of node and ppv features
end procedure

Algorithm 1 and Fig. 2 illustrates the process of generating these embeddings in pseudocode. For ease of notation, it is assumed that triples are accessible through $\mathcal{N}_r(i)$, which contains the neighbors connected to an entity v_i through a relation $r \in \mathcal{R}$. It is also assumed that the relations in $\mathcal{R}$ are encoded as integers starting at one. Apart from the KG's characterization, the algorithm takes an embedding size e, a seed s, and a number of layers n as input. Based on the seed s, random

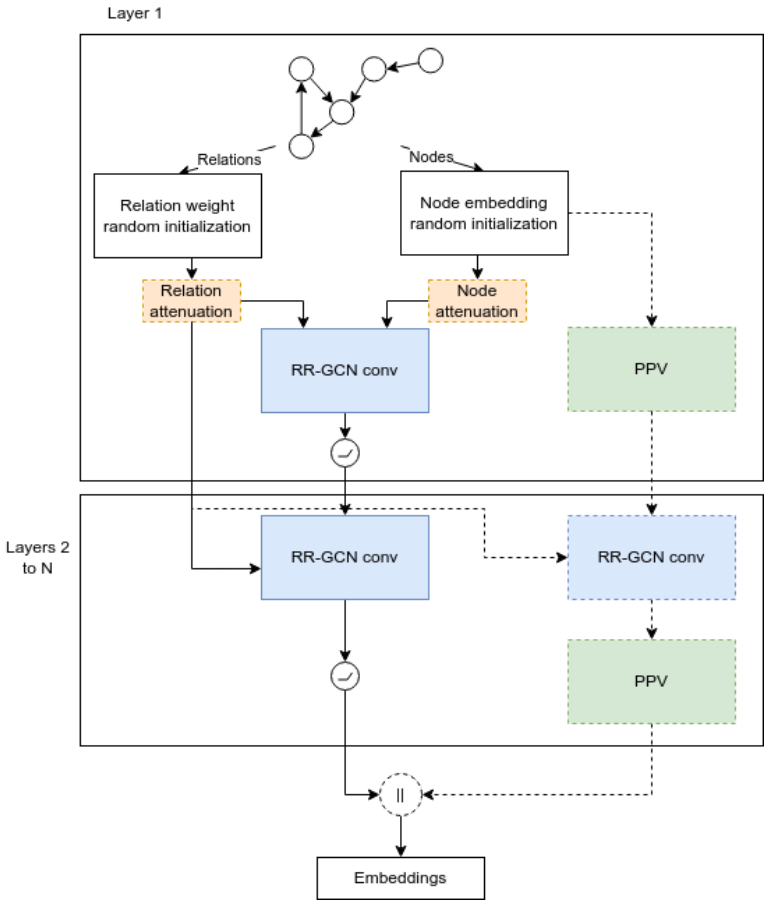

Fig. 2. RR-GCN - model architecture. The dotted components are optional and enabled by PPV and attenuation hyperparameters. The small operation after RR-GCN conv is a ReLU activation function. The last operation before the embeddings is a concatenation operation.

initial embeddings are sampled from a normal distribution with variance chosen such that the sum of a node's embedding is a standard normal random variable. These embeddings are attenuated by a node attenuation parameter α, either as one parameter for all the embeddings or as a per-entity parameter which scales with the number of entities. This initialization strategy is chosen over Glorot as the range of Glorot-initialized matrices gets smaller with both the number of rows and columns. This makes sense for transformation matrices, but not for embeddings. The seed s is then reused to generate a list of additional random seeds, one for every transformation matrix W_i. This list of seeds is given as an argument to the RRGCN-CONV function, listed in Algorithm 2.

The RRGCN-CONV function implements the R-GCN's message passing equation with optimized memory usage. Relation-specific (and self-loop) contributions are

Algorithm 2 RR-GCN Message Passing

procedure RRGCN-CONV(S, e, H)
  $X \leftarrow O_{e\times e}$ ▷ Zero initialized matrix
  for all $r \in \mathcal{R}$ **do**
    $W_r \leftarrow$ RANDOM-UNIFORM($e \times e$, S_r) ▷ Relation random weight matrix
    $x_i \leftarrow h_i + \sum_{j\in\mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|} W_r h_j^{(l)}$
  end for
  $W_0 \leftarrow$ GLOROT-UNIFORM($e \times e$, S_0)
  $x_i \leftarrow h_i + W_0 h_i^{(l)}$
  return X
end procedure

iteratively added to the zero-initialized ($O_{e\times e}$) output representation matrix X, generating the required transformations on-the-fly and attenuating them using the relation attenuation parameter from the given seeds. Note that x_i and h_i stand for the i^{th} row of X and H respectively, and that the computations for these rows are done all at once using efficient matrix multiplications. The row-wise notation is only used for clarity.

After the first message passing round, the obtained representations are passed to a ReLU non-linearity and stored in H. These hidden representations are then used to calculate the one-hop PPV features P as described in Section 3.1.2. The ReLU activation function is not used for PPV representations as they are positive by definition and the result of a non-linear operation.

Using the updated hidden node features, the previous steps are repeated $n-1$ times. There are two noteworthy details here: (i) the same seeds are used for every convolution, and (ii) the PPV features are convoluted independently from the regular representations (but with the same seeds). In our experiments, it was noticed that using different transforms in every layer was not necessary and sometimes even hurt performance for RR-GCNs.

After the random message passing layers, the concatenation ($\|$) of H and P along the horizontal axis yields the final node embeddings.

4. Results

In this section, the datasets and their statistics being used in the experiments are reported, followed by the results of the model and the baselines on the datasets. The empirical results show that RR-GCN method matches – and in some cases exceeds – the performance of end-to-end trained networks for the node classification task.

The RR-GCN layer and embedder are implemented in PyG [44], an extension of the popular deep learning framework PyTorch [45] that facilitates the implementation of message passing networks. PyG provides parallel execution of node representation updates.

4.1. *Task and Datasets*

This subsection aims to introduce the datasets being utilized for this work and reports the statistics of the datasets for ease of comparison.

The datasets used for node classification contain N nodes which are assigned to one of various different classes. For generalization purposes, each set of N nodes is subdivided into two sets of nodes $\mathcal{N}_{train}$ and $\mathcal{N}_{test}$ which are used for training and validating the model respectively. The goal of the task is to train a model M such that a loss function $\mathcal{L}(.)$ is minimized based on the neighborhood of each node $n \in \mathcal{N}$ and the edges $\mathcal{E}$ connecting them.

4.1.1. *Datasets*

Statistic	**AIFB**	**MUTAG**	**BGS**	**AM**
Entities	8,285	23,644	333,845	1,666,764
Relations	45	23	103	133
Edges	29,043	74,227	916,199	5,988,321
Train Entities	141	272	117	802
Val. Entities	0	0	0	0
Test Entities	37	68	29	198
Classes	4	2	2	11
Mean Degree	7.82	6.27	11.89	13.74
Max. Degree	1,281	6,783	83,024	73,447

Table 2. Dataset statistics for the KGs from Ristoski et al.

The terminology used to describe the datasets are explained below:

- *Entities* - The number of nodes present in the KG.
- *Relations* - The unique types of relation between nodes.
- *Edges* - The number of relations present in the KG.
- *Degree* - The number of relations each node has with its neighbors (both incoming and outgoing).
- *Classes* - The number of types of nodes to classify.

First, benchmark KGs of varying sizes, provided by Ristoski et al. [46]-AIFB, MUTAG, BGS, and AM, are considered for node classification. The descriptions of the datasets are given below [47]:

(1) AM: contains semantic information about artefacts in the Amsterdam Museum. Each artefact in the dataset is linked to other artefacts and details about its production, material, and content. In this task, 1000 samples are available for an 11-class classification problem where the goal is to predict to which category

an artefact belongs (347, 127, 116, 86, 81, 56, 55, 50, 42, 25, 15 samples for each class respectively).

(2) BGS: describes three geological measurements in Great Britain. The task of this dataset is to predict the lithogenesis property of named rock units. The dataset contains 146 named rock units with a lithogenesis, from which we use the two largest classes.

(3) AIFB: describes the AIFB research institute in terms of its staff, research group, and publications. The dataset was first used to predict the affiliation (i.e., research group) for people in the dataset. The dataset contains 178 members of a research group, however the smallest group contains only 4 people, which is removed from the dataset, leaving 4 classes.

(4) MUTAG: contains about 340 molecules classified as "mutagenic" (129) or "not mutagenic" (211). The dataset includes 75k triples with 24 unique relation types and 250 predicate-object pairs per node, used for binary classification of mutagenicity.

The salient properties of these datasets are listed in Table 2.

Additionally, larger multimodal KG datasets from "kgbench" [48] are also being compared-AMPLUS, DMG777K, DMGFULL, MDGENRE, and DBLP. These datasets contain larger training, testing, and validation sets. The datasets are described below [48]:

(1) AMPLUS: extends the AM dataset without the 1000 sample restriction and maps the nodes to smaller classes to ensure balanced class distributions (25782, 22048, 7558, 5455, 4333, 4012, 2672, 1563 samples for 8 classes).

(2) DMG777K: the Dutch Monument Graph (DMG) is a dataset from the Digital Humanities. Encompassing knowledge from several organizations, the DMG contains information about 63 566 registered monuments in the Netherlands. Engineered with the goal of creating a highly multimodal dataset, the DMG contains information in six modalities, five of which are encoded as literals. The 777k -variant is a subset encompassing 8 399 monuments created by sampling monuments from the top-5 monument classes that have no missing values.

(3) DMGFULL: The full DMG dataset is included with the missing values with 63,566 monuments included in the dataset.

(4) MDGENRE: The Movie datasets are subsets of Wikidata in the movie domain. The movies which are nominated or won an awards are selected and the people associated with the movies by a whitelisted relation are added to the dataset. The thumbnails of the movies are added to the dataset. There are 8863 movies to be labeled. This creates a knowledge graph that is centered around movie-related data and has a longest path of 4, making the knowledge graph relatively simple. The main objective of this dataset is to predict the dominant genre of the movies.

(5) DBLP: The DBLP repository is a large bibliographic database of publications

in the domain of computer science. The dataset contains 86535 DOIs filtered from 2 databases which are the number of samples used. The goal is to predict the citation count into two classes: those papers which received one citation, and those which received more.

Table 3 lists the properties of the kgbench datasets.

Statistic	AMPLUS	DMG777K	DMGFULL	MDGENRE	DBLP
Entities	1,153,679	341,270	842,550	349,344	4,470,778
Relations	33	60	62	154	68
Edges	2,521,046	777,124	1,850,451	1,252,247	21,985,048
Train Entities	13,423	5,394	23,566	3,846	26,535
Val. Entities	20,000	1,001	10,001	1,006	10,000
Test Entities	20,000	2,001	20,001	3,005	20,000
Classes	8	5	14	12	2
Mean Degree	4.37	4.53	4.47	7.17	9.83
Max. Degree	154,828	65,576	121,217	57,363	3,364,084

Table 3. Dataset statistics for the larger-scale "kgbench" KGs

4.2. *Results*

A first step in the evaluation procedure is to reduce the size of the KG by excluding vertices that are further than n hops away from any of the training ($\mathcal{V}_{tr}$) or testing ($\mathcal{V}_{te}$) vertices, as no information from more than n hops away can be propagated to the nodes of interest with n message passing layers. Once the size of the KG is reduced, they are passed through n layers of our RR-GCN to create embeddings of size e. The constants n and e are tuneable hyper-parameters. Once the embeddings are generated, they are provided as input to a gradient boosting classifier. In this section, CatBoost [49] is used for the non trainable RR-GCN and the results are reported without attenuation enabled. Multi Layer Perceptron (MLP) could also be used for the classifier but then would need a backward pass to train the model which is a lot slower than a simple Catboost. We used Catboost since it would allow for better evaluation of the embedding from the RR-GCN layer by using the embedding directly whereas the MLP would be an affine transformation of the embedding. Note that when a backward pass is necessary (RR-GCN with attenuation), we will use an MLP layer to train the model end to end. The No NA, No RA results of 8 provides the benefit attained from using MLPs.

For the small-scale benchmark KGs, the hyper-parameters n and e are tuned using a grid search with stratified five-fold cross-validation, with a different RR-GCN seed in every fold. For the larger-scale KGs, RR-GCN models were evaluated five times with different seeds on the provided validation set to tune

	number of layers		embedding size	
Model	NO PPV	PPV	NO PPV	PPV
AIFB	4	1	256	512
AM	5	5	768	768
BGS	5	5	512	512
MUTAG	2	2	1024	1024
AMPLUS	5	5	1024	1024
DBLP	5	5	256	256
DMG777K	2	2	1024	1024
DMGFULL	2	1	256	1024
MDGENRE	5	5	768	1024

Table 4. Optimal hyperparameters per dataset, for RR-GCNs that use PPV features and RR-GCNs that leave them out

the hyperparameters. The hyperparameter space for n is $\{1, 2, 3, 4, 5\}$ and e is {256, 384, 512, 768, 1024} and optimal values are chosen based on log-loss, ignoring configurations that resulted in more than 24GB of GPU memory. An important hyperparameter for CatBoost is the number of boosting iterations. During validation, this is determined using "early stopping". For evaluation runs on a given dataset's test set, the models are run for the maximum number of iterations required for that dataset's validation runs with optimal hyperparameters.

The R-GCN results from Schlichtkrull et al. [18] were reproduced using an external implementation [43], as the original code uses deprecated libraries. For the small-scale KGs, the hyper-parameter configuration reported in their study [18] were used along with early stopping with 10 epochs patience, and a validation set held out from the training data to determine the optimal number of epochs. For the larger-scale KGs, the hyper-parameters as reported in the "kgbench" paper [48] were utilized. All "kgbench" measurements were performed on CPU, to accommodate the high memory requirements for backpropagation. Only for the DBLP dataset, for which no results were reported in the original paper due to high memory requirements, an embedding size of 10 (as opposed to 16) with 40 base functions was used to make it fit in 64GB of CPU RAM.

The R-GCN and RR-GCN setups were repeated ten times with different seeds and the corresponding test accuracies recorded. The results for both the RR-GCN as described in Algorithm 1 (RR-GCN-PPV) and a version that does not include the PPV features (RR-GCN) are reported. The mean accuracy results, and their corresponding standard errors are provided in Table 5.

4.2.1. *Performance ablation*

The three datasets which have significantly worse performance in our RR-GCN-PPV - AIFB, BGS, AM are selected for performance ablation. All 3 are noticed to have

Dataset	r-gcn	rr-gcn	rr-gcn-ppv
AIFB	**96.11** ± 0.45	83.33 ± 1.37	86.11 ± 0.93
AM	**88.99** ± 0.39	81.67 ± 0.57	84.65 ± 0.62
BGS	**86.21** ± 0.89	80.00 ± 2.34	78.97 ± 2.44
MUTAG	72.50 ± 0.91	70.00 ± 0.83	**79.41** ± 0.58
AMPLUS	83.81 ± 0.13	76.85 ± 0.06	**84.54** ± 0.08
DBLP	68.51 ± 0.99	70.18 ± 0.11	**70.61** ± 0.07
DMG777K	62.51 ± 0.38	61.40 ± 0.32	**63.97** ± 0.26
DMGFULL	57.52 ± 0.19	60.50 ± 0.26	**63.38** ± 0.17
MDGENRE	**67.33** ± 0.19	65.09 ± 0.10	67.15 ± 0.08

Table 5. The average accuracy and standard error of 10 measurements.

a large number of low-degree nodes such as literals which have only one neighbor and so, a degree of 1. PPV relies on a larger degree neighborhood to generate a good augmentation. So, it is possible that these low-degree nodes add noise to our representations, which can overpower the useful signal in some datasets. To test this hypothesis, R-GCN and RR-GCN-PPV are trained with filtered versions of AIFB, BGS and AM and the same hyperparameters as before. In these filtered KGs, low-degree nodes (degree ≤ 5) are removed.

Model	AIFB	BGS	AM
R-GCN	**96.11 ± 0.45**	**86.21 ± 0.89**	**88.99 ± 0.39**
RR-GCN-PPV	86.11 ± 0.93	78.97 ± 2.44	84.65 ± 0.62
R-GCN-CUT	95.56 ± 0.45	**86.21 ± 0.89**	**88.13 ± 0.41**
RR-GCN-PPV-CUT	**95.83 ± 0.62**	84.14 ± 1.38	84.80 ± 0.23

Table 6. The average accuracies and standard error of 10 measurements with degree cutting.

After the low degree node cutting, R-GCN performance in Table 6 is barely affected in AIFB and AM and exactly the same in BGS. RR-GCN-PPV, however, demonstrates almost 11% and 6% improvement in scores, with the scores for AM dataset remaining stagnant.

4.2.2. *Attenuation*

While the non-attenuated RR-GCN could avoid the intermediate gradients, the attenuation made it so that models will need more GPU memory to store the gradients. The hyperparameters n and e are still used for this method. Hence, a chief difference here is the use of a linear layer rather than a boosting classifier to allow for end-to-end loss back propagation. While a linear layer could be used with the

non-attenuated RR-GCN, it would require multiple passes over the model and large intermediate activations which would have increased memory usage. This means that the results from Table 5 are not directly comparable with the results from this section. Another difference is the use of only the small KGs for evaluation since the models consume too much memory for the bigger "kgbench" datasets (approaching 100 GB peak memory for DBLP). The hyperparameters are chosen for lowest log losses reported from the models, provided they could fit in 48GB (the GPU memory available on Nvidia A40 graphics card). The models are run for 3000 epochs with an ADAM optimizer [50] along with an early stopping mechanism. Additionally, validation is performed every 50 epochs as a means of speeding up the training of the models. Table 7 lists the hyperparameters used in the attenuation models for both ppv and non-ppv models.

Model	**number of layers**	**embedding size**
AIFB	4	256
AM	2	256
BGS	5	256
MUTAG	5	512

Table 7. Optimal hyperparameters per dataset, for RR-GCNs with attenuation enabled

Model		**AIFB**		**MUTAG**		**BGS**		**AM**	
		ppv	no ppv	ppv	no ppv	ppv	no ppv	ppv	no ppv
RA	No NA	91.67	88.89	79.41	76.47	**89.66**	86.21	**86.87**	84.34
	1-NA	91.67	88.89	77.94	75.00	86.21	86.21	86.32	81.82
	PNA	**94.44**	**94.44**	80.88	66.18	**89.66**	82.76	86.36	85.35
No RA	No NA	*83.33*	88.89	82.35	72.06	*82.76*	86.21	77.27	73.74
	1-NA	91.66	88.89	**83.82**	73.53	86.21	86.21	82.32	80.81
	PNA	*91.67*	**94.44**	80.88	72.06	**89.66**	86.21	**86.87**	83.84

Table 8. The accuracies of ablated RR-GCN models. RA stands for relation attenuation, NA for Node Attenuation, 1-NA for single node attenuation, and PNA for per node attenuation. The highest accuracies for each dataset is highlighted in bold. Notable results are italicized. The base RR-GCN results are underlined for comparison.

The results in Table 8 are reported with three main ablation parameters - PPV (present, absent), node attenuation (no node attenuation, single node attenuation, per node attenuation), and relational attenuation (present, absent) - with 12 possible combinations. Since these models take quite a long time to train, the models

are trained only once.

4.3. *Comparison against other Graph embedding baselines*

R-GCN is the foundation for nearly all relational graph convolutional networks. The previous experiments used R-GCN as the baseline since the RR-GCN is mostly similar to R-GCN's message passing. In this section, we compare against newer variants of relational graph networks and some non relational graph convolutional networks. We have selected the models with a variation on the message passing paradigm. The baseline models being compared are as follows:

- GCN: Graph Convolutional Network (GCN) [22] is a semi-supervised approach to handling homogeneous graphs (graphs with only one edge type). The different relational information in the datasets is lost when using GCNs.
- GAT: Graph Attention Network (GAT) [27] is another semi-supervised approach to handle homogeneous graphs with the message passing adapting self attention approach to graph structures.
- R-GCN: Relational Graph Convolutional Network (R-GCN) [18] was motivated by GCN model yet introduces relation-specific transformations by setting up an independent adjacency matrix for each relation.
- CompGCN: CompGCN [21] jointly embeds both nodes and relations in a relational graph with entity-relation composition operations.
- RHGNN: RHGNN [51] is a general heterogeneous graph neural network, which employs hidden correlations between different types in heterogeneous graphs via Type2vec and reasonably combines edge and node propagation layers together to enhance the learning of both nodes and edges representation. For comparison, the model without the Type2vec pretraining is also included.
- r-GAT: r-GAT [52] is a relational graph attention network to learn multi-channel entity representations. Specifically, each channel corresponds to a latent semantic aspect of an entity. The model includes a query-aware attention layer on these channel representations to select useful aspects of the channels for subsequent tasks.
- RR-GCN: Our RR-GCN uses random transforms on random node features to generate untrained unsupervised representation of the graph. RR-GCN-PPV captures additional data from node neighborhood via the proportion of positive values around each node. Another variant is RR-GCN-PPV-ATTENUATION which uses tuned attenuation hyperparameters for select trainable parameters to bridge the performance gap between RR-GCN and R-GCN.

The models are compared on the 4 smaller graphs - AIFB, MUTAG, BGS, AM - used in all previous results since they are more widely used benchmark datasets available for all the models. The comparison between the models is available in Table 9. Note the r-GAT model results for AM dataset is not available and the code for the model is not made public.

Model	AIFB	MUTAG	BGS	AM
RGCN	95.83±0.62	79.85±3.86	83.10±0.80	86.72±2.38
RHGNN -pretraining	94.17±2.05	77.06±2.52	89.66±0.00	86.58±0.00
RHGNN	97.22±0.00	82.45±1.72	93.20±0.25	90.40±0.61
CompGCN	89.17±2.76	74.56±1.40	70.69±4.38	90.06±0.20
GAT	94.72±3.06	70.44±1.46	84.48±2.44	67.30±2.31
GCN	91.39±3.81	75.15±1.62	80.34±3.65	69.90±1.91
r-GAT	97.22	88.24	89.66	Unavailable
RR-GCN	83.33±1.37	70.00±0.83	80.00±2.53	81.67±0.57
RR-GCN-PPV	86.11±0.93	79.41±0.58	78.97±2.44	84.65±0.62
RR-GCN-PPV-ATTENUATION	94.44	83.88	89.66	86.87

Table 9. Performance comparison of different models across various datasets. The results for the baselines are taken from Zhu et. al. [51] and Chen et. al. [52]. RHGNN -pretraining refers to the RHGNN model without Type2Vec pretraining for edge features.

5. Discussion

This section discusses the accuracy results of the experiments. A commentary on the memory usage of trained and random R-GCN is also provided.

5.1. *Performance of base* rr-gcn *and* ppv

Table 5 shows that, for most small-scale datasets, R-GCNs score better. However, the differences are not that drastic considering one method is trained end-to-end and another is unsupervised and based on *random* transformations. The fact that RR-GCN comes so close to R-GCN without any training follow our hypothesis that the message passing powers most of the performance of R-GCN. In DBLP and DMG-FULL, base RR-GCN model is able to surpass R-GCN. This is likely due to the fact that the hyperparameters are not fully tuned for the larger kgbench datasets.

PPV features, which measure the "representation diversity" around every node, seem to have a positive impact for most datasets. With PPV, RR-GCNs even score significantly better for the MUTAG dataset. Only for BGS, PPV seemingly deteriorates performance. However, the difference between RR-GCN-PPV and RR-GCN is statistically insignificant (Mann-Whitney test with cutoff 0.05).

For the larger datasets, the average performance of R-GCNs and RR-GCNs is even more similar, with most datasets statistically significantly favoring RR-GCN-PPV over a trained model. The difference for MDGENRE, the only "kgbench" dataset where R-GCNs score better on average, is statistically insignificant.

5.1.1. *Performance ablation*

As shown in Table 6, removing low-degree nodes seems to close the gap in average performance for most datasets: the performance for R-GCN stays roughly the same, while RR-GCN results improve.This supports our hypothesis that R-GCN was

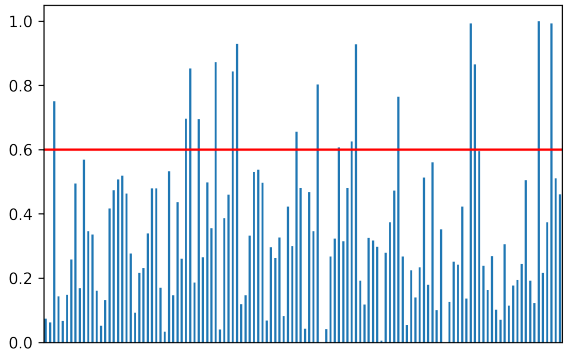

Fig. 3. Mean absolute weights for per-relation transformations learned by a two-layer R-GCN for the AM dataset, relative to the maximum per-relation mean absolute weight.

learning to ignore low degree nodes and that they were harming the RR-GCN-PPV performance by adding noise to labeled nodes. The performance differences to the trained counterparts for AIFB and BGS even become statistically insignificant. For AM, however, low-degree node removal seems to have little to no effect. For this dataset, the R-GCN likely learns to ignore a subset of relation types. To check this, the mean absolute values of a trained R-GCN's *learned* per-relation transformation matrices (true and added inverse relation types are averaged), averaged over all layers is calculated.

Indeed, Fig. 3 illustrates that there are quite a few relation types that a trained R-GCN learns to attenuate. If this "relation importance" information is taken into account by removing (in addition to low-degree nodes) all relation types that have less than 60% of the maximum "importance", RR-GCN obtains an accuracy of 91.31 $\pm$ 0.24%, which is even statistically significantly better than trained R-GCNs.

The chief benefit of RR-GCN comes as a quick baseline for graph data. By not requiring training to generate an embedding of the graph, without having random walks through the graphs, RR-GCN provides near R-GCN level results for node classification problems. There is a clear time benefit especially since the RR-GCN model is untrained and uses CatBoost model whereas the R-GCN model uses a MLP layer and requires both gradient calculation and backpropagation (see Section 1). This means the results for RR-GCN are available in the order of minutes and R-GCN in hours/days. This provides huge savings in GPU compute times (along with the attached environmental benefits).

5.2. *Attenuation*

The results in Table 8, in general, show that with fewer training parameters, RR-GCN with attenuation comes very close to the accuracies of R-GCN for AIFB (94.44 vs 96.11) and AM (86.87 vs 88.99) and even exceeds them for MUTAG (83.82 vs 72.5) and BGS (89.66 vs 86.21). Some part of this improvement can be attributed

to the addition of the linear layer but the addition of the attenuation improves the performance significantly more. The general trend for PPV is that it improves the performance of the RR-GCN, with some notable exceptions which are italicized in the Table 8. This may be an issue with bad initialization of parameters. PPV, as seen in Tables 5 and 8 demonstrate that additional statistical information for each feature in the neighborhood of the node provides more separable embeddings for the classifier layer which leads to better results in nearly all cases.

There is no clear winner between node attenuation and relation attenuation. This can be due to the different characteristics of each dataset. AIFB and BGS seem to perform best with a per-node attenuation and AM works better with relation-based attenuation. The per node attenuation performs better with low train entity datasets and relational attenuation is preferred for datasets with a large number of relation types. This follows some intuition that more parameters equals more performance. For the smaller datasets, using per node attenuation would benefit the most of the options available and for BGS and AM with larger relation types prefer relation attenuation. MUTAG with low entities and relations bucks the trend of more parameters equals more performance, instead preferring single node attenuation and PPV. The preference for per node attenuation can also be attributed to the task being a node classification task. The performance varies between different hyperparameters for each dataset and can be used as a hyperparameter during hyperparameter search.

5.3. *Comparison against other baselines*

The models for homogeneous graph convolution - GCN, GAT - ignore relational information and so unsurprisingly perform well in dataset with low relation types. However, the GAT model for BGS performs slightly better than R-GCN which can be attributed to its attention like message passing. Our RR-GCN-PPV-ATTENUATION model however does consistently outperform the homogeneous graph models which is attributable to the relational information in the graphs since R-GCN also beats these models.

Sections 5.1 and 5.2 already discuss the performance of R-GCN against our model. CompGCN performs better than RR-GCN variants in only AM dataset which might be that the joint embedding of relation and nodes performs better in very large graphs due to needing more training entities (see Table 2). r-GAT performs similar to RHGNN in performance in all datasets available except MUTAG. This could be a benefit of the self attention performed after the channel entity representation to disentangle node and edge features. But this model with its self attention layer probably would not scale well since the original paper avoided using AM dataset, favoring only the easier and smaller datasets. However, RR-GCN-PPV-ATTENUATION still performs favourably against r-GAT. Of the most interest in this comparison is RHGNN models. Without the pretraining of edge features using Type2Vec, the model underperforms RR-GCN-PPV-ATTENUATION but with the pretraining, beats

our model in all datasets. This suggests that the RR-GCN performance can be greatly enhanced by using some pretrained node features and relation transforms instead of random assignment. We will leave this as an avenue to improve RR-GCN performance.

5.4. *Memory usage*

This subsection covers one of the main reasons for favoring RR-GCN over regular R-GCN which is the *huge* memory required for R-GCNs.

Trained R-GCNs need to store their parameters in (GPU) memory. Since relation-specific transformations are typically quite small, this is dominated by the initial node representations. Especially when using one-hot encoding as initial representations, this memory load can become quite significant. Additionally, in the backward pass, gradients for all weights (and for optimizers such as Adam [53], even gradient moments) have to be stored as well, which at least doubles the parameter memory requirements. For a KG with nodes $\mathcal{V}$, an R-GCN with B bases (obtained using Basis function decomposition [18]) and an initial representation size e thus needs to be able to keep *at least* two $B \times |\mathcal{V}| \times e$-dimensional float-tensors (4 bytes per element) in memory. For DBLP (a dataset with 4,470,778 nodes, Table 3) and a 40-base R-GCN with a 16-dimensional embedding size this results in at least 22.89GB of memory.

An even bigger source of memory usage for trained R-GCNs, is the storage of intermediate activations. Even if per-relation contributions are iteratively summed to an accumulator as in Algorithm 2, by default, all individual contributions across *all layers* are still kept in memory during training, as they are required for back propagation. This results in a $|\mathcal{V}| \times e_l$-dimensional float-tensor for *every relation* $r \in \mathcal{R}$ and the layer's global output, for every layer l, with e_l the output dimensionality for layer l. For DBLP (136 relations with inverses) and a single R-GCN layer with a 16-dimensional output, e.g., this results in 39.20GB. Activation checkpointing [54] could be used to avoid storing intermediate activations by recalculating them during the backward pass, trading in memory for compute but it is still a significant memory cost.

Since RR-GCNs do not need a backward pass, there is no need to keep any gradients or intermediate activations in memory. Moreover, because the initial node embeddings and transformation matrices are random, they need not be kept in memory. Storing the seeds is sufficient to generate the matrices as and when they are needed. The peak memory usage for an RR-GCN with embedding size e is dominated by (1) the previous-layer node representations, (2) the previous-layer PPV features, (3) the accumulator matrix X (Algorithm 2) and (4) the intermediate results for *a single* relation type; four $|\mathcal{V}| \times e$-dimensional float-matrices, which is comparable to the memory complexity of an R-GCN's forward pass during inference. RR-GCNs clearly need much less memory than their trained counterparts *for a given embedding size*. However, as these embeddings result from combinations of randomly

transformed random representations, to capture useful information, a larger embedding size is needed compared to R-GCNs, which are trained to select *only* the useful features for the downstream task. As such, RR-GCNs are not necessarily more memory efficient than R-GCNs, depending on how many random features are necessary for the downstream task. For DBLP and an RR-GCN with embedding size 512 as an example, the resulting peak memory usage is 36.62GB. It is important to note that, once trained, R-GCN inference actually requires less memory because of the more compact representations. The main advantage of RR-GCN is that it provides inference without requiring additional training.

Attenuation approach acts as an intermediate between R-GCN and RR-GCN by requiring backward and intermediate activations to be kept in memory (with activation checkpointing to reduce the peak memory usage to fit into a gpu). However the weights are still generated as required and far fewer parameters are used overall and as a result, the peak memory usage is still smaller than that of the R-GCN.

6. Conclusion

Inspired by the success of random non-linear transformations in the time series domain, the application of random convolutions (RR-GCN) to KGs was evaluated. In this exploratory study, empirical results show that random transformations usually match or exceed the performance of end-to-end trained R-GCNs. The experiments indicate that a KG's *structure* alone, which is the only thing RR-GCNs can capture, contains enough semantic information as is, and that when R-GCNs do perform better, they mostly learn to *ignore* certain parts of that structure. Some of this advantage can be negated using attenuation, trading in the ease of training for performance. The RR-GCN with attenuation has demonstrated a capability to beat the performance of trained R-GCNs.

Aside from surprisingly good performance, RR-GCNs exhibit other interesting properties: (1) embeddings can easily be generated for a small subset of the graph for testing purposes (whereas R-GCNs need to process the entire graph to train their weights), and (2), as training through backpropagation is not necessary, they can be used to test the influence of new e.g. aggregation functions (such as PPV) before considering differentiable variants, as a new computationally inexpensive baseline model for comparisons. This opens up many new interesting avenues for further research, both in terms of improving the RR-GCNs themselves and with regards to KG embedding in general.

Acknowledgments

The authors wish to thank David Vander Mijnbrugge and Michael Weyns for their valuable inputs during the paper review. This study was partially funded by the Flemish Government under the Flanders Artificial Intelligence Research Program and the VLAIO O&O ADAM project with AZ Delta (HBC.2020.323).

Code and data availability

The AIFB, AM, BGS, and MUTAG KGs used in the paper, are public benchmark datasets downloadable using Pytorch geometric library (https://pytorch-geometric.readthedocs.io/). The larger KGs are available using kgbench-loader (http://kgbench.info/) library. The code used to obtain the results in this paper is available at https://github.com/predict-idlab/RR-GCN/.

References

[1] P. Ristoski and H. Paulheim, Semantic web in data mining and knowledge discovery: A comprehensive survey, *Journal of Web Semantics* **36** (2016) 1–22.
[2] M. Palmonari and P. Minervini, Knowledge graph embeddings and explainable ai, *Knowledge Graphs for Explainable Artificial Intelligence: Foundations, Applications and Challenges, IOS Press,, Amsterdam* (2020) 49–72.
[3] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, Relational inductive biases, deep learning, and graph networks, *arXiv preprint arXiv:1806.01261* (2018).
[4] Z. Wang, Y. Wang, C. Yuan, R. Gu and Y. Huang, Empirical analysis of performance bottlenecks in graph neural network training and inference with gpus, *Neurocomputing* **446** (2021) 165–191.
[5] A. Dempster, F. Petitjean and G. I. Webb, Rocket: exceptionally fast and accurate time series classification using random convolutional kernels, *Data Mining and Knowledge Discovery* **34**(5) (2020) 1454–1495.
[6] A. Dempster, D. F. Schmidt and G. I. Webb, Minirocket: A very fast (almost) deterministic transform for time series classification, in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, (ACM, 2021), pp. 248–257.
[7] B. Steenwinckel, G. Vandewiele, M. Weyns, T. Agozzino, F. De Turck and F. Ongenae, Ink: knowledge graph embeddings for node classification, *Data Mining and Knowledge Discovery* (2021) p. in production.
[8] G. Vandewiele, B. Steenwinckel, F. De Turck and F. Ongenae, Mindwalc: mining interpretable, discriminative walks for classification of nodes in a knowledge graph, *BMC Medical Informatics and Decision Making* **20**(4) (2020) 1–15.
[9] G. K. de Vries, A fast approximation of the weisfeiler-lehman graph kernel for rdf data, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, (Springer, 2013), pp. 606–621.
[10] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor and K. M. Borgwardt, Graph kernels, *Journal of Machine Learning Research* **11** (2010) 1201–1242.
[11] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston and O. Yakhnenko, Translating embeddings for modeling multi-relational data, in *Advances in neural information processing systems*, (NeurIPS, 2013), pp. 2787–2795.
[12] S. M. Kazemi and D. Poole, Simple embedding for link prediction in knowledge graphs, *arXiv preprint arXiv:1802.04868* (2018).
[13] B. Yang, W. tau Yih, X. He, J. Gao and L. Deng, Embedding entities and relations for learning and inference in knowledge bases (2015).
[14] T. Dettmers, P. Minervini, P. Stenetorp and S. Riedel, Convolutional 2d knowledge graph embeddings, in *Thirty-second AAAI conference on artificial intelligence*, (AAAI, 2018).
[15] S. Zhang, Y. Tay, L. Yao and Q. Liu, Quaternion knowledge graph embeddings, in

Advances in Neural Information Processing Systems, eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett **32**, (Curran Associates, Inc., 2019).

[16] T. Mikolov, K. Chen, G. Corrado and J. Dean, Efficient estimation of word representations in vector space (2013).

[17] P. Ristoski and H. Paulheim, Rdf2vec: Rdf graph embeddings for data mining, in *International Semantic Web Conference*, (Springer, 2016), pp. 498–514.

[18] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov and M. Welling, Modeling relational data with graph convolutional networks, in *European semantic web conference*, (Springer, 2018), pp. 593–607.

[19] L. Cai, B. Yan, G. Mai, K. Janowicz and R. Zhu, Transgcn, *Proceedings of the 10th International Conference on Knowledge Capture* (Sep 2019).

[20] D. Nathani, J. Chauhan, C. Sharma and M. Kaul, Learning attention-based embeddings for relation prediction in knowledge graphs, *arXiv preprint arXiv:1906.01195* (2019).

[21] S. Vashishth, S. Sanyal, V. Nitin and P. Talukdar, Composition-based multi-relational graph convolutional networks, *arXiv preprint arXiv:1911.03082* (2019).

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

[23] M. Chen, Z. Wei, Z. Huang, B. Ding and Y. Li, Simple and deep graph convolutional networks, in *International Conference on Machine Learning*, (Proceedings of Machine Learning Research, 2020).

[24] G. Li, C. Xiong, A. K. Thabet and B. Ghanem, Deepergcn: All you need to train deeper gcns, *ArXiv* **abs/2006.07739** (2020).

[25] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio and C.-J. Hsieh, Cluster-gcn, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Jul 2019).

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, Attention is all you need (2017).

[27] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, Graph attention networks (2017).

[28] S. Brody, U. Alon and E. Yahav, How attentive are graph attention networks? (2021).

[29] K. K. Thekumparampil, C. Wang, S. Oh and L.-J. Li, Attention-based graph neural network for semi-supervised learning (2018).

[30] D. Kim and A. Oh, How to find your friendly neighborhood: Graph attention design with self-supervision, in *International Conference on Learning Representations*, (OpenReview.net, 2021).

[31] E. Ranjan, S. Sanyal and P. Talukdar, Asap: Adaptive structure aware pooling for learning hierarchical graph representations, *Proceedings of the AAAI Conference on Artificial Intelligence* **34** (Apr 2020) p. 5470–5477.

[32] D. Busbridge, D. Sherburn, P. Cavallo and N. Y. Hammerla, Relational graph attention networks (2019).

[33] H. Zhang, Z. Yu, G. Dai, G. Huang, Y. Ding, Y. Xie and Y. Wang, Understanding gnn computational graph: A coordinated computation, io, and memory perspective (2021).

[34] C. Huang, M. Li, F. Cao, H. Fujita, Z. Li and X. Wu, Are graph convolutional networks with random weights feasible?, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **45**(3) (2023) 2751–2768.

[35] B. Schrauwen, D. Verstraeten and J. Campenhout, An overview of reservoir computing: Theory, applications and implementations, in *Proceedings of the 15th European*

Sympsosium on Artificial Neural Networks, (ESANN, 01 2007), pp. 471–482.
[36] P. Geurts, D. Ernst and L. Wehenkel, Extremely randomized trees, *Machine learning* **63**(1) (2006) 3–42.
[37] T. Kipf, How powerful are graph convolutional networks?
[38] V. C. Chian, M. Hildebrandt, T. Runkler and D. Dold, Learning through structure: towards deep neuromorphic knowledge graph embeddings (2021).
[39] R. Abboud, İ. İ. Ceylan, M. Grohe and T. Lukasiewicz, The surprising power of graph neural networks with random node initialization, *CoRR* **abs/2010.01179** (2020).
[40] R. Sato, M. Yamada and H. Kashima, Random features strengthen graph neural networks, *CoRR* **abs/2002.03155** (2020).
[41] M. Galkin, J. Wu, E. Denis and W. L. Hamilton, Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs, *CoRR* **abs/2106.12144** (2021).
[42] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, *Journal of Machine Learning Research - Proceedings Track* **9** (01 2010) 249–256.
[43] T. Thanapalasingam, L. van Berkel, P. Bloem and P. Groth, Relational graph convolutional networks: A closer look (2021).
[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32*, eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett (Curran Associates, Inc., 2019) pp. 8024–8035.
[45] M. Fey and J. E. Lenssen, Fast graph representation learning with PyTorch Geometric, in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, (ICLR, 2019).
[46] P. Ristoski, G. K. D. De Vries and H. Paulheim, A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web, in *International Semantic Web Conference*, (Springer, 2016), pp. 186–194.
[47] P. Ristoski, G. K. D. de Vries and H. Paulheim, A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web, in *The Semantic Web – ISWC 2016*, (Springer International Publishing, Cham, 2016), pp. 186–194.
[48] P. Bloem, X. Wilcke, L. van Berkel and V. de Boer, kgbench: A collection of knowledge graph datasets for evaluating relational and multimodal machine learning, in *European Semantic Web Conference*, (Springer, 2021), pp. 614–630.
[49] A. V. Dorogush, V. Ershov and A. Gulin, Catboost: gradient boosting with categorical features support, *arXiv preprint arXiv:1810.11363* (2018).
[50] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2017).
[51] S. Zhu, S. Zhang, Y. Liu, C. Zhou, S. Pan, Z. Li and H. Chen, Rhgnn: imposing relational inductive bias for heterogeneous graph neural network, *International Journal of Machine Learning and Cybernetics* (2024) 1–17.
[52] M. Chen, Y. Zhang, X. Kou, Y. Li and Y. Zhang, r-gat: Relational graph attention network for multi-relational graphs, *ArXiv* **abs/2109.05922** (2021).
[53] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2014), cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
[54] T. Chen, B. Xu, C. Zhang and C. Guestrin, Training deep nets with sublinear memory cost (2016).