Improving the Relational Inductive Bias of Graph Neural Networks

UNIVERSITY OF OXFORD

Candidate no. 1058615

Word count: 16,071 (from Overleaf)

Submitted in partial completion of the

Master of Science in Advanced Computer Science

☑ I hereby certify that this is entirely
my own work unless otherwise stated.

Trinity 2022

Abstract

Knowledge graphs, networks of entities and the relations between them, have emerged as a useful data structure for storing general knowledge. One of the main ways they are used in the context of machine learning is in knowledge completion tasks, where various types of models are challenged to find missing links in a given knowledge graph. Shallow embedding models, simple models that attempt to learn how to represent entities and relations, have so far been dominant for knowledge completion tasks in the transductive setting, but are unable to complete inductive knowledge completion tasks. To ameliorate this, some inductive knowledge completion models that utilise graph neural networks in unique ways have been proposed, each with their own advantages and disadvantages.

We propose GraphRE, a new neural model for tackling knowledge completion tasks in an inductive setting. The GraphRE model works by integrating a shallow embedding model into the message-passing scheme of a graph neural network using the shallow model as a tool to transform node representations during message-passing. We perform transductive and inductive experiments to show that the GraphRE model is competitive with other state-of-the-art inductive knowledge completion models despite being more lightweight and foregoing much of the pre-processing steps taken by many other inductive knowledge completion models. We also run smaller focused experiments to explore various aspects of the GraphRE model. These include looking at how the GraphRE model differs from classical graph neural network models when it comes to picking an optimal aggregation scheme, exploring a new way the GraphRE model can combine node representations with those of their neighbourhoods by taking advantage of the structure of its underlying knowledge graph, and examining the effect dimensionality has on the performance of the model.

Contents

Appendices

List of Figures

List of Abbreviations

WN18	WordNet18 [1]; a knowledge graph dataset created by taking a subset of WordNet, a graph modelling the lexical relations between words [2].
WN18RR	WordNet18RR [3]; a reduced version of WN18 that removes its inverse relations.
FB15k	Freebase15k [1]; a knowledge graph dataset created by taking a subset of Freebase, a general knowledge dataset made by organising human knowledge [4].
FB15k-237	Freebase15k-237 [5]; a reduced version of FB15k that removes its inverse relations.
NELL	Never-Ending Language Learning [6]; a knowledge graph based on the NELL dataset originally compiled in 2010 through a continuously running web-scraping program.
TransE	Translational embedding [1]; a kind of shallow embedding model that represents the relations within a knowledge graph as vectors. These are then added to the head entities of the graph's triples to approximate their tail entities.
RotatE	Rotational embedding [7]; a kind of shallow embedding model that represents the entities and relations of a knowledge graph in the complex plane and models triples by rotating head entities along vectors representing relations to approximate tail entities.
RESCAL	[8]; a type of bilinear shallow embedding model that represents each relation in a knowledge graph using a dense 2-dimensional matrix.
DistMult	[9]; a type of bilinear shallow embedding model that is a modified version of the RESCAL model. It limits the representations of relations in knowledge graphs so they can only be represented using diagonal matrices.

ComplEx . . .	Complex embedding [10]; a type of bilinear shallow embedding model that is almost the same thing as the DistMult model except that it extends the representations of the entities and relations of knowledge graphs into the complex plane.
GNN	Graph neural network; a kind of neural model that operates on graph structured data for the purpose of analysis or to make predictions about new unseen data.
GCN	Graph convolutional network [11]; a type of message-passing GNN where the embeddings of node neighbours share their weights central nodes after aggregation.
WL Test . . .	Weisfeiler-Lehman test [12]; an imperfect algorithm for determining whether two graphs are isomorphic using an iterative colouring technique.
GIN	Graph isomorphism network [13]; a type of message-passing GNN that is designed to have as much expressive power as the WL test while remaining scalable and efficient.
R-GCN	Relational graph convolutional network [14]; a kind of GNN used for completing knowledge graphs by treating each relation type as its own induced subgraph then doing standard link prediction over those graphs.
GraIL	Graph inductive learning [15]; a kind of GNN framework used to do inductive knowledge graph completion by inducing subgraphs for candidate triples made up of the paths connecting the two entities in the triple, then tuning a GNN on that new subgraph.
INDIGO . . .	[16]; a kind of GNN framework used to do inductive knowledge graph completion by constructing a new graph made up of nodes that represent triples in the knowledge graph as well as a predefined set of candidate triples, then using a GCN network to score those nodes.
NBFNet . . .	Neural Bellman-Ford network [17]; a kind of GNN used to do inductive knowledge graph completion by using message-passing to approximate path embedding summations for scoring triples in knowledge graphs.
GraphRE . . .	Graph relational embedding; the new neural model introduced in this thesis for tackling inductive knowledge graph completion tasks by performing message-passing and transforming messages using a shallow embedding model while they are being passed along.

MRR	Mean reciprocal rank; a metric used in ranking tasks. After models rank potential candidates based on how likely they are to be real data, the average of the inverse of the ranks of the true data represents the mean reciprocal rank.
Hits@k	a metric used in ranking tasks. After models rank potential candidates based on how likely they are to be real data, the proportion of the real data that has a rank less than or equal to k represents the Hits@k score.
AUC-PR	Area under the curve (precision-recall); a useful metric used in binary classification tasks with a noticeable class imbalance. It is bounded between 0 and 1 and represents the area under the curve of a plot between the precision and recall of a particular model.

1
Introduction

Contents

We begin this thesis by going through some of the major background knowledge surrounding the topics that will be tackled in this project.

1.1 Motivation

Over the last decade, the once budding field of graph representation learning has grown from a small niche in the machine learning community to a rich research landscape with a wide array of useful applications [18]. This development has been motivated in large part by the ubiquitous nature of graph-structured data. We find graphs everywhere: social media apps [19], transportation systems [20] computer networks [21], and even the molecules making up the world around us [22]. Understanding them and being able to answer complicated questions about them could therefore help us uncover innovative solutions to problems in a broad range of adjacent research areas [23].

With all this in mind, let us unpack briefly what constitutes a graph and why they are so challenging to work with in certain conditions. A simple graph is a tuple (V, E) where V is a set of nodes (or points) and E is the set of edges which connect the nodes together. For example, three points connected together to form a triangle forms a graph where each side of the triangle is an edge and the three points of the triangle are nodes. This example graph could be represented in the form:

$$G = (V, E) \; where \; V = \{a, b, c\}, \; E = \{(a, b), (b, c), (c, a)\}$$

Edges have the potential to additionally come in undirected and directed varieties. As the names suggest, a directed edge has a direction while an undirected one does not. This means that, for an undirected edge (a, b), the following holds:

$$(a, b) = (b, a)$$

while this would not be the case if (a, b) were a directed edge. If (a, b) were a directed edge, we would call a the *head* of the edge and b the *tail* of the edge.

Having established the concept of a graph, we can now imagine how graphs can become much more complex and intricate in the extreme. There are currently about 2.9 billion active Facebook users[1], if we consider the graph where each of these users is a node and two users are connected with an edge if they are “friends” on the platform, it becomes clear how these types of large graphs can quickly get difficult to analyse.

We can impose additional conditions on graphs and the underlying data they represent to formulate knowledge graphs. Knowledge graphs are a data structure used to represent graphs made of different entities (nodes) that are connected to each other through different types of relations [24]. Knowledge graphs can be used to connect together entities into organised graphs that represent something more akin to general knowledge, hence their name [24].

---

[1] https://datareportal.com/essential-facebook-stats

In this thesis we will explore a way of analysing knowledge graphs using a new type of graph neural model that leverages the benefits of simpler, shallower models and exports those benefits into a more intricate neural space. This new construction will allow us to answer questions about knowledge graphs that shallower models are unable to answer. We will then compare this new model with other similar neural models designed to tackle analogous graph analysis problems.

1.2 Background

We now go through some of the tools and ideas making up the foundation of the research of this thesis.

1.2.1 Knowledge Graphs

Knowledge graphs are a useful and widespread tool for storing and managing data. They utilise directed graphs to map relationships between real-world entities and allow for various tasks like query answering and reasoning [24]. Knowledge graphs are sets of triples, also known as facts, that each encode the way in which one entity interacts with another [25]. We will first define in more detail what these triples are and then give a formal definition of what a knowledge graph is. For the purposes of this thesis, we will be using the Resource Description Framework (RDF) format when discussing knowledge graphs and the elements that make them up [26].

We now define formally the concept of a triple. Given a relation r and two entities c and d, a triple is a tuple (c, r, d). Conventionally, c is the *head entity* of the triple and d is the *tail entity* of the triple. Intuitively, a triple is meant to represent the fact that c is related to d through relation r.

Triples in general can be thought of as a form of directed edge whose edge type is given by the relation of the triple and whose head and tail are represented by the triple's entities. For example, we can imagine that information about Oxford might be encoded using triples such as $(Alice, type, Student)$, $(Bob, type, Professor)$, and $(Bob, supervisorOf, Alice)$ to model the fact that Alice is a student, Bob is a professor, and Bob is Alice's supervisor. Because of the way triples can capture the way entities interact with one another, we can analogously represent triples using directed edges and map them all together into a graph [25].

Now that we understand the idea of a triple, we can formally introduce the concept of a knowledge graph. Given a set of entities E and a set of relations R, a knowledge graph (KG) K is a set of triples over E and R.

This simple construction can be extended to capture data of a wide variety of types connected together through many different types of relations, which is why knowledge graphs are considered to be the most apt data structure for modelling "general knowledge" and have, for example, been used to try and organise the way facts are represented in online environments [27], retrieve information [28], model human languages [29], and enhance recommender systems [30].

While there are many different types of tasks and challenges posed by knowledge graphs in the context of machine learning, this thesis focuses on the task of knowledge completion. The goal of models doing knowledge completion tasks is to "complete" knowledge graphs with connections that they may be missing [31]. Continuing our previous example regarding information about Oxford, it could be that there exists a complementary relation "$supervisedBy$" in addition to the "$supervisorOf$" relation. In this case, if a graph only contains the triple $(Bob, supervisorOf, Alice)$ but not $(Alice, supervisedBy, Bob)$ then a model trained to "complete" that knowledge graph would ideally suggest that the missing triple be added to the graph. In this case, the model would have hopefully learned that "$supervisedBy$" and "$supervisorOf$" are inverse relations. These types of connections between relations are called inference patterns [32]. In general, there are many different ways that relations can be linked to each other, and we can model these links by defining various types of inference patterns. We will now list some of the most prominent types of inference patterns:

- Symmetric relations: A relation r is symmetrical if, for every triple (c, r, d), the triple (d, r, c) holds.

- Anti-symmetric relations: A relation r is anti-symmetric if, for every triple (c, r, d), there should not exist a triple (d, r, c).
- Inverse relations: Given a relation r_1, a relation r_2 is the inverse relation of r_1 if, for every triple (c, r_1, d), the triple (d, r_2, c) holds and vice versa.
- Composition: Given two relations r_1 and r_2, a relation r_3 composes r_1 and r_2 if, for every pair of triples (c, r_1, d) and (d, r_2, t), the triple (c, r_3, t) holds.
- Hierarchical relations: Given a relation r_1, a relation r_2 has a hierarchical relationship with r_1 if, for every triple (c, r_1, d), the triple (c, r_2, d) holds.
- Intersectional relations: Given two relations r_1 and r_2, a relation r_3 intersects r_1 and r_2 if, for every pair of triples (c, r_1, d) and (c, r_2, d), the triple (c, r_3, d) holds.
- Mutually exclusive relations: Given a relation r_1, a relation r_2 is mutually exclusive with r_1 if, for every triple (c, r_1, d), there should not exist a triple (c, r_2, d).

There can exist other, more complicated types of inference patterns but the ones listed above tend to represent the majority of the patterns found between relations in real-world datasets [32].

Neural models described in upcoming sections will be challenged to ideally identify rule inference patterns in the knowledge graphs they train on. If a particular model is able to theoretically identify any possible rule inference pattern, the model is deemed to be *fully expressive*. While this may sound powerful in the abstract, in practice fully expressive models do not necessarily perform better than non-fully expressive ones [7]. For starters, fully expressive models can be more prone to overfit the data they train on and also tend to be less scalable than lighter models [33].

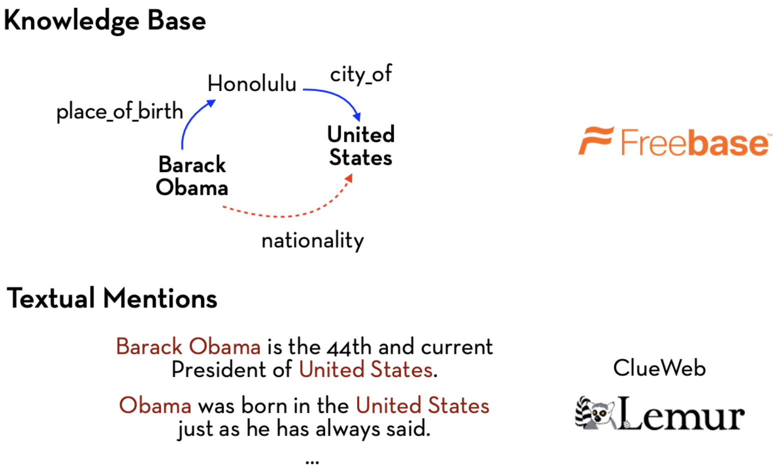

Knowledge Base
Honolulu
city_of
place_of_birth
United States
Barack Obama
nationality
Freebase
Textual Mentions
Barack Obama is the 44th and current President of United States.
Obama was born in the United States just as he has always said.
...
ClueWeb
Lemur

Figure 1.1: An example of information captured by Freebase. (From [5])

Knowledge Graph Datasets

Training and testing models meant to operate on knowledge graphs necessitates robust real-world datasets. To this end, a few prominent knowledge graphs have been designed over the years to be used for this purpose [34]. We now go through those datasets and briefly discuss some of their properties.

WordNet18 [1]:

One of the first and most widely used knowledge graph datasets is WordNet18 (WN18). This dataset is a subset of WordNet, a graph originally put together in 1995 that maps out the lexical relations between different words [2]. The most prominent inference patterns present in this dataset are symmetry, anti-symmetry, and inverse relations [7].

WordNet18RR [3]:

Four years after WN18 was introduced, a subset of WN18 was created to control more closely what types of inference patterns the models training on the dataset would be exposed to. The new dataset was named WordNet18RR (WN18RR) and is essentially just the WN18 graph with its inverse relations deleted. The most prominent inference patterns in this dataset are therefore symmetric, anti-symmetric, and composition relations [7].

Freebase15k [1]:

Another important knowledge graph dataset is Freebase15k (FB15k). This dataset is a subset of Freebase, a knowledge graph constructed in 2008 by accruing general human knowledge and connecting it together [4]. An example of the type of data contained in Freebase can be found in Figure 1.1. The most prominent inference patterns present in this dataset are symmetry, anti-symmetry, and inverse relations [7].

Freebase15k-237 [5]:

In a move similar to the creation of WN18RR, two years after FB15k was created, a smaller version of the dataset was created to tune the distribution of inference patterns in the graph. The inverse relations present in FB15k were deleted to create a new dataset: Freebase15k-237 (FB15k-237). The most prominent inference patterns present in this dataset are symmetry, anti-symmetry, and composition relations [7].

Never-Ending Language Learning [6]:

Another important dataset to consider is the Never-Ending Language Learning (NELL) knowledge graph dataset. The usable knowledge graph version of this dataset was created by taking a subset of the original NELL dataset and editing it to leave only meaningful relations behind. The original NELL graph comes from an ongoing program that continuously scrapes the web for information and analyses it to add to the graph [35]. The most prominent inference patterns present in this dataset are symmetry, anti-symmetry, and inverse relations.

1.2.2 Shallow Embedding Models

One particular class of models that tackles link prediction in the context of knowledge graphs are shallow embedding models. Shallow embedding models are relatively simplistic encoder-decoder models that use supervised learning to learn how to embed knowledge graph entities into a high-dimensional Euclidean space [36]. The relations between entities are then represented through learned functions on those entity embeddings that relate the heads and tails of the triples through a scoring function. There are a myriad of ways that scoring function can be defined, so those definitions in addition to other small decisions give rise to all the different types of shallow embedding models that are widely used today [25].

We now look at some of the major types of shallow embedding models:

TransE

The TransE model, introduced in 2013, is one of the first and simplest shallow embedding models [1]. The phrase “TransE” is a shortened version of “translational embedding,” which captures partly how the TransE model works.

First, we will define the number of dimensions that the Euclidean space of our embeddings will have, let us call this number d for the purposes of this section. The goal of the TransE model is to learn a vector embedding of size d for each entity of the knowledge as well as an optimal vector of size d for each type of relation in the dataset. The way TransE works is that, after training, if two entities c and m are related to each other in the knowledge graph via a relation r in a triple (c, r, m), then:

$$\mathbf{m} \approx \mathbf{c} + \mathbf{r}$$

where $\mathbf{c}$, $\mathbf{m}$, and $\mathbf{r}$ are the learned vectors representing c, m, and r respectively [1].

To train such a model, first the entities and relational embeddings are all randomly initialised. The model then iteratively uses gradient descent over a loss function defined as the following difference function:

$$L = d(\mathbf{t}, \mathbf{h} + \mathbf{r}) - d(\mathbf{t}', \mathbf{h}' + \mathbf{r}) + \gamma$$

where $\mathbf{h}$ and $\mathbf{t}$ represent the vectors for the heads and tails of real triples in the dataset respectively, $\mathbf{h}'$ and $\mathbf{t}'$ represent the vectors of “corrupted” triples that do not exist in the dataset (obtained via negative sampling), $\mathbf{r}$ represents the vector of the relational embedding for relation r, and γ is a slack term that is necessary because the model will likely not be able to perfectly satisfy these loss conditions in every case [1].

Because each relation is represented simply as a vector that is summed with entity embeddings, the TransE model is lightweight and easy to train. This simplicity, however, also means that the TransE model is quite limited in what types of relations and inference patterns it is able to represent. Most notably, TransE cannot capture symmetric relations [33]. This becomes clear when we look at how the model transforms head entities into tail entities. Given two true facts in the a knowledge graph (h, r, t) and (t, r, h), to capture both of these triples at the same time the TransE model would need the following to be true:

$$\mathbf{t} - \mathbf{h} = \mathbf{r} = \mathbf{h} - \mathbf{t}$$
$$\Rightarrow \mathbf{h} = \mathbf{t}$$

Therefore if $\mathbf{h} \neq \mathbf{t}$ then the symmetrical relation r cannot be captured by TransE.

While TransE can capture other inference patterns like anti-symmetry, inversion, and composition, the fact that it cannot capture symmetry is a big issue because symmetry is a commonly found inference pattern in real-world datasets.

Another important limitation of TransE is that it cannot capture 1-to-n relations. Assume we have the triples (c, r, d) and (c, r, h) in our knowledge graph. Then, in a TransE model:

$$\mathbf{d} = \mathbf{c} + \mathbf{r} = \mathbf{h}$$

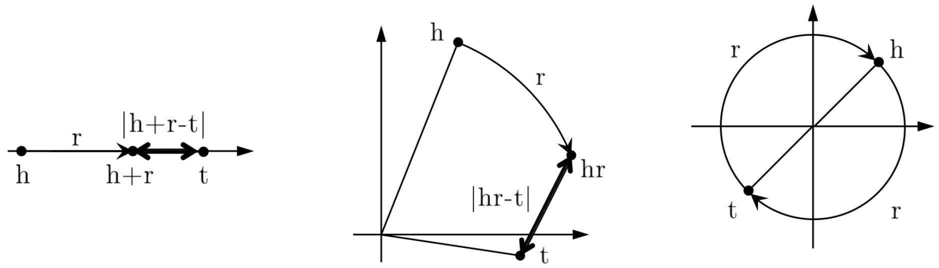
Therefore the TransE model assumes that $d = h$ when that may not be the case in reality.

To overcome these limitations, other more complicated models have been introduced to address TransE's drawbacks.

RotatE

This type of shallow embedding model is, like TransE, also quite simplistic, but is designed to specifically address some of the things TransE is not able to do, namely the issue of symmetric relations.

RotatE, a shortened way of writing rotational embedding, works by mapping each entity and relation in a knowledge into the complex plane [7]. Similarly to how TransE models relations as vectors that are added to the heads of triples to approximate their tails, RotatE models relations as complex vectors that rotate



(a) TransE models $\mathbf{r}$ as translation in real line.

(b) RotatE models $\mathbf{r}$ as rotation in complex plane.

(c) RotatE: an example of modeling symmetric relations $\mathbf{r}$ with $r_i = -1$

Figure 1.2: A visualisation of how RotatE works compared to TransE. (From [7])

the heads of triples to approximate their tails. Given a candidate triple (c, r, d), we can formalise this in the following way:

$$\mathbf{d} \approx \mathbf{c} \odot \mathbf{r}$$

where $\mathbf{c}$, $\mathbf{d}$, and $\mathbf{r}$ are the learned complex vector representations of c, d, and r respectively [7]. This formulation is visualised in Figure 1.2.

Training a RotatE model first involves initialising the complex representations of all the entities and relations randomly. The model then uses gradient descent in tandem with supervised learning to iteratively improve its embeddings using the following loss formulation:

$$L = -\log \sigma(\gamma - d_r(\mathbf{h},\ \mathbf{t})) - \sum_{i=1}^{n} \frac{1}{k} \log \sigma(d_r(\mathbf{h}'_i, \mathbf{t}'_i) - \gamma)$$

where $d_r(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} \odot \mathbf{r} - \mathbf{b}\|$ is the distance function used to measure the loss of a single candidate triple, $\mathbf{h}$ and $\mathbf{t}$ are the complex vector representations of the heads and tails of the triples in the knowledge graph respectively, k is the number of negative samples per true fact, $\mathbf{h}'_i$ and $\mathbf{t}'_i$ are the i^{th} negative sample's complex head and tail representations respectively, γ is a slack term to make sure the loss can converge even in cases where the relational rotations are only approximations, and n is the total number of negative samples.

RotatE is useful because it can capture all of the same inference patterns that TransE can capture, but additionally can handle symmetry. In fact, it can be shown that TransE is just a special case of RotatE. The way that RotatE captures symmetry relations comes from the fact that its relations are represented through rotations rather than translations. Given a symmetric relation, the complex vector representing that relation would translate into a 180° rotation around a circle. A head entity rotated twice through such a relation vector would therefore end up back in the same place, reflecting the symmetric nature of the relation.

Despite this added benefit, RotatE still has come major drawbacks. For one, the types of symmetric relations it can fit are limited in their expressivity. To demonstrate this, let us take two entities, c and d, which are related to each other through symmetric relations r and s such that (c, r, d) and (d, s, c) are two triples contained in the knowledge graph. A RotatE model would be able to capture the symmetry of these two relations as is, but now let us add a new entity h that is related to d by the triple (d, s, h). The following must therefore be true:

$$\mathbf{c} = \mathbf{d} \odot \mathbf{s} = \mathbf{h}$$

Unless $c = h$, RotatE is not capable of handling the relation s. Like TransE, this means that RotatE cannot handle 1-to-n relations, but in addition to this limitation, RotatE adds on the additional assumption that (h, r, d) holds because of the symmetric connection between r and s. This means that RotatE can only handle symmetric relations that are purely symmetric, and cannot handle relations that are symmetric but are also involved in additional triples beyond the ones that are mirrored versions of one another.

While RotatE does expand on the capabilities of TransE, to overcome the problem of 1-to-n relations we need to move to a new strain of shallow embedding models altogether.

RESCAL

One way to overcome the issue of 1-to-n relations that models like TransE and RotatE could not handle is to use a new category of shallow embedding models: bilinear models. These are shallow embedding models that use a different approach, based on matrix factorisation, to learn relational embeddings. The first type of bilinear model we will look at is RESCAL [8].

In bilinear models, the embedded representation of relations takes the form of a matrix. In the case of RESCAL these matrices are dense, while the entities of the knowledge graph continue to be represented as simple vectors in a Euclidean space. To determine whether two particular entities are joined through a certain relation, RESCAL uses a scoring function that combines the embeddings of all three of these elements together through matrix multiplication. We can formalise this function in the case of the triple (c, r, d) like this:

$$\textsc{score}(c, r, d) = \mathbf{c}\mathbf{M}_r\mathbf{d}^T$$

where $\mathbf{c}$ and $\mathbf{d}$ are the vector representations of c and d respectively and $\mathbf{M}_r$ is the matrix representation of the relation r [8].

Depending on how the model is regularised, the exact loss function used to optimise these embeddings varies, but regardless of which method is used, the number of parameters used in a RESCAL model is an order of magnitude higher than in the translational (non-bilinear) models we previously discussed, meaning the training procedure for a RESCAL model is less scalable than that of TransE or RotatE.

Despite this drawback, RESCAL is fully expressive. This may appear on the surface to mean that RESCAL will always be a better performing model than TransE or RotatE, but in practice this property can cause RESCAL models to overfit to their training data and draw connections between relations that may not be actually expressed in the real world [37].

DistMult

We can motivate our second bilinear model, the DistMult model, as an attempt to dial back the RESCAL model and make it more scalable and less prone to overfitting [9]. DistMult and RESCAL are almost the same model mathematically except for the fact that the representational matrices of relations in DistMult are limited to only being diagonal matrices. This simple change both reduces the number of parameters back down one order of magnitude and allows us to replace costly matrix multiplications with simpler element-wise multiplications.

We can formalise this change with the following scoring function used by DistMult for a given triple (c, r, d):

$$\textsc{score}(c, r, d) = \mathbf{c}\mathbf{D}_r\mathbf{d}^T$$

where $\mathbf{c}$ and $\mathbf{d}$ are the vector representations of c and d respectively and $\mathbf{D}_r$ is the diagonal matrix representation of relation r [9].

While DistMult is much more lightweight and scalable than RESCAL, those improvements come at the cost of representation, and in the case of DistMult that cost is quite heavy. DistMult models lose the ability to represent anti-symmetry, inverse relation, and composition. Additionally DistMult models make the general assumption that all relations are symmetrical. We can see why when we examine the scoring function. Given a triple (c, r, d) in an arbitrary knowledge graph, the following is true:

$$\textsc{score}(c, r, d) = \mathbf{c}\mathbf{D}_r\mathbf{d}^T = \mathbf{d}\mathbf{D}_r\mathbf{c}^T = \textsc{score}(d, r, c)$$

This means that if a trained DistMult model predicts that a candidate triple should exist in a knowledge graph, the same model should necessarily predict that the symmetric version of that triple exists as well, meaning that the DistMult model assumes that relation r is symmetric.

ComplEx

Given the substantial representational drawbacks of the DistMult model, it is natural to posit whether it could be improved to make it more expressive while keeping its scalable properties. On that note, another important bilinear model we will look at is the ComplEx model [10]. ComplEx, a shortened version of complex embedding, is a model similar to DistMult, except for the fact that it extends the DistMult model into the complex plane. This simple switch extends the representational power of DistMult dramatically.

In the ComplEx model, its scoring function looks similar to DistMult's except for the fact that the model's entity and relation embeddings are made up of complex numbers. To make the scoring function usable, however, it only takes the the real part of the final complex score as the true score. Putting this all together, we can formalise ComplEx's scoring function, given a triple (c, r, d), in the following way:

$$\textsc{score}(c, r, d) = \mathrm{Re}(\mathbf{c}\mathbf{C}_r\mathbf{d}^T)$$

where $\mathbf{c}$ and $\mathbf{d}$ are the complex vector representations of c and d respectively and $\mathbf{C}_r$ is the complex diagonal matrix representing relation r [10].

ComplEx's extension into the complex plane not only gives the model much more representational power than DistMult, but actually makes the model fully expressive once again. Notably, ComplEx regains the ability to learn anti-symmetric and inverse relations and removes the assumption that all relations are symmetric that DistMult suffered from.

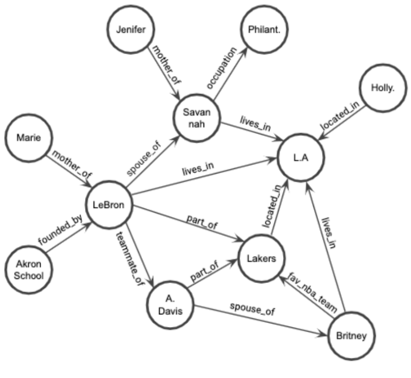
1.2.3 Transductive vs. Inductive Learning

This thesis is aiming to address the discrepancy between models that are capable of working in inductive learning environments and those only capable of working in transductive learning environments. We now explore the differences between these two types of learning.

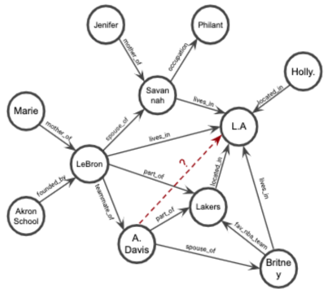
Transductive Learning

Transductive learning is only capable of producing results and conducting analysis on entities that were represented in the model's training set [38]. In other words, the model is incapable of learning an inductive function which governs the way its training data is structured and which could be used to make predictions about new, unseen data. We can visualise this type of learning by comparing the training set in part *a* of Figure 1.3 with the set in part *b* of the same figure.

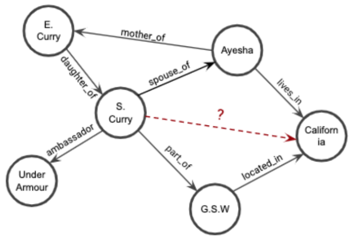
To get good performance on a test set, these types of models require that some amount of information about the objects in the test set is present during the training of the model [39]. In the context of knowledge graphs and embedding models, this means that, while the training, validation, and testing sets are disjoint partitions



a. Training graph



b. Transductive inference



c. Inductive inference

Figure 1.3: An example of transductive vs. inductive learning tasks in the case of a specific knowledge graph. (From [15])

of the total set of triples in the knowledge graph, all the triples in the testing and validation sets are made up of entities found in the training set. This is the type of learning that shallow embedding models engage in when they are applied to knowledge graphs, which is why these types of models do not perform well when challenged to make predictions about new, unseen entities [16].

Inductive Learning

Inductive learning is a form of learning wherein a model learns an inductive function that governs how its training data came to be in the first place so that the model can then generalise and make predictions about unseen data [15]. Instead of learning data-specific functions that only allow predictions to be made in a limited scope, the functions that inductive models learn have a deeper connection to the way in which the data was generated [40]. We can visualise this type of learning by comparing the training set in part *a* of Figure 1.3 with the set in part *c* of the same figure.

If a model is capable of this type of learning, then it can be tested using data created after the model's learning has already been completed. This makes these types of models much more useful for real-world tasks because they remain relevant for a lot longer and are easier to use in “live learning” environments where models are trained continuously as new data is streamed in [41]. Inductive models are also useful in the field of interpretable machine learning as a direct consequence of the fact that the rules they learn are more likely to approximate real connections that exist intrinsically in the world [42].

1.2.4 Graph Neural Networks

By their very nature, graphs have consistently proven to be difficult to integrate into the machine learning field. Their high potential for complexity makes them hard to transform into easily accessible formats for neural models without losing some of the inherent structural biases that graphs posses [43]. While there was some development on the side of using the spectral properties of graphs to integrate them with neural models, these models did not adequately capture many of the previously mentioned structural biases of graph [44–47]. Solving these problems required the development of a new type of neural model: graph neural networks (GNNs).

GNN models work by taking as input a graph and using a set of learned parameters to analyse or answer questions about the graph [48]. There are various ways that GNNs can leverage the structure of graphs to learn these parameters, and we will now look at a few of them.

GNN Preliminaries

We now go through some of the fundamentals of GNNs needed to understand the core message-passing framework that powers them.

First, we will define node neighbourhoods. Given an undirected graph G with a set of nodes V and edges E, the neighbourhood of a node c in V is the set $N(c)$ such that:

$$N(c) = \{s \mid \exists e \in E \ s.t. \ e = (s, c)\}$$

where (a, b) refers to an edge connecting nodes a and b.

In the directed case, nodes have both an in-neighbourhood and and out-neighbourhood that correspond to incoming and outgoing edges respectively.

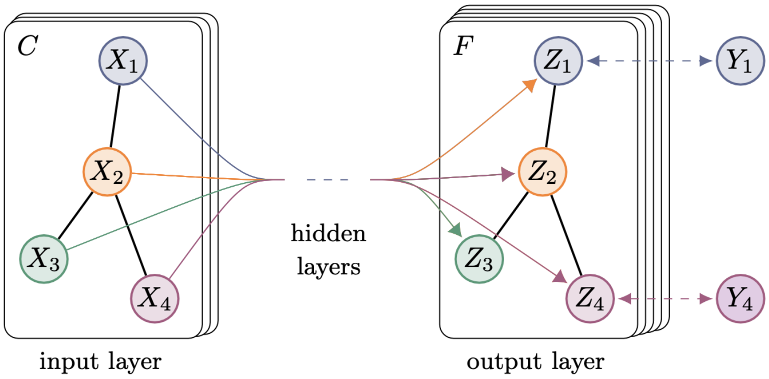


Figure 1.4: A visualisation of the learning process utilised by message-passing graph neural networks. (From [11])

We also need to introduce the concept of layers to understand GNNs. A full GNN model is made up of a number of message-passing layers stacked on top of one another. The input of each layer is a set of node embeddings which the layer then feeds through a message-passing process to output a new set of node embeddings. The final node embeddings of the entire model are either left as is if the model is completing a node-level task or they are combined in some way to form a graph embedding if the model is completing a graph-level task.

Message-Passing Neural Networks

We now present the message-passing neural network (MPNN) framework: the core concept that defines how popular GNN layers work [49].

The message-passing model is a pivotal innovation in the field of GNNs. Instead of translating graphs into the Euclidean domain before training and thus losing some of the structural information of the graph [50, 51], message-passing models learn the model parameters by directly using the structure of the graph to inform how the representations of each of the nodes is distributed during the training process. They are based on the idea that, starting with some set of initial node features, we can iteratively update node representations using information passed along edges from local neighbourhoods. We can see a visualisation of this idea in Figure 1.4.

We now formalise these ideas by first going through node embedding (representation) notation. We write $\mathbf{h}_u^{(t)}$ as the node embedding of node u at layer

t, where $\mathbf{h}_u^{(0)}$ is node u's initial features. With that, we can formalise the core message-passing process in the following way:

$$\mathbf{h}_u^{(t)} = \textsc{combine}^{(t)}(\mathbf{h}_u^{(t-1)},\ \textsc{aggregate}^{(t)}(\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\})) \tag{1.1}$$

where $N(u)$ is the neighbourhood of node u, $\textsc{aggregate}^{(t)}(\cdot)$ is a differentiable and permutation-invariant function at layer t that aggregates the representations of a set of nodes, and $\textsc{combine}^{(t)}(\cdot)$ is a differentiable function at layer t that combines the previous representation of a particular node with the aggregated representation of its neighbourhood. The $\textsc{aggregate}^{(t)}(\cdot)$ function is most commonly set as either a summation, mean, or max function.

If the $\textsc{aggregate}^{(t)}(\cdot)$ and $\textsc{combine}^{(t)}(\cdot)$ functions are the same across all layers, we call the model *homogeneous* and the superscripts on the function names can be dropped, otherwise we call the model *non-homogeneous*. For the purposes of this thesis, all our models are assumed to be homogeneous.

We can derive a baseline MPNN model formally by first setting $\textsc{combine}(\cdot)$ and $\textsc{aggregate}(\cdot)$ to some concrete functions. For example, let us use a summation aggregation scheme and set $\textsc{combine}(\cdot)$ to be a weighted sum transformed followed by a non-linear function $\sigma(\cdot)$. Then our baseline MPNN model becomes:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right) \tag{1.2}$$

where $\mathbf{W}_{self}^{(t)}$ and $\mathbf{W}_{neigh}^{(t)}$ are learnable weight parameters at layer t for the central node representation and aggregated neighbourhood representation respectively.

The first type of message-passing GNN that was designed using a strategy akin to this scheme is the graph convolutional network (GCN) model [11]. GCNs take a slightly simplified version of the base message-passing model by combining the neighbour-specific weight matrix and the self-specific weight matrix. We now present the base form of GCNs below:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(u) \bigcup \{u\}} \frac{\mathbf{h}_v^{(t-1)}}{\sqrt{N(u) + N(v)}}\right) \tag{1.3}$$

where $\mathbf{W}^{(t)} \in \mathbb{R}^{f_t \times f_t}$ is the matrix of weights at layer t, and $\sigma(\cdot)$ is a differentiable non-linearity [11].

We can see from Equation 1.3 that, when using a GCN, the model is unable to differentiate between the representation of the central node and its neighbours during aggregation. In this way, the GCN model is similar to the form of the baseline MPNN model in Equation 1.2 if self-loops are added to the graph. Additionally, the GCN model normalises the representations of each node by the size of their neighbourhoods.

On the one hand, by using this simplified form, GCNs benefit from being fast and scalable, making them good candidate models for use on large complicated graphs. On the other hand, the decision to mix in the central nodes' representations with that of their neighbours makes GCNs representationally limited, resulting in the model's inability to perform certain types of tasks on certain types of graphs [13, 52, 53]. We will explore the details of these limitations when we introduce the graph isomorphism network model.

Some other GNN variants include the GAT model [54] and GraphSAGE [55]. In the GAT model, an attention mechanism is added to the MPNN framework to allow nodes to learn to attend to and emphasise only neighbours that are useful to the task the model is being challenged to complete [54]. Meanwhile, GraphSAGE is a GNN framework that focuses on improving the inductive capabilities of the model by learning how to aggregate information from local neighbourhoods then generating the node embeddings using learning model parameters [55].

Moving forward with GNNs, it can become important to try and formalise how expressive they might be to better theoretically ground them. One way we might do this is by testing how well a particular model is at checking whether or not two graphs are isomorphic. This is known to be an NP-intermediate problem, but there does exist an efficient but imperfect algorithm, called the Weisfeiler-Lehman (WL) test, that can correctly tell whether two graphs are isomorphic in most cases [12]. By comparing the ability of GNNs to tackle the graph isomorphism problem with that of the WL test, it is possible to classify GNNs based on their expressive power.

An important realisation that researchers had when they conducted this investigation is that simple message-passing GNNs (GNNs that work merely off of the basic principle of transmitting and then combining messages along edges that are made up of node representations) are limited to only being as "powerful" as the WL test [13]. This means that no simple message-passing GNN can ever do better at the graph isomorphism problem than the WL test. Given this limitation, we can purposefully construct a new type of GNN model that is maximally powerful and expressive while still maintaining scalability, this model is the graph isomorphism network (GIN) model [13]. GIN achieves this by modeling an injective multiset function for neighbourhood aggregation.

While the previously discussed simple GNN models have shown to be very successful at various graph tasks, it might still be useful to design models that push past the limit placed by the WL test. These types of GNN models are labelled "higher order models" and can be exploited in specific instances for their unique theoretical properties, however, they can also quickly become too complicated to run on even moderately sized real-world datasets [53, 56]. Their increased expressive power does not necessarily improve their performance in other tasks, in some cases they they do not even perform as well as simpler, more lightweight models [57]. Additionally, this type of expressive power can also be achieved by randomising a portion of the node features, making the motivation behind higher order models somewhat superfluous [58].

As we can see, many different types of GNNs have been developed in the few years since the field took off. These models have shown promising results in tackling inductive problems related to graphs, an ability that we will use to develop a new way of tackling inductive learning tasks specifically in the realm of knowledge graph completion.

1.2.5 Relational Inductive Bias

Having introduced shallow embedding models, inductive learning, and GNNs, we now present the concept of relational inductive bias, what distinguishes it from the concept of an inductive learning environment, and why it is important to the work presented in this thesis.

We say that a model operating on a relational graph incorporates a *relational inductive bias* if the model is able to learn things like relational symmetries in the data, hierarchies, and other rule inference patterns that may be present in the data [42]. Shallow embedding models are designed to be have this property when they operate on knowledge graphs and we can clearly predict which types of rule inference patterns certain shallow embedding models will be able to find and which ones they will struggle with. Meanwhile, GNNs do not inherently have this property when operating on relational graphs.

To be clear, the concept of a relational inductive bias is distinct from that of an *inductive learning environment*. The former is a property that *models* have as described above, while the latter is a property of a type of *task* described in Section 1.2.3 where models are challenged to be able to generalise to unseen data not present in the models' training sets.

Models operating on knowledge graphs are particularly poised to benefit from having a relational inductive bias. This is because knowledge graphs, by nature, have a strong proclivity for their relations to be guided by principled structure because knowledge graphs represent *facts* about real-world entities [42]. This is why using a model that has a strong relational inductive bias could be a huge benefit in inductive knowledge completion tasks.

1.3 Problem Statement

We now present the main problem that we address in this work:

Given the fact that existing widely used shallow embedding models are currently unable to extend their capabilities into the inductive learning setting. but do exhibit a strong relational inductive bias, how can we merge some of the processes used by those shallow models with the inductive learning potential of graph neural networks to create a new type of neural model that is able to perform a form of message-passing directly on knowledge graphs in a scalable fashion and complete inductive tasks on those graphs at a state-of-the-art level all while having some desirable theoretical properties?

2
Related Work

Contents

We now go through some of the prominent models designed to tackle the main problem in this thesis or problems adjacent to it.

2.1 The R-GCN Model

The first impactful model to tackle this inductive knowledge completion issue is the relational GCN (R-GCN) model [14]. This model works on knowledge graphs by representing each relation in the graph with its own weight matrix. Given an entity in the graph, the node embeddings of that entity's neighbours are each transformed using the respective weight matrix that defines the relation connecting that neighbour to the central entity. These transformed embeddings are then all summed together along with the embedding of the central node and fed through a ReLU function to get the new embedding of the central entity in the next layer.

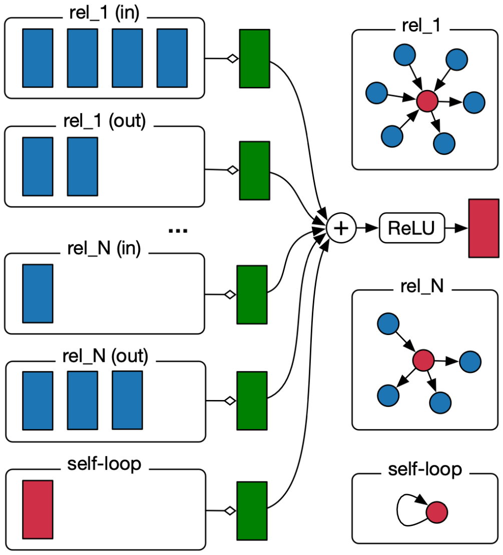


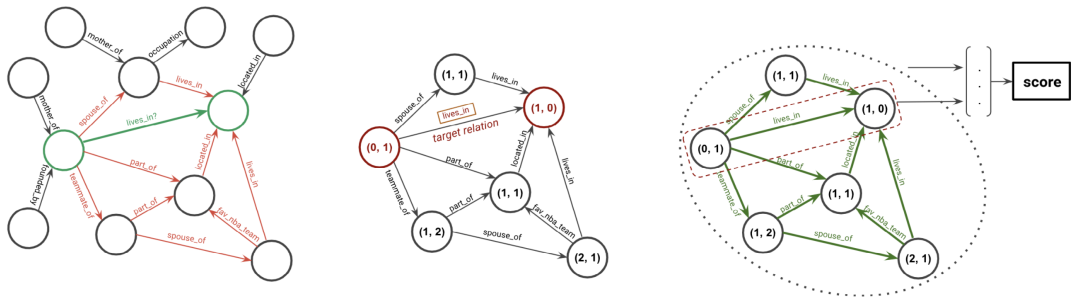
Figure 2.1: A visualisation of how R-GCN models work. (From [14])

We can formalise this process from a node-level perspective with the following equation dictating how the embeddings in the model change over time:

$$\mathbf{h}_i^{(t)} = \sigma\left(\sum_{r \in R}\sum_{j \in N_i^r} \frac{1}{c_{i,r}} \mathbf{W}_r^{(t)} \mathbf{h}_j^{(t-1)} + \mathbf{W}_0^{(t)} \mathbf{h}_i^{(t-1)}\right) \tag{2.1}$$

where R is the set of relations in the knowledge graph, $\mathbf{W}_r^{(t)}$ is the weight matrix of relation r at layer t (where $r = 0$ is the self-loop relation), and N_i^r is the subset of node i's neighbourhood that is connected to i through relation r [14]. This equation can be further visualised in Figure 2.1 above.

The R-GCN model has the benefit of being conceptually simple and tractable: by representing each relation with its own weight matrix, we are essentially training R different message-passing GNNs where the messages are partially shared. The downside of this construction is that the model can be prone to overfitting if one of the relations in the knowledge graph is too sparse or if there are too many relations [15]. Since each relation is given a full weight matrix, R-GCNs are, in principle, capable of capturing any type of inference pattern or rule that may be present in the knowledge graph when it comes to relations, but in practice the limited interactions between each of the relations mediated through the aggregation of messages makes it hard for R-GCNs to fully learn the underlying rules that may be present in the data [59].



1. Sample the enclosing sub-graph around the link to be predicted (target link).

2. Label the nodes w.r.t the target nodes to identify their structural role. Uniquely labels target nodes to mark them for the model.

3. Pass messages across the (sub-)graph to predict a score indicating how strongly the structure around the target link supports its logical plausibility.

Figure 2.2: A visualisation of how GraIL models work. (From [15])

R-GCNs have potential in the inductive learning space, but are held back by the fact that their learned node embeddings do not generalise well to unseen nodes unless the nodes themselves have features attributed to them. This limitation places the R-GCN model, with regards to its capabilities, in-between shallow embedding models and the other models we will look at next when it comes to knowledge graph completion.

2.2 The GraIL Model

The next important model we will look at is the graph inductive learning (GraIL) model [15]. This model is notably different from the R-GCN model because the GNN utilised in GraIL does not pass messages directly along edges in the knowledge graph. Instead, the message-passing occurs in subgraphs that are constructed to specifically facilitate inductive learning in the model.

The subgraphs induced by the GraIL model for each triple are made up of the paths that exist between the two nodes in the triple. These paths are limited in length by a parameter k and, for a triple (c, r, d), are made by calculating the intersection $N_k(c) \bigcap N_k(d)$, where $N_k(u)$ is the k-hop neighbourhood of node u. Since the GraIL model operates solely based on the structural information of the graph, the features of the nodes are set to be tuples representing the distance from the head and tail nodes respectively. The head and tail nodes themselves are uniquely labeled $(0, 1)$ and $(1, 0)$ respectively. With this setup complete, the model can now begin scoring triples. This is done by essentially running a GNN model over the induced subgraph that works similarly to the R-GCN model, then concatenating key node and relation embeddings from that model to get a graph-level embedding. The node embedding aggregation is formalised here:

$$\begin{aligned}\mathbf{h}_u^{(t)} &= \textsc{combine}^{(t)}(\mathbf{h}_u^{(t-1)}, \mathbf{a}_u^{(t)})\\ \mathbf{a}_u^{(t)} &= \textsc{aggregate}^{(t)}(\{\mathbf{h}_s^{(t-1)} : s \in N(u)\}, \mathbf{h}_u^{(t-1)})\end{aligned}$$

where $\textsc{aggregate}(\cdot)$ is the same multi-relational aggregation scheme used by the R-GCN model in Equation 2.1, $\textsc{combine}(\cdot)$ is a function also taken from the R-GCN model that uses a $W_{self}^{(t)}$ weight matrix to get final embeddings for each node, and $\mathbf{a}_u^{(t)}$ is an intermediary vector used in between aggregation and the $\textsc{combine}(\cdot)$ function [15].

With this formalisation out of the way, the graph-level representation of each triple-induced subgraph is made by concatenating the mean of the node embeddings, the node embedding of the head, the node embedding of the tail, and the embedding of the relation r. This final concatenated vector is then multiplied by a final weight matrix to get the score of the graph. Both the node-level and graph-level parts of this calculation can be seen in Figure 2.2.

As we can see, the GraIL model is relatively sophisticated. The model outperforms most other inductive models on knowledge graph tasks and has proven itself to be quite robust. The biggest downside of the GraIL model is that it needs to induce a subgraph for each triple it scores. This means that scalability is out of the question when it comes to large unseen test sets [16]. Rather, the GraIL model is best used in cases when a small set of poignant triples are to be tested.

2.3 The INDIGO Model

Another model that has approached this problem is the INDIGO model. Similarly to the GraIL model, the INDIGO model is an inductive model that induces a graph which it then uses a GNN on to score triples [16]. Unlike the GraIL model, however, the INDIGO model is not limited to scoring triples one at a time, instead, the graph the INDIGO model induces captures the structure of multiple triples at once.

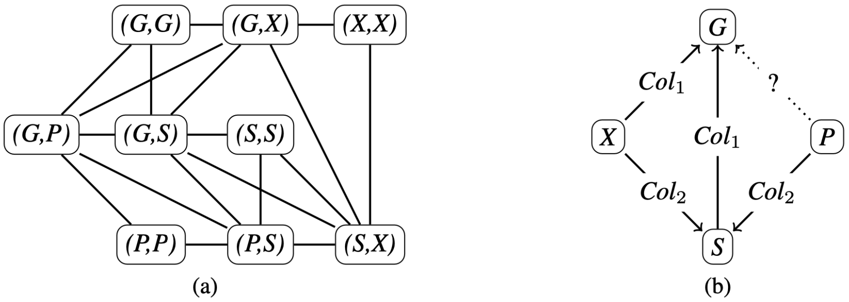


Figure 2.3: A visualisation of how INDIGO models work compared to R-GCN models. (From [16])

The INDIGO model is set up by first providing a knowledge graph along with a set Λ of candidate triples. A graph is then constructed such that each node in the graph corresponds to a triple found in either the knowledge graph or Λ (where the two entities can be the same). The order of the nodes in the pair does not matter and there are no two nodes sharing the same pair. Nodes in this graph are connected if their pairs share an entity, for example, the nodes $u_{(c,d)}$ and $u_{(d,g)}$ would be connected. The nodes have integer features such that, given a triple in the knowledge graph but not in Λ, each feature corresponds to a relation in the knowledge graph. The features of such nodes are 1 if there is a triple with that relation and those nodes, -1 if there is an inverted triple with that relation, and 0 otherwise. Nodes representing triples in Λ have features made up of only zeros. An example of this type of graph is shown in Figure 2.3.

Once this graph is constructed, a simple GCN is run on the graph to calculate embeddings for all the nodes. This model is trained such that, to predict whether or not a candidate triple in Λ should be in the knowledge graph, we simply need to check if any of the features in that node's final embedding are above some threshold. The original paper uses a threshold of 0.5, which was effective for the tasks set out in said paper [16]. If feature i was above this threshold, then the model would predict that the two entities of the node should be connected with relation I.

From a scalability perspective, the INDIGO model has an advantage over the GraIL model because it is able to make predictions over a bulk set of candidate triples using one induced graph rather than having to create a new graph for each triple. Despite this, the INDIGO model is still not able to easily handle live learning environments where new candidates come to the model in a stream because the model is limited to only working on entity pairings present in its originally induced graph. While the INDIGO model is therefore able to perform relatively well with inductive tasks, using the model in real-world settings would still likely necessitate that it be completely re-trained with each new batch of incoming candidate triples [60].

2.4 The NBFNet Model

Finally, we will also look at the the Neural Bellman-Ford Network (NBFNet) model [17]. This model is slightly similar to the GraIL model in that it uses information from paths between two nodes in a triple to score that triple. Instead of inducing a subgraph, the NBFNet model works by embedding a triple as a sum of the representations of each of the paths that exist between the two nodes of the triple. The representations for each of these paths, meanwhile, is a product of the representations of its parts.

Instead of calculating these values directly, the model uses a modified version of the generalised Bellman-Ford algorithm. Using this strategy, the representations of the triples are initialised as being all zeros except for the case where the head and tail entities are the same. The model then iteratively uses message-passing to calculate path summations between nodes for increasingly longer paths, essentially converging on the representations that one would arrive at had they calculated the path summations directly. This iterative process is detailed below:

$$\mathbf{h}_u^{(0)} = \textsc{indicator}(u, r, d)$$

$$\mathbf{h}_u^{(t)} = \textsc{aggregate}\left(\{\textsc{message}(\mathbf{h}_s^{(t-1)}, \mathbf{w}_d(s, r, u)) | (s, r, u) \in E(u)\} \bigcup \{\mathbf{h}_u^{(0)}\}\right)$$

where $E(u)$ is the set of edges connected to entity u, $\mathbf{w}_d(s, r, u)$ is the weight of the representation of triple (s, r, u), $\textsc{indicator}(x, r, y)$ is a function that returns a vector of ones if the two entities in the triple (x, r, y) are equal and a vector of zeros otherwise, and $\textsc{message}(\cdot)$ is a generic GNN message-passing function used to define possible reformatting operations on messages before aggregation [17].

The NBFNet model is moderately lightweight but is still able to perform competitively against the previously discussed models in an inductive learning environment. The model also has the advantage of having prediction scores which are direct measurements of the importance of various possible paths when making predictions. This makes the model more interpretable and could allow it to be used in other contexts that involve the analysis of paths in a knowledge graph [17].

3

Proposed Approach

Contents

Having discussed other inductive knowledge graph models, we now introduce our own model for tackling problems in this space: the GraphRE model.

3.1 Justification and Derivation

The goal of this project is to derive a new neural model with a strong inductive bias that is capable of operating on relational graphs, specifically knowledge graphs, in an inductive setting. Ideally, such a model should be scalable, operate directly on the knowledge graph, and require very little pre-processing. We will now go through step by step how our model can be derived logically by breaking down parts of the problem.

To start, the reason for our interest in GNNs is their ability to operate in *inductive learning environments.* Despite this, as we mentioned earlier in Section 1.2.5, GNNs do not exhibit a strong *relational inductive bias* on relational graphs (we explored the distinction between relational inductive bias and an inductive learning environment in Section 1.2.5). In other words, they are unable to learn and account for various inherent phenomena that are typically found in relational graphs such as relational symmetries, hierarchies, and the aforementioned rule inference patterns. This limitation stops them from reaching their full potential when being used on graphs where relational bias accounts for much of the graphs' underlying structure.

In the case of knowledge graphs, this relational inductive bias is crucial. Compared with other types of relational graphs, knowledge graphs have an especially strong inductive bias on account of the fact that edges in knowledge graphs correspond to principled, factual connections between real-world entities [42]. Given this characteristic, if we are to fully realise the potential of GNNs in the knowledge graph space, it will necessarily require us to inject an inductive capability into our models.

On this note, shallow embedding models *do* have this strong inductive bias. They are adept at learning things like inference patterns and symmetries by design. Their only major flaw for the purposes of this thesis is that, while they do exhibit a strong *inductive bias*, they are unable to complete tasks in an *inductive setting*. Given this information, we deducted that it might be possible to combine shallow embedding models and GNNs to produce a new type of model that has the inductive bias of a shallow embedding with the ability to operate in an inductive environment like a GNN.

The first problem to address is developing a hybrid approach that reaps the benefits of both shallow embedding models and GNNs. Shallow embedding models work directly with entity embeddings, and since, as we mentioned earlier, we want our model to ideally operate directly on knowledge graphs instead of using a secondary graph, it is reasonable for us to use shallow embedding models in such a way as to operate on the node representations used in the GNN, which are the graph analogues of the entity embeddings in shallow embedding models. Naturally, this means we would need to integrate the shallow embedding model with the message-passing portion of the GNN. To this end, we opt to use the shallow embedding

Algorithm 1 The GraphRE model

```
Require: Knowledge graph $K$ with $V$ entities, run for $T$ layers
  $\mathbf{h}_v^{(0)} \leftarrow x_v, \forall v \in V$
  for $t = 1...T$ do
      for $v \in V$ do
          $\mathbf{h}_{N_{KG}(v)}^{(t)} \leftarrow \textsc{aggregate}(\Phi_r^{\pm}(\mathbf{h}_v^{(t-1)}, \mathbf{h}_u^{(t-1)}), \forall u \in N_{KG}(v))$
          $\mathbf{h}_v^{(t)} \leftarrow \sigma(\mathbf{W}^{(t)} \cdot \textsc{merge}(\mathbf{h}_v^{(t-1)}, \mathbf{h}_{N_{KG}(v)}^{(t)}))$
      end for
      $\mathbf{h}_v^{(t)} \leftarrow \textsc{normalise}(\mathbf{h}_v^{(t)}), \forall v \in V$
  end for
  $\mathbf{z}_v \leftarrow \mathbf{h}_v^{(T)}, \forall v \in V$
```

models to transform the messages of the GNN right before they are passed along the edges of the graph. By using this message-passing scheme, we hope to more directly inject the structural properties of the underlying graph into the learning process.

To illustrate this process further, we now formalise some of these ideas in mathematical notation. Let us assume we have a knowledge graph K. Given a specific entity c in K, we can define its neighbourhood $N_{KG}(c)$ as:

$$N_{KG}(c) = \{d \mid \exists r \; s.t. \; (c, r, d) \in K \vee (d, r, c) \in K\}$$

An important thing to note about this neighbourhood is that it includes nodes connected to c in both directions. We can then envision a message-passing GNN that calculates a new representation for c by merging its previous value with an aggregation of the nodes in $N_{KG}(c)$. We could then augment these neighbourhoods by using a shallow embedding model to define semantically meaningful messages to propagate within the neighbourhoods. In this case, the shallow embedding model is transforming the values of each of the representations of the nodes in the $N_{KG}(c)$ before the aggregation step is done. The details of this shallow embedding transformation for a particular node d in $N_{KG}(c)$ would depend on which relation r connected c and d as well as which direction the relation is in. Intuitively we can see that such a scheme might have the potential of allowing the GNN to learn how the different relations in K interact with one another. We will expand on the details of how such a model can be actually implemented in the section below.

3.2 Proposed Method: the GraphRE Model

We now formally introduce the GraphRE model, its functionality, and its main properties.

The GraphRE model, short for graph relational embedding model, functions in a manner akin to the process introduced in the previous section. We define it formally below:

Definition 3.2.1 (GraphRE Model)**.** Given a preset shallow embedding model function $\Phi_r^{\pm}(\cdot)$ to transform the messages before they are passed along, the calculation of the representation of a node v under the GraphRE model is defined as follows:

$$
\begin{aligned}
\mathbf{h}_{N_{KG}(v)}^{(t)} &= \textsc{aggregate}(\{\Phi_r^{\pm}(\mathbf{h}_v^{(t-1)}, \mathbf{h}_u^{(t-1)})\} | \forall u \in N_{KG}(v)) \\
\mathbf{h}_v^{(t)} &= \sigma(\mathbf{W}^{(t)} \cdot \textsc{merge}(\mathbf{h}_v^{(t-1)}, \mathbf{h}_{N_{KG}(v)}^{(t)}))
\end{aligned}
$$

where $\textsc{aggregate}(\cdot)$ is defined in the same way as in Equation 1.1, $\textsc{merge}(\cdot)$ is a differentiable function combining $\mathbf{h}_v^{(t-1)}$ and $\mathbf{h}_{N_{KG}(v)}^{(t)}$, and $\mathbf{W}^{(t)}$ is a learnable weight matrix.

As we can see, given a knowledge graph K, the GraphRE model operates directly on the underlying knowledge graph. That is, we do not induce any secondary graph through which messages are passed, they are instead passed along the real triples in K.

As we previously discussed, the representation of any given node c is calculated by merging its previous representation with an aggregation of representations of nodes in $N_{KG}c)$ that have been transformed by a shallow embedding model. The transformation of a particular node d in $N_{KG}c)$ can be generalised by a function $\Phi_r^{\pm}(\cdot)$ where r is the specific relation between the central node and the node being transformed and ± determines whether the tail of the relation is c (the + case) or d (the − case). We can see how this works visually for a specific example in Figure 3.1. In this example, the representation of the central node c's neighbourhood at

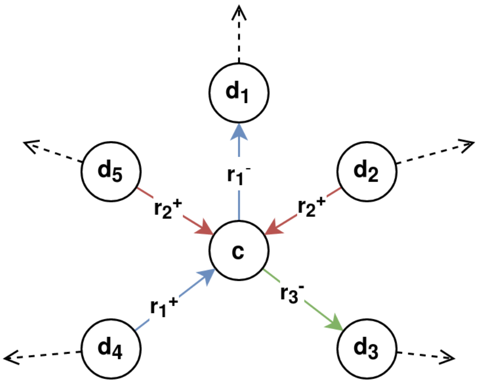


Figure 3.1: A diagram showing how the GraphRE model performs message-passing. Here the central entity c is connected to 5 other entities through a total of 3 different relations. The $+$ or $-$ above the relation label indicates what direction the relation is with respect to the central node, which will inform how the shallow embedding model transforms the message passing along that edge.

a particular layer t of the GraphRE model would be:

$$\mathbf{h}^{(t)}_{N_{KG}(c)} = \\ \text{AGGREGATE}(\{\Phi^{-}_{r_1}(\mathbf{h}^{(t-1)}_{d_1}), \Phi^{+}_{r_2}(\mathbf{h}^{(t-1)}_{d_2}), \Phi^{-}_{r_3}(\mathbf{h}^{(t-1)}_{d_3}), \Phi^{+}_{r_1}(\mathbf{h}^{(t-1)}_{d_4}), \Phi^{+}_{r_2}(\mathbf{h}^{(t-1)}_{d_5})\})$$

and the representation of the central node itself would then be:

$$\mathbf{h}^{(t)}_{c} = \sigma(\mathbf{W}^{(t)} \cdot \text{MERGE}(\mathbf{h}^{(t-1)}_{c}, \mathbf{h}^{(t)}_{N_{KG}(c)}))$$

We can combine all of these concepts together and abstract some of the implementation details to get to the main algorithm used by the GraphRE model, which is presented in Definition 3.2.1 and Algorithm 1.

At the heart of this algorithm is the $\Phi^{\pm}_{r}(\cdot)$ function. We can define this function in a variety of different ways depending on which shallow embedding model we are merging with our GNN. To do this, we simply manipulate the scoring function of those shallow embedding models to arrive at an expression that is meant to capture what role the central node c is playing in the scoring function of that shallow embedding model. To illustrate this point, we have listed various equations below which are used for different types of shallow embedding models in the case where c is the central node and d is one of the nodes in $N_{KG}c)$ whose representation is being transformed before the aggregation procedure:

- **TransE$^+$** $(c \leftarrow d)$:
  $\Phi_r^+(\mathbf{h}_c, \mathbf{h}_d) = \mathbf{h}_d + \mathbf{r}$

- **TransE$^-$** $(c \rightarrow d)$:
  $\Phi_r^-(\mathbf{h}_c, \mathbf{h}_d) = \mathbf{h}_d - \mathbf{r}$

- **DistMult$^+$** $(c \leftarrow d)$:
  $\Phi_r^+(\mathbf{h}_c, \mathbf{h}_d) = \mathbf{h}_d \odot \mathbf{r}$

- **DistMult$^-$** $(c \rightarrow d)$:
  $\Phi_r^-(\mathbf{h}_c, \mathbf{h}_d) = \mathbf{h}_d \odot \mathbf{r}$

- **RESCAL$^+$** $(c \leftarrow d)$:
  $\Phi_r^+(\mathbf{h}_c, \mathbf{h}_d) = \mathbf{h}_d\mathbf{R}$

- **RESCAL$^-$** $(c \rightarrow d)$:
  $\Phi_r^-(\mathbf{h}_c, \mathbf{h}_d) = (\mathbf{R}\mathbf{h}_d^T)^T$

- **ComplEx$^+$** $(c \leftarrow d)$:
  $\Phi_r^+(\mathbf{h}_c, \mathbf{h}_d) = \Re(\mathbf{h}_d) \odot (\Re(\mathbf{r}) + \Im(\mathbf{r})) + \Im(\mathbf{h}_d) \odot (\Re(\mathbf{r}) - \Im(\mathbf{r}))$

- **ComplEx$^-$** $(c \rightarrow d)$:
  $\Phi_r^-(\mathbf{h}_c, \mathbf{h}_d) = \Re(\mathbf{h}_d) \odot (\Re(\mathbf{r}) - \Im(\mathbf{r})) + \Im(\mathbf{h}_d) \odot (\Re(\mathbf{r}) + \Im(\mathbf{r}))$

Where $\odot$ is the element-wise multiplication operator, $\Re(\mathbf{z})$ returns the real part of $\mathbf{z}$, and $\Im(\mathbf{z})$ returns the imaginary part of $\mathbf{z}$.

3.3 Properties of the GraphRE Model

The AGGREGATE$(\cdot)$ function can be defined in the same way as it is in regular MPNNs, but with a few limitations. We will explore this in more detail in Section 5.1, but intuitively these limitations comes from the fact that knowledge graphs are able to contain many different *types* of entities at the same time, which is one of the main reasons inductive learning on knowledge graphs has been difficult to do in the past. The GraphRE model can be seen as a way of tackling this issue by using shallow embedding models as a means of transitioning the representations of entities in $N_{KG}c$) into the representational space of the central node c through the relation r between those two nodes. This process, however, does not survive aggregation through summation because it forces the aggregated vector out of c's representational space.

The MERGE$(\cdot)$ function also has a variety of possible implementations. One way to generalise the function and optimise over many different implementations is to simply add a self-loop relation over every entity that corresponds to a new relation r_s. The model then does message-passing over these new relations in tandem with the relations already existing in the graph, ensuring that the latest representations of the central nodes of each neighbourhood are merged with the aggregated representation of the neighbours. For certain embedding model choices, these self-loops can simplify further, making the merge easy to calculate and scalable. We explore this further in Section 5.2.

Finally, we will look at another relevant property of the GraphRE model concerning the effect that the dimensionality of its entity features and node representations have on its performance. In most pure shallow embedding models there is a clear performance improvement that comes from increasing the size of the entity representations coming from the fact that these larger vectors are able to store more information compared to smaller models [25]. This correlation does not carry over empirically in existing message-passing GNN models that attempt to tackle inductive knowledge completion tasks [61]. Instead, the performance of the models remains quite consistent given a reasonable range of dimensions [15]. The GraphRE model, however, does exhibit an improvement in performance when increasing the dimensionality of its node representations up to a certain degree. This makes sense given both the model's reliance on shallow embedding models, which we expect would benefit from working in higher dimensions, and the model's GNN component, which we expect would perform worse as the model starts to overfit. We take a deeper look at these ideas in Section 5.3.

4

Results

Contents

We now look at what experiments we ran to test the merits of our model, what the results of those experiments were, and why those results are relevant.

4.1 Datasets

In the experiments we describe below, we use data from a total of three different knowledge graph datasets organised in various ways. These datasets are WordNet18RR, Freebase15k-237, and NELL. As we will explain in more detail in the next section, we have broadly split our experiments into two sets: transductive ones and inductive ones. While some of the experiments in these sets may use data from the same dataset, the way that data is organised is crucial to the nature of those experiments.

4.1.1 WordNet18RR

As we previously discussed, this dataset is subset of WordNet, a graph consisting of lexical relations between words [2]. We use this dataset in our transductive and inductive experiments.

In the transductive setting, we use the entirety of the WN18RR dataset and partition it into a training, validation, and testing set with the goal of training models for the task of link prediction. The split we use in this case is the standard split used in the original TransE paper as well as all further shallow embedding papers since then [1]. These splits are purposefully designed to be transductive, with entities in the testing set also appearing in the training set. The total dataset has 40,943 entities, 11 relations, and 93,003 triples. The training, validation, and testing splits each have 86,835, 3,034, and 3,034 triples respectively.

In the inductive setting, we do not use the full WN18RR dataset. Instead, we use a series of subsets of WN18RR originally introduced in the GraIL paper to allow the knowledge graph to be tested in an inductive setting. This was done by running a randomised algorithm that partitions small subsections of the WN18RR graph into training, validation, and testing sets such that there is no leakage between the sets, fulfilling the requirements needed for the experiment to be labelled inductive [15]. This randomised algorithm was run multiple times to create four progressively larger inductive subsets of WN18RR, which are labeled **v1_ind**, **v2_ind**, **v3_ind**, and **v4_ind** in the WN18RR table. The GraIL paper used a slightly modified version of this same randomised algorithm to create four progressively larger transductive subsets of the WN18RR dataset, each of which are labelled **v1**, **v2**, **v3**, and **v4**. We have run our models on these datasets as well and have included those results in the inductive results table to clearly contrast comparable transductive and inductive results. The details of the sizes of each of these smaller datasets can be found in Appendix A.

4.1.2 Freebase15k-237

As we have also previously discussed, this dataset is a subset of Freebase, a graph which attempts to organise general human knowledge [4]. We use this dataset in our transductive and inductive experiments.

In the transductive setting, we use the entirety of the FB15k-237 dataset and partition it into a training, validation, and testing set with the goal of training models for the task of link prediction. The split we use in this case is, just like WN18RR, the standard split used in the original TransE paper as well as all further shallow embedding papers since then [1]. These splits are purposefully designed to be transductive, with entities in the testing set also appearing in the training set. The total dataset has 14,541 entities, 237 relations, and 310,116 triples. The training, validation, and testing splits each have 272,115, 17,535, and 20,466 triples respectively.

In the inductive setting, we do not use the full FB15k-237 dataset. Instead, we use a series of subsets of FB15k-237 originally introduced in the GraIL paper to allow the knowledge graph to be tested in an inductive setting. This was done by running the same randomised algorithm used for WN18RR to partition small subsections of the FB15k-237 graph into training, validation, and testing sets such that there is no leakage between the sets, fulfilling the requirements needed for the experiment to be labelled inductive [15]. The randomised algorithm was run multiple times to create four progressively larger inductive subsets of FB15k-237, which are labeled **v1_ind**, **v2_ind**, **v3_ind**, and **v4_ind** in the FB15k-237 table. The GraIL paper used a slightly modified version of the randomised algorithm to also create four progressively larger transductive subsets of the FB15k-237 dataset, each of which are labelled **v1**, **v2**, **v3**, and **v4**. We have run our models on these datasets as well and have included those results in the inductive results table to clearly contrast comparable transductive and inductive results. The details of the sizes of each of these smaller datasets can be found in Appendix A.

4.1.3 NELL

As we have again previously discussed, this dataset is a subset of the Never-Ending Language Learning project, a graph made by continuously scraping data from public webpages [6]. We use this dataset only in our inductive experiments.

In the inductive setting, we do not use the full NELL dataset. Instead, we use a series of subsets of NELL originally introduced in the GraIL paper to allow the knowledge graph to be tested in an inductive setting. This was done by running the same randomised algorithm used for WN18RR and FB15k-237 to partition small subsections of the NELL graph into training, validation, and testing sets such that there is no leakage between the sets, fulfilling the requirements needed for the experiment to be labelled inductive [15]. The randomised algorithm was run multiple times to create four progressively larger inductive subsets of NELL, which are labeled **v1_ind**, **v2_ind**, **v3_ind**, and **v4_ind** in the NELL table. The GraIL paper used a slightly modified version of the randomised algorithm to also create four progressively larger transductive subsets of the NELL dataset, each of which are labelled **v1**, **v2**, **v3**, and **v4**. We have run our models on these datasets as well and have included those results in the inductive results table to clearly contrast comparable transductive and inductive results. The details of the sizes of each of these smaller datasets can be found in Appendix A.

4.2 Experimental Setup

For the purpose of this thesis, we ran two main sets of experiments: transductive ones aimed at comparing the GraphRE model with pure shallow embedding models and inductive ones pitting the GraphRE model against other inductive knowledge completion models.

In the transductive experiments, we compare the performance of the GraphRE model against pure shallow embedding models to see what effect adding a message-passing component has on the shallow model. We chose to compare the GraphRE model to a set of shallow models that we feel adequately represents a wider range of shallow models with regards to complexity. The TransE and RotatE models are both relatively simple yet powerful and should sufficiently cover what can be expected from translational models. Meanwhile, we chose DistMult and ComplEx to see how bilinear models would fare. DistMult has the benefit of being a relatively lightweight bilinear model, while ComplEx is able to benefit from not having a symmetrical bias while still being much more scalable than other bilinear models like RESCAL.

These shallow embedding models were compared with two versions of the GraphRE model: GraphRE-TransE and GraphRE-ComplEx. These two variations differ in what type of shallow embedding procedure is used to transform the node embeddings during the message-passing process. GraphRE-TransE learns an embedding for each relation that is simply added to node embeddings as they are passed along as messages, while GraphRE-ComplEx learns a complex embedding for each relation that transforms messages in a way that corresponds to how triples are scored in a ComplEx shallow model. Both of these procedures are outlined in Section 3.2. By choosing TransE and ComplEx, we can see how the GraphRE model reacts in both bilinear and translational settings.

In the inductive experiments, we similarly compare the performance of both GraphRE-TransE and GraphRE-ComplEx with a variety of other inductive knowledge completion models. Specifically, we compare GraphRE with GraIL, RuleN, DRUM, Neural-LP, R-GCN, NBFNet, and INDIGO. There is not as much consistency in this space relative to the transductive setting when it comes to models, metrics, and datasets, so we have tried our best to aggregate the results that are available in the best way we can to make the comparison fair. In this regard, the tables for the WN18RR and FB15k-237 datasets do have the privilege of having more models to compare with, which shows how much more those datasets have been utilised in this field. We have chosen to compare these models with GraphRE-TransE and GraphRE-ComplEx for the same reasons as previously stated in the transductive case.

4.2.1 Evaluation Metrics

To evaluate the performance of the GraphRE model in these different configurations and settings, we have opted to use the same metrics used by other, similar papers focusing on knowledge graph completion tasks. The main metrics we used for our experiments were mean reciprocal rank and Hits@10 for the transductive experiments and area under the curve (precision-recall) and Hits@10 for the inductive experiments.

Mean Reciprocal Rank

The first metric that we use to evaluate the performance of our models is mean reciprocal rank (MRR). This metric is used in tasks where models are challenged to discern real candidates from a large body of negative samples. It is calculated by first having the model sort a list of potential candidates in order of how likely they are to appear in the data. Each candidate is then given a rank according to this list. The candidates representing true facts have their ranks inverted, and these values are then averaged across runs to give us the final MRR score.

In the context of knowledge completion, this body of negative samples consists of a filtered list of perturbed facts based on a real one. More specifically, given a real fact (c, r, d), we can create two sets of negative facts: $(c, r, d'), \forall d' \neq d \in E$ and $(c', r, d), \forall c' \neq c \in E$, where E is the set of all entities in the knowledge graph and both sets are filtered so that they do not contain any true facts. We can now simply rank where (c, r, d) sits in comparison to both of these sets, calculate the inverse of those ranks, then average those numbers for all true facts in the test set of our dataset. This calculation can be formalised below:

$$MRR = \frac{1}{2|E_{test}|} \sum_{(c,r,d) \in E_{test}} \frac{1}{\textsc{rank}((c, r, d)|(c, r, d'))} + \frac{1}{\textsc{rank}((c, r, d)|(c', r, d))}$$

where E_{test} is the test set of entities in our dataset and $\textsc{rank}((c, r, d)|(c, r, d'))$ is the rank of the true fact (c, r, d) when compared with the set of filtered and perturbed negative sampled defined by the set involving (c, r, d') described above (with an analogous definition for $\textsc{rank}((c, r, d)|(c', r, d))$).

MRR can be a useful metric in that it is able to show well models can do given suitable conditions, this is because individually well-scoring facts can sway the MRR score quite heavily. We use this metric in our transductive experiments because it is one of the primary metrics used by pure shallow embedding model papers.

Hits@k

The next metric we use is Hits@k. This metric is also used in tasks that have models score how likely certain candidates are among a large body of negative samples. Calculating this metric is quite simple. Once a given set of true facts and each of their associated negative samples have been scored, the true facts are ranked based on their positions in their sorted lists. Each true fact is then queried to check whether its rank in is less than or equal to k, and the final Hits@k metric measures the proportion of true facts that meet this criteria.

The choice of negative samples to compare with the true fact is done differently in various places of the literature depending on which models are being compared with each other and in which setting. In our transductive experiments, we employ the negative sampling procedure utilised by the pure shallow embedding papers where the negative samples are simply the same set of negative samples used in the MRR metric calculation described above. Using these same definitions of (c, r, d') and (c', r, d), we can formalise this version of Hits@k in the following way:

$$Hits@k = \frac{1}{2|E_{test}|} \sum_{(c,r,d) \in E_{test}} [\mathbf{1}(\textsc{rank}((c,r,d)|(c,r,d')) \leq k) \\ + \mathbf{1}(\textsc{rank}((c,r,d)|(c',r,d)) \leq k)]$$

where $\mathbf{1}(g)$ is the indicator function that returns 1 if g is true and 0 otherwise.

In our inductive experiments, we follow the lead set by the GraIL paper and followed later by other papers introducing new inductive knowledge completion models by simply using a filtered set of 50 random candidate facts as the body of negative samples for each true fact. This slight variation can be written in the following way:

$$Hits@k = \frac{1}{|E_{test}|} \sum_{(c,r,d) \in E_{test}} \mathbf{1}(\textsc{rank}((c,r,d)|\textsc{random}(50)) \leq k)$$

where RANDOM(n) is a function that returns n random filtered negative samples.

Area Under the Curve (Precision-Recall)

The last metric we use is the area under the curve (precision-recall), also referred to as AUC-PR. This metric is a useful way of measuring how well a binary classifier performs without suffering from all the drawbacks of a raw accuracy score. It is calculated my measuring the area under the curve of the chart plotting precision on one axis and recall on the other.

Precision here is defined as being the proportion of positive guesses that are actually correct. In the context of knowledge graphs, this corresponds to the proportion of triples the model predicted to be in the graph that are actually in the test set. Recall, meanwhile, is the proportion of real positives that were actually identified by the model. In the context of knowledge graphs this corresponds to the proportion of the total test set of real triples that the model was actually able to predict were in the graph. Both precision and recall are values that are bounded between 0 and 1, so the area below any curve plotted between them must similarly be bounded by 0 and 1, making AUC-PR easy to read and good for comparisons.

All together AUC-PR has the benefit of being relatively comprehensible, accounting for things like class imbalances, which raw accuracy does not handle well, and focusing its attention on how well models deal with positive examples rather than negative ones. These factors are why this metric has been used in previous literature regarding inductive knowledge completion tasks and why we also use it in our inductive experiments. Following the example of previous papers, one of the tasks we run in our inductive experiments is to challenge our model to discern between real triples and random negative samples, with a sampling ratio of 50 negative examples for every real fact, and measure the performance of this classification using AUC-PR.

4.2.2 Implementation Details and Hyperparemeters

We now discuss what parameters were used to tune and optimise our models. Our models were all tuned to use the most efficient hyperparameters for each given dataset.

Our GraphRE models consist of 3 GraphRE layers stacked back to back. Each GraphRE layer is implemented based on Algorithm 1, the main procedure introduced previously.

The $\Phi_r^{\pm}(\cdot)$ function described in the algorithm is implemented for the GraphRE-TransE and GraphRE-ComplEx cases in the same way as it is described for the TransE and ComplEx cases respectively in the list of possible implementations found in Section 3.2.

The AGGREGATE$(\cdot)$ function described in the algorithm is implemented as a mean aggregator which calculates the average of all the transformed node representations in the neighbourhood of any given central node. The reasons for choosing this type of aggregator specifically are explained in Section 5.1.

The MERGE$(\cdot)$ function described in the algorithm is addressed by simply adding self-loops to the knowledge graph mediated through a new relation. More specifically, given a knowledge graph K, a new relation r_s is added to K along with triples (c, r_s, c) for every entity c in K. This new setup lets the GraphRE model learn how to merge neighbourhood and central node representations on its own and removes the need for the MERGE$(\cdot)$ function to have an explicit implementation beyond just returning $h^t_{N(v)}$ (which, with this self-loop construction, now includes information about h^{t-1}_v). More details about this setup and its properties can be found in Section 5.2.

The NORMALISE$(\cdot)$ function in the algorithm is simply a ReLU function in our implementation.

When it comes to setting the dimensionality of the GraphRE model, one of the interesting things we noticed is that changing the dimensionality of our entity embeddings had a significant impact on the performance of the model, with the optimal dimensionality values being larger than what would normally be expected of a GNN. This makes sense because we are performing message-passing using the embeddings generated by shallow embedding models, which in turn tend to perform better as the number of entity dimensions increases. From our investigations, this dimensional property does not similarly hold for other inductive neural knowledge completion models like GraIL that perform message-passing on constructed graphs. With all this in mind, we set gave our model 128 dimensions in the transductive setting and set the dimensionality optimally in the inductive case depending on the dataset the model was running on and which shallow embedding model was integrated into GraphRE. The details of what the specific dimension settings were in the inductive case can be found by looking at the best performing values in Table 5.4, Table 5.5, and Table 5.6. More details about this property are discussed in Section 5.3.

For our experiments, we used a learning rate of 9e-4 in the transductive case and 1.5e-3 in the inductive case. For both cases we set our weight decay to be 5e-4.

All our experiments used early-stopping to determine how many epochs to run for. Our setup makes the model run for at least 30 epochs before waiting until no improvements occur over the validation set for 80 epochs. Each result is an average over 5 runs.

In training, we generate negative samples with a ratio of 50 negative samples per real fact. This ratio is the same one used by other previous papers in the field.

4.3 Experimental Results

We now present the results of the previously described experiments.

4.3.1 Transductive Experiments

		WordNet18RR	**Freebase15k-237**
MRR	TransE [1]	22.60	**29.40**
	ComplEx [10]	**44.00**	24.70
	GraphRE-TransE	29.45	27.35
	GraphRE-ComplEx	37.87	22.26
Hits@10	TransE [1]	50.10	**45.60**
	ComplEx [10]	**51.00**	42.80
	GraphRE-TransE	50.64	42.89
	GraphRE-ComplEx	50.97	41.95

Table 4.1: Results from transductive experiments.

First, we ran transductive experiments to see how well the GraphRE model would stack up against various shallow embedding models on commonly used datasets. The results of those transductive experiments can be seen in Table 4.1.

As we can see, the GraphRE-TransE model outperforms the pure TransE model on WN18RR, but does slightly worse on FB15k-237. Meanwhile, the GraphRE-ComplEx model does slightly worse than the pure ComplEx model on both WN18RR and FB15k-237. Overall, the GraphRE model appears to perform about as well as the pure shallow embedding models in the transductive setting. This is not too surprising considering the fact that shallow models have the advantage of adapting completely to whatever dataset they are operating on in a transductive setting. Since they are not designed to function in inductive settings, shallow embedding models do not have as much pressure to generalise what they learn. On the other hand, the GraphRE model adds a GNN component on top of its shallow model, making it slower to converge in a transductive setting but allowing it to generalise better.

4.3.2 Inductive Experiments

		NELL							
		v1	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
AUC-PR	GraIL [15]	83.95	92.73	92.30	89.29	86.05	92.62	93.34	87.50
	RuleN [15]	80.16	87.87	86.89	84.45	84.99	88.40	87.20	80.52
	DRUM [15]	N/A	N/A	N/A	N/A	59.86	83.99	87.71	85.94
	Neural-LP [15]	N/A	N/A	N/A	N/A	64.66	83.61	87.58	85.69
	R-GCN [16]	N/A	N/A	N/A	N/A	74.50	50.40	52.00	51.00
	INDIGO [16]	N/A	N/A	N/A	N/A	94.50	92.50	95.10	92.90
	GraphRE-TransE	**94.87**	**98.00**	**97.80**	97.49	**99.34**	95.22	**96.07**	**97.32**
	GraphRE-ComplEx	93.73	97.61	97.59	**97.62**	99.28	**96.58**	95.99	96.99
Hits@10	GraIL [15]	64.08	86.88	84.19	82.33	59.5	93.25	91.41	73.19
	RuleN [15]	62.82	82.82	80.72	58.84	53.5	81.75	77.26	61.35
	DRUM [15]	N/A	N/A	N/A	N/A	19.42	78.55	82.71	80.58
	Neural-LP [15]	N/A	N/A	N/A	N/A	40.78	78.73	82.71	80.58
	GraphRE-TransE	**89.01**	95.95	**96.21**	**96.05**	**100.00**	92.12	91.52	**95.73**
	GraphRE-ComplEx	87.24	**97.42**	93.55	**96.05**	**100.00**	**94.70**	**94.38**	95.35

Table 4.2: Inductive results from **NELL**.

Next, we ran inductive experiments to test how well the GraphRE model performs against existing neural knowledge graph completions models that were

		WordNet18RR							
		v1	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
AUC-PR	GraIL [15]	**89.00**	**90.66**	88.61	**90.11**	**94.32**	**94.18**	85.80	92.72
	RuleN [15]	81.79	83.97	81.51	82.63	90.26	89.01	76.46	85.75
	DRUM [15]	N/A	N/A	N/A	N/A	86.02	84.05	63.20	82.06
	Neural-LP [15]	N/A	N/A	N/A	N/A	86.02	83.78	62.90	82.06
	R-GCN [16]	N/A	N/A	N/A	N/A	49.00	49.80	53.10	50.20
	INDIGO [16]	N/A	N/A	N/A	N/A	91.20	92.50	**92.40**	**94.70**
	GraphRE-TransE	87.93	88.22	88.85	87.23	87.66	89.21	87.23	88.06
	GraphRE-ComplEx	87.79	88.38	**89.98**	88.91	90.85	90.13	85.67	90.14
Hits@10	GraIL [15]	65.59	69.36	64.63	67.28	82.45	78.68	58.43	73.41
	RuleN [15]	63.42	68.09	63.05	65.55	80.85	78.23	53.39	71.59
	DRUM [15]	N/A	N/A	N/A	N/A	74.37	68.93	46.18	67.13
	Neural-LP [15]	N/A	N/A	N/A	N/A	74.37	68.93	46.18	67.13
	NBFNet [17]	N/A	N/A	N/A	N/A	**94.80**	**90.50**	**89.30**	**89.00**
	GraphRE-TransE	83.50	**86.43**	86.38	82.62	80.05	86.17	80.74	83.76
	GraphRE-ComplEx	**85.07**	84.24	**90.51**	**83.34**	86.70	89.17	80.17	85.99

Table 4.3: Inductive results from **WN18RR**.

		Freebase15k-237							
		v1	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
AUC-PR	GraIL [15]	88.97	**93.78**	**95.04**	**95.68**	84.69	90.57	91.68	94.46
	RuleN [15]	87.07	92.49	94.26	95.18	75.24	88.70	91.24	91.79
	DRUM [15]	N/A	N/A	N/A	N/A	69.71	76.44	74.03	76.20
	Neural-LP [15]	N/A	N/A	N/A	N/A	69.64	76.55	73.95	75.74
	R-GCN [16]	N/A	N/A	N/A	N/A	51.00	50.50	50.50	52.60
	INDIGO [16]	N/A	N/A	N/A	N/A	**93.40**	**96.30**	**96.60**	**95.80**
	GraphRE-TransE	**91.05**	92.16	91.94	91.59	86.58	90.74	89.37	90.19
	GraphRE-ComplEx	90.45	91.67	91.69	91.42	86.69	90.86	89.68	89.74
Hits@10	GraIL [15]	71.93	86.3	88.95	91.55	64.15	81.8	82.83	89.29
	RuleN [15]	67.53	88.00	**91.47**	**92.35**	49.76	77.82	87.69	85.6
	DRUM [15]	N/A	N/A	N/A	N/A	52.92	58.73	52.9	55.88
	Neural-LP [15]	N/A	N/A	N/A	N/A	52.92	58.94	52.9	55.88
	NBFNet [17]	N/A	N/A	N/A	N/A	**83.40**	**94.90**	**95.10**	**96.00**
	GraphRE-TransE	**86.03**	85.72	89.85	88.66	77.40	86.72	83.70	84.57
	GraphRE-ComplEx	82.16	**88.39**	88.41	85.60	71.60	82.85	82.25	84.43

Table 4.4: Inductive results from **FB15k-237**.

made to be used in an inductive learning environment. The results of those inductive experiments can be seen in Table 4.2, Table 4.3, and Table 4.4.

As we can see, the GraphRE model squarely outperformed every other model on the NELL dataset. Meanwhile, the model did a fair job on the transductive subgraphs of both WN18RR and FB15k-237, with the model lagging slightly behind the INDIGO and NBFNet models in the inductive setting but still tending to do fairly better than the GraIL model in most cases. These results show that the

GraphRE model is able to remain competitive with other models in an inductive setting despite the fact that those other models all do message-passing on tailor-made induced graphs and the GraphRE model does not. The NELL dataset further shows that, under certain conditions, the GraphRE model is even able to do much better than other inductive models despite its more lightweight nature.

4.4 Empirical Benefits

We now briefly go over some of the practical benefits of the GraphRE model that we noticed while running our experiments.

For starters, the GraphRE model is scalable and efficient even when faced with a large dataset. Since it does not need to construct a new graph to perform its message-passing on and instead just passes messages along real triples in the original knowledge graph, the GraphRE model has the benefit of getting results quickly. This property also means that this model has the potential to be used in unique settings like live learning where constructing a new graph to run message-passing on every time a new entity or triple is added to the knowledge graph is simply not scalable.

Another benefit of the GraphRE model is its adaptability. Since it is possible to change which shallow embedding model is integrated with the larger neural model, GraphRE has the potential to specialise better to certain tasks and applications than other models which are more rigid.

Finally, as we have shown with the data above, the GraphRE model gets these benefits while still being able to deliver results that perform competitively at a state-of-the-art level against other high-end models. The fact that the GraphRE model is more lightweight does not have a large enough effect on its performance to make it untenable on inductive knowledge completion tasks.

5
Discussion

Contents

We now discuss some of the theoretical properties of the GraphRE model and explore some small-scale experiments meant to hone in on some specific aspects of the model.

5.1 Aggregation Limitations

As we have previously discussed, the AGGREGATE$(\cdot)$ function in Algorithm 1 can be implemented in the same way for the GraphRE model as it can be for any GNN. In general, this is either a summation function, a max function, or a mean function. In previous literature, it has been shown that the summation aggregator has certain theoretical benefits over the other two methods, namely that it can encode more information about the neighbourhood of each node than the other two methods, which tend to give up information regarding the size of the neighbourhood they are encoding [13]. In the case of the GraphRE model, the goal of this aggregation is slightly different than for a classical GNN. Here, we can view the transformation done by the shallow embedding model within GraphRE as a way of transitioning from the representational space of a neighbouring node into that of the central node through the true triple found in the underlying knowledge graph connecting those two nodes. Given this interpretation, we can see intuitively why the summation aggregator does not accurately represent a prediction of the central node's representation, while the max and mean aggregators have more relevance.

With this mind, we can formulate an experiment for testing this property by simply changing the aggregator of the GraphRE model and seeing whether there is a clear difference between the sum, mean, and max aggregators.

5.1.1 Aggregation Experiments

We now present the results of comparing the GraphRE model with various different aggregator functions to show how much of a radical difference the choice of function makes in the case of the GraphRE model. These experiments have been set up in a manner similar to the experiments described in Section 4.

		NELL							
		v1	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
Sum	GraphRE-TransE	78.73	76.37	84.21	66.72	73.77	79.77	71.37	74.39
	GraphRE-ComplEx	61.82	65.87	62.02	59.56	61.63	72.54	65.35	60.89
Max	GraphRE-TransE	94.42	97.74	97.50	97.52	97.78	96.39	95.63	96.87
	GraphRE-ComplEx	93.74	97.93	97.59	96.81	74.71	96.12	95.87	95.72
Mean	GraphRE-TransE	**94.87**	**98.00**	**97.80**	97.49	**99.34**	95.22	**96.07**	**97.32**
	GraphRE-ComplEx	93.73	97.61	97.59	**97.62**	99.28	**96.58**	95.99	96.99

Table 5.1: AUC-PR results with different aggregators for **NELL**.

As we can see from Table 5.1, Table 5.2, and Table 5.3, it is clear that the mean aggregator tends to trump the other two aggregators for the most part. In the NELL dataset, the best performance values all come from the mean aggregator. In the WN18RR dataset, there are a few instances where the max aggregator fairs better, but for the most part the mean aggregator is still the best choice. The FB15k-237 dataset, however, breaks this trend. Here, the max aggregator does

		WordNet18RR							
		v1	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
Sum	GraphRE-TransE	83.58	83.89	84.03	85.34	82.30	89.58	64.87	85.97
	GraphRE-ComplEx	82.98	85.34	72.88	85.75	79.92	82.27	59.04	84.24
Max	GraphRE-TransE	86.05	88.21	84.31	87.57	90.33	**91.47**	82.54	88.19
	GraphRE-ComplEx	**90.25**	**92.36**	87.76	88.55	90.78	88.55	83.93	88.74
Mean	GraphRE-TransE	87.93	88.22	88.85	87.23	87.66	89.21	**87.23**	88.06
	GraphRE-ComplEx	87.79	88.38	**89.98**	**88.91**	**90.85**	90.13	85.67	**90.14**

Table 5.2: AUC-PR results with different aggregators for **WN18RR**

		Freebase15k-237							
		v1	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
Sum	GraphRE-TransE	78.24	84.65	73.43	84.45	76.34	79.77	76.99	76.74
	GraphRE-ComplEx	59.65	58.83	59.93	59.47	66.67	63.37	56.11	73.90
Max	GraphRE-TransE	90.93	**93.28**	92.19	93.10	85.89	88.89	**91.25**	89.44
	GraphRE-ComplEx	90.95	93.24	**93.52**	**93.48**	86.25	**91.89**	89.87	89.22
Mean	GraphRE-TransE	**91.05**	92.16	91.94	91.59	86.58	90.74	89.37	**90.19**
	GraphRE-ComplEx	90.45	91.67	91.69	91.42	**86.69**	90.86	89.68	89.74

Table 5.3: AUC-PR results with different aggregators for **FB15k-237**.

better more times than the mean aggregator. Despite this, one thing that is clear from this data is that the summation aggregator is never the best option in any case. As we have discussed, this makes sense given the fact that the shallow embedding model used in the GraphRE model is essentially transforming the representation of neighbouring nodes into the representational space of central nodes during aggregation. The summation aggregator breaks this transformation, which is why it tends to perform the worst. While both the mean and max aggregators do not break this condition, the mean aggregator lets neighbouring nodes each "vote" on a candidate representation for the central node while the max aggregator makes a single node speak for the entire neighbourhood. This behaviour on the part of the max aggregator puts a disproportionate amount of importance on individual triples when it comes to representing the central nodes, which is why the max aggregator tends to perform worse than the mean aggregator, although it is clear that in some datasets the max aggregator may be useful in cutting out noise.

5.2 Self-Loop Embeddings

We briefly explored the MERGE(·) function in Section 3.3 when we looked back at Algorithm 1. As we intuited earlier, this function can have many different implementations, each with their own benefits and drawbacks, which is why it is important for us to address this part of the GraphRE model in a way that fits with its other parts.

To that end, one way we might tackle the MERGE(·) function is to essentially side-step it entirely by having the merger of the representations of the node neighbourhoods with the representations of the central nodes take place during the aggregation step. We would ideally want the model to be able to differentiate between the central node and all of its neighbours during this process so that it can apply a unique set of learned parameters specifically for use on the representations of central nodes during aggregation. To achieve this, we can simply take advantage of the fact that, being in a knowledge graph setting, our dataset has edge types (in the form of relations). By adding a new relation r_s to the knowledge graph and adding a self-loop to each node that is of this new type, the GraphRE model can simply aggregate the representation of the central node of each neighbourhood using those self-loops and transform those central node representations using its shallow embedding model, which is guaranteed to incorporate a unique embedding reserved for relation r_s.

All together we can define the process formally in terms of triples. Given a knowledge graph K, as a part of the pre-processing phase of the training of a GraphRE model we simply add a new relation r_s to K as well as new triples (c, r_s, c) for each entity c in K. Given the variable in Algorithm 1, this change can be thought of as expanding our AGGREGATE(·) function in the following way:

$$\begin{aligned}\mathbf{h}_{N(v)}^{(t)} = \text{AGGREGATE}[&\Phi_{r_s}^{\pm}(\mathbf{h}_v^{(t-1)}, \mathbf{h}_v^{(t-1)}),\\ &\Phi_{r}^{\pm}(\mathbf{h}_v^{(t-1)}, \mathbf{h}_u^{(t-1)}), \forall u \in N_{KG}(v)]\end{aligned}$$

We then just simply define the MERGE(·) function like so:

$$\text{MERGE}(\mathbf{h}_v^{(t-1)}, \mathbf{h}_{N_{KG}(v)}^{(t)}) = \mathbf{h}_{N_{KG}(v)}^{(t)}$$

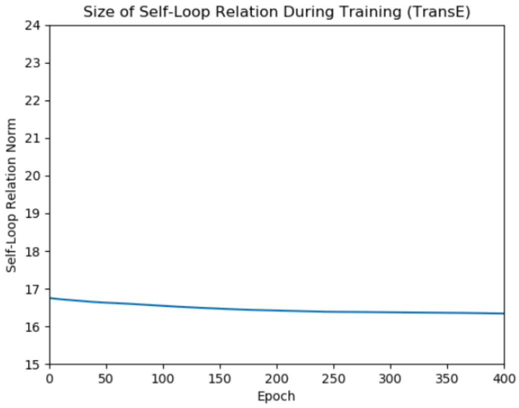


Figure 5.1: The evolution of the r_s relation embedding norm during training in the case of having a TransE model integrated with GraphRE and a ComplEx model integrated with GraphRE.

What is nice about this formulation is the way it has the capacity to simplify under certain conditions. For example, if the shallow embedding model integrated with the GraphRE model is TransE with a mean aggregator, the representation for a given central node c becomes the following:

$$\begin{aligned}\text{MERGE}(\mathbf{h}_c^{(t-1)}, \mathbf{h}_{N_{KG}(c)}^{(t)}) &= \frac{\mathbf{h}_c^{(t-1)} + \mathbf{r}_s + \mathbf{h}_c^{(t-1)} - \mathbf{r}_s + \sum_{u\in N_{KG}(c)} \Phi_r^{\pm}(\mathbf{h}_c^{(t-1)}, \mathbf{h}_u^{(t-1)})}{|N_{KG}(c)| + 2} \\ &= \frac{2\mathbf{h}_c^{(t-1)} + \sum_{u\in N_{KG}(c)} \Phi_r^{\pm}(\mathbf{h}_c^{(t-1)}, \mathbf{h}_u^{(t-1)})}{|N_{KG}(c)| + 2}\end{aligned}$$

To flesh out this concept, we now go through some experiments that map and measure the changes and final value of r_s to show how it is affected by the choice of shallow embedding model integrated with the GraphRE model.

5.2.1 Self-Loop Experiments

We now present the results of some experiments we ran to explore some these ideas regarding adding self-loops to the knowledge graph as a way of dealing with the merger of the representations of central nodes with those of their neighbourhoods. These experiments have been set up in a manner similar to the experiments described in Section 4. In this case, we only needed to select one dataset to test our hypothesis, so we decided to use NELL's **v3_ind** sub-dataset, but any of the other datasets would have also worked.

As we can see from Figure 5.1, in the case of using a TransE model to transform node representations in the GraphRE model, the r_s embedding remains quite unchanged from its original value. This is because it essentially cancels itself out in the process of being used to merge central node representations with those of their neighbours, so there are no gradients to pass back to the embedding. In contrast, the embedding changes over time as we expect it to in the ComplEx case, where this simplification does not occur in the same way.

These results show how using this self-loop scheme can help improve the GraphRE model by taking advantage of the nature of knowledge graphs and the way they are structured.

5.3 Dimensionality Scaling

We mentioned earlier that we expect the GraphRE model to be noticeably sensitive to choices regarding the dimensionality of the node representations depending on which shallow embedding model we choose to use for node transformations. This is because the performance of most pure shallow embedding models is directly related to entity dimensionality, so if we are to expect the shallow model used in our neural model to do its job well, we should adequately equip it with enough dimensions to do said job [25]. On the other hand, GNNs have a tendency to overfit when supplied with too many dimensions, so we still expect there to be a limit to the effectiveness of adding dimensions to the model to support its shallow embedding component [61]. Determining this sweet spot depends on things like which dataset is being operated on and the choice of shallow embedding model integrated with the GNN, but by varying the dimensionality and measuring the performance of the model, it should become apparent that the GraphRE model in general allows for models with larger node representation vectors than other classical GNNs while still eventually overfitting given a large enough dimensionality.

To this end, we have run the above described experiment to test how the GraphRE model reacts to changes in dimensionality.

5.3.1 Dimensionality Experiments

We now present the results of experiments that demonstrate the previously explained property regarding the dimensionality of the GraphRE model. These experiments have been set up in a manner similar to the experiments described in Section 4.

		NELL							
	Dim.	**v1**	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
GraphRE-TransE	32	91.82	97.37	96.69	96.45	98.30	93.40	94.87	96.88
	64	**94.87**	97.44	97.53	96.99	**99.34**	**95.22**	**96.07**	**97.32**
	128	94.20	**98.00**	**97.80**	**97.49**	97.36	94.39	96.00	97.27
	256	91.43	96.01	96.44	96.22	98.93	94.47	93.93	97.07
GraphRE-ComplEx	32	**93.73**	96.65	96.71	96.89	**99.28**	**96.58**	94.81	**96.99**
	64	91.58	**97.61**	97.38	97.43	99.22	96.13	**95.99**	96.92
	128	92.13	97.23	**97.59**	**97.62**	98.99	95.93	95.73	96.82
	256	91.99	97.49	94.80	96.41	99.21	95.41	95.45	96.52

Table 5.4: AUC-PR results with different representational dimensions for **NELL**.

		WordNet18RR							
	Dim.	**v1**	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
GraphRE-TransE	32	84.29	86.38	86.22	84.93	86.05	87.99	80.18	86.99
	64	85.70	87.23	87.02	86.78	85.00	88.11	81.40	86.59
	128	**87.93**	87.09	86.67	**87.23**	86.12	88.59	86.97	87.61
	256	87.24	**88.22**	**88.85**	86.81	**87.66**	**89.21**	**87.23**	**88.06**
GraphRE-ComplEx	32	87.78	87.17	87.06	86.29	88.23	89.57	83.87	86.10
	64	**87.79**	87.17	86.62	**88.91**	**90.85**	88.29	**85.67**	**90.14**
	128	88.73	**88.38**	85.70	87.33	84.77	89.31	83.37	89.16
	256	87.65	86.57	**89.98**	88.30	89.26	**90.13**	85.18	88.86

Table 5.5: AUC-PR results with different representational dimensions for **WN18RR**.

		Freebase15k-237							
	Dim.	**v1**	**v2**	**v3**	**v4**	**v1_ind**	**v2_ind**	**v3_ind**	**v4_ind**
GraphRE-TransE	32	87.81	88.68	87.20	88.86	82.69	88.04	86.58	87.42
	64	89.86	92.00	89.99	90.22	85.52	90.37	88.86	89.53
	128	90.81	**92.16**	91.29	91.31	86.28	90.37	89.62	89.53
	256	**91.05**	91.43	**91.94**	**91.59**	**86.58**	**90.74**	**89.37**	**90.19**
GraphRE-ComplEx	32	87.58	90.24	89.66	89.53	83.11	87.36	88.17	87.82
	64	87.61	91.30	90.90	**91.42**	83.82	89.46	89.18	88.85
	128	**90.45**	90.58	91.66	91.09	83.48	**90.86**	**89.68**	**89.74**
	256	90.11	**91.67**	**91.69**	89.70	**86.69**	88.78	87.12	89.04

Table 5.6: AUC-PR results with different representational dimensions for **FB15k-237**.

As we can see from Table 5.4, Table 5.5, and Table 5.6, there do seem to exist apparent “sweet spots” in dimensionality of the GraphRE model that changes depending on the dataset and the choice of shallow embedding model integrated into the GraphRE model. Interestingly, across all three datasets it appears that the TransE variant of the GraphRE model performs best in a dimensional category that is larger than the category that the ComplEx variant of the GraphRE model performs best in. In the case of the NELL dataset, the TransE variant settles at 64 dimensions while the ComplEx variant settles at 32. In the case of the WN18RR dataset, the TransE variant settles at 256 dimensions while the ComplEx variant settles at 64. In the case of the FB15k-237 dataset, the TransE variant settles at 256 dimensions while the ComplEx variant settles at 128. This consistent pattern matches what we could expect from these shallow embedding models because, when attempting to do tasks on their own in a transductive setting, the TransE model tends to benefit more from a high dimensionality than the ComplEx model [7]. The fact that this aspect of the models carries over after being integrated with a GNN further indicates that the GraphRE model is actively making use of the advantage that the shallow embedding models are granting it through their transformations of the node representations being passed as messages.

6
Conclusion

Contents

We now briefly go through some of ways future work could expand the efforts of this thesis as well as sum up the topics built on in this project.

6.1 Potential Future Extensions

Since we are introducing a new model in this thesis, future work that builds upon this could either go in the direction of further theoretical analysis or in the direction of applying the GraphRE model in novel environments where it could prove useful.

On the theoretical side, it might be interesting to do a deeper dive investigating the relationship between the GraphRE model and the limitations of the shallow embedding model integrated within it. For example, it might be interesting to see whether the GraphRE model is limited to learning only the rule inference patterns that its underlying shallow embedding model can learn or whether the GNN structure allows the GraphRE model to move past those boundaries. Another aspect of the model that might be worth exploring is the way the model learns to represent the spaces of different entity types. Entities of the same type may end up being mapped into spaces which share similar properties.

On the applications side, the GraphRE model's inductive capabilities mixed with its scalability might set it up for some interesting uses in the realm of live-learning. There exists a number of continuously growing knowledge graphs that are being expanded in a live manner [6, 62]. If it were possible to attach a GraphRE model to such a process and allow for the knowledge in the graph to not only grow but also for it to be subjected to a neural model that might help fill in missing links at the same time, the quality of such a knowledge graph could dramatically improve. The Graph RE strategies applied in a live learning setting may also be applicable to temporal knowledge graphs, a class of knowledge graphs that have timestamps describing changes in the graph data [63, 64]. It could also be interesting to see whether the GraphRE model has the capability of being used outside of knowledge graphs. If that were possible, the GraphRE model could act as a conduit for shallow embedding models to be used in a wider range of fields and areas of research.

6.2 Closing Remarks

Tackling knowledge completion tasks in an inductive learning setting may be the key to getting knowledge graphs to reach their full potential in useful applications outside of machine learning. While shallow embedding models have shown how effective they can be in transductive settings, their inability to move past that barrier has held them back.

In the course of thesis, we have shown that, by mixing shallow embedding models with message-passing concepts from the field of graph representation learning in the form of graph neural networks, we can augment shallow embedding models and use some of their strengths to solve inductive knowledge completion tasks. The GraphRE model demonstrates how that augmentation can be done in a distinct way that is competitive with existing state-of-the-art inductive models while remaining scalable and lightweight.

Appendices

A

Inductive Dataset Statistics

Contents

Here we will present statistics regarding the datasets used in the inductive experiments discussed in Section 4.1.

A.1 Dataset Statistics

		WN18RR			**FB15k-237**			**NELL**		
		relations	nodes	links	relations	nodes	links	relations	nodes	links
v1	train	9	2746	6678	183	2000	5226	14	10915	5540
	test	9	922	1991	146	1500	2404	14	225	1034
v2	train	10	6954	18968	203	3000	12085	88	2564	10109
	test	10	2923	4863	176	2000	5092	79	4937	5521
v3	train	11	12078	32150	218	4000	22394	142	4647	20117
	test	11	5084	7470	187	3000	9137	122	4921	9668
v4	train	9	3861	9842	222	5000	33916	77	2092	9289
	test	9	7208	15157	204	3500	14554	61	3294	8520

Table A.1: Statistics of the datasets used in our inductive experiments. Data taken from [15].

The dataset statistics are shown above in Table A.1.

References

[1] Antoine Bordes et al. “Translating Embeddings for Modeling Multi-relational Data”. In: *Advances in Neural Information Processing Systems*. Vol. 26. Curran Associates, Inc., 2013. URL: `https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html`.

[2] George A. Miller. “WordNet: a lexical database for English”. In: *Communications of the ACM* 38.11 (Nov. 1, 1995), pp. 39–41. URL: `https://doi.org/10.1145/219717.219748`.

[3] Tim Dettmers et al. “Convolutional 2D Knowledge Graph Embeddings”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 25, 2018). URL: `https://ojs.aaai.org/index.php/AAAI/article/view/11573`.

[4] Kurt Bollacker et al. “Freebase: a collaboratively created graph database for structuring human knowledge”. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’08. New York, NY, USA: Association for Computing Machinery, June 9, 2008, pp. 1247–1250. URL: `https://doi.org/10.1145/1376616.1376746`.

[5] Kristina Toutanova and Danqi Chen. “Observed versus latent features for knowledge base and text inference”. In: *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 57–66. URL: `https://aclanthology.org/W15-4007`.

[6] Wenhan Xiong, Thien Hoang, and William Yang Wang. “DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. arXiv:1707.06690. 2017, pp. 564–573. arXiv: `1707.06690[cs]`. URL: `http://arxiv.org/abs/1707.06690`.

[7] Zhiqing Sun et al. “RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space”. In: *International Conference on Learning Representations*. 2019. URL: `http://arxiv.org/abs/1902.10197`.

[8] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. “A Three-Way Model for Collective Learning on Multi-Relational Data”. In: *Proceedings of the 28th International Conference on Machine Learning*. 2011, pp. 809–816. URL: `https://openreview.net/forum?id=H14QEiZ_WS`.

[9] Bishan Yang et al. “Embedding Entities and Relations for Learning and Inference in Knowledge Bases”. In: *Proceedings of the International Conference on Learning Representations*. arXiv:1412.6575. 2015. arXiv: `1412.6575[cs]`. URL: `http://arxiv.org/abs/1412.6575`.

[10] Théo Trouillon et al. “Complex Embeddings for Simple Link Prediction”. In: *Proceedings of The 33rd International Conference on Machine Learning.* International Conference on Machine Learning. ISSN: 1938-7228. PMLR, June 11, 2016, pp. 2071–2080. URL: `https://proceedings.mlr.press/v48/trouillon16.html`.

[11] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *Proceedings of the International Conference on Learning Representations.* arXiv:1609.02907. 2017. arXiv: `1609.02907[cs,stat]`. URL: `http://arxiv.org/abs/1609.02907`.

[12] Boris Weisfeiler. *On construction and identification of graphs.* Vol. 558. Springer, 2006.

[13] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: *Proceedings of the International Conference on Learning Representations.* arXiv:1810.00826. 2019. arXiv: `1810.00826[cs,stat]`. URL: `http://arxiv.org/abs/1810.00826`.

[14] Michael Schlichtkrull et al. “Modeling Relational Data with Graph Convolutional Networks”. In: *European Semantic Web Conference.* arXiv:1703.06103. Springer. 2018, pp. 593–607. arXiv: `1703.06103[cs,stat]`. URL: `http://arxiv.org/abs/1703.06103`.

[15] Komal K. Teru, Etienne Denis, and William L. Hamilton. “Inductive Relation Prediction by Subgraph Reasoning”. In: *International Conference on Machine Learning.* arXiv:1911.06962. PMLR. 2020, pp. 9448–9457. arXiv: `1911.06962[cs,stat]`. URL: `http://arxiv.org/abs/1911.06962`.

[16] Shuwen Liu et al. “INDIGO: GNN-Based Inductive Knowledge Graph Completion Using Pair-Wise Encoding”. In: *Advances in Neural Information Processing Systems.* Vol. 34. Curran Associates, Inc., 2021, pp. 2034–2045. URL: `https://proceedings.neurips.cc/paper/2021/hash/0fd600c953cde8121262e322ef09f70e-Abstract.html`.

[17] Zhaocheng Zhu et al. “Neural Bellman-Ford Networks: A General Graph Neural Network Framework for Link Prediction”. In: *Advances in Neural Information Processing Systems* 34.arXiv:2106.06935 (2021), pp. 29476–29490. arXiv: `2106.06935[cs]`. URL: `http://arxiv.org/abs/2106.06935`.

[18] Zonghan Wu et al. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[19] Cheng-Te Li. “Explainable detection of fake news and cyberbullying on social media”. In: *Companion Proceedings of the Web Conference 2020.* 2020, pp. 398–398.

[20] Fan Zhou et al. “Variational graph neural networks for road traffic prediction in intelligent transportation systems”. In: *IEEE Transactions on Industrial Informatics* 17.4 (2020), pp. 2802–2812.

[21] David Pujol Perich et al. “NetXplain: Real-time explainability of graph neural networks applied to computer networks”. In: *Proceedings of the First MLSys Workshop on Graph Neural Networks and Systems (GNNSys’ 21), San Jose, CA, USA, 2021.* 2021, pp. 1–7.

[22] Qiong Yang et al. “Prediction of liquid chromatographic retention time with graph neural networks to assist in small molecule identification”. In: *Analytical Chemistry* 93.4 (2021), pp. 2200–2206.

[23] Jie Zhou et al. “Graph neural networks: A review of methods and applications”. In: *AI Open* 1 (2020), pp. 57–81.

[24] Shaoxiong Ji et al. “A survey on knowledge graphs: Representation, acquisition, and applications”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.2 (2021), pp. 494–514.

[25] Quan Wang et al. “Knowledge graph embedding: A survey of approaches and applications”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.12 (2017), pp. 2724–2743.

[26] Frank Manola and Eric Miller. *RDF 1.1 Primer*. URL: `https://www.w3.org/TR/rdf11-primer/`.

[27] Tianxing Wu et al. “Knowledge graph construction from multiple online encyclopedias”. In: *World Wide Web* 23.5 (2020), pp. 2671–2698.

[28] Chenyan Xiong, Russell Power, and Jamie Callan. “Explicit semantic ranking for academic search via knowledge graph embedding”. In: *Proceedings of the 26th international conference on world wide web*. 2017, pp. 1271–1279.

[29] Robert Logan et al. “Barack’s Wife Hillary: Using Knowledge Graphs for Fact-Aware Language Modeling”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 5962–5971. URL: `https://aclanthology.org/P19-1598`.

[30] Hongwei Wang et al. “Multi-task feature learning for knowledge graph enhanced recommendation”. In: *The world wide web conference*. 2019, pp. 2000–2010.

[31] Zhe Chen et al. “Knowledge graph completion: A review”. In: *Ieee Access* 8 (2020), pp. 192435–192456.

[32] Ralph Abboud et al. “Boxe: A box embedding model for knowledge base completion”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9649–9661.

[33] Seyed Mehran Kazemi and David Poole. “Simple embedding for link prediction in knowledge graphs”. In: *Advances in neural information processing systems* 31 (2018).

[34] Yuanfei Dai et al. “A survey on knowledge graph embedding: Approaches, applications and benchmarks”. In: *Electronics* 9.5 (2020), p. 750.

[35] Andrew Carlson et al. “Toward an Architecture for Never-Ending Language Learning”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI’10. Atlanta, Georgia: AAAI Press, July 11, 2010, pp. 1306–1313.

[36] Dat Quoc Nguyen. “A survey of embedding models of entities and relationships for knowledge graph completion”. In: *Proceedings of the Graph-based Methods for Natural Language Processing (TextGraphs)*. Barcelona, Spain (Online): Association for Computational Linguistics, Dec. 2020, pp. 1–14. URL: `https://aclanthology.org/2020.textgraphs-1.1`.

[37] Christian Meilicke et al. “Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion”. In: *International semantic web conference*. Springer. 2018, pp. 3–20.

[38] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. "Revisiting semi-supervised learning with graph embeddings". In: *International conference on machine learning*. PMLR. 2016, pp. 40–48.

[39] Peifeng Wang et al. "Logic attention based neighborhood aggregation for inductive knowledge graph embedding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7152–7159.

[40] Mingyang Chen et al. "Meta-Knowledge Transfer for Inductive Knowledge Graph Embedding". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2022, pp. 927–937.

[41] Paul W Baim. "The PROMISE method for selecting most relevant attributes for inductive learning systems". In: *Reports of the Intelligent Systems Group* (1982), pp. 82–1.

[42] Peter W Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[43] Ryan A Rossi et al. "Transforming graph data for statistical relational learning". In: *Journal of Artificial Intelligence Research* 45 (2012), pp. 363–441.

[44] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". In: *arXiv preprint arXiv:1312.6203* (2013).

[45] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in neural information processing systems* 29 (2016).

[46] Thorsten Joachims. "Transductive learning via spectral graph partitioning". In: *Proceedings of the 20th international conference on machine learning (ICML-03)*. 2003, pp. 290–297.

[47] Chu Wang, Babak Samari, and Kaleem Siddiqi. "Local spectral graph convolution for point set feature learning". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 52–66.

[48] William L Hamilton. "Graph representation learning". In: *Synthesis Lectures on Artifical Intelligence and Machine Learning* 14.3 (2020), pp. 1–159.

[49] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[50] Zhiqian Chen et al. "Bridging the gap between spatial and spectral domains: A survey on graph neural networks". In: *arXiv preprint arXiv:2002.11867* (2020).

[51] Reginald Sawilla. *A survey of data mining of graphs using spectral graph theory*. Defence R & D Canada-Ottawa, 2008.

[52] Jiong Zhu et al. "Beyond homophily in graph neural networks: Current limitations and effective designs". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7793–7804.

[53] Dylan Sandfelder, Priyesh Vijayan, and William L Hamilton. "Ego-gnns: Exploiting ego structures in graph neural networks". In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 8523–8527.

[54] Petar Veličković et al. “Graph Attention Networks”. In: *Proceedings of the International Conference on Learning Representations.* arXiv:1710.10903. 2018. arXiv: `1710.10903[cs,stat]`. URL: `http://arxiv.org/abs/1710.10903`.

[55] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *Advances in Neural Information Processing Systems.* Vol. 30. Curran Associates, Inc., 2017. URL: `https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html`.

[56] Clemens Damke, Vitalik Melnikov, and Eyke Hüllermeier. “A novel higher-order Weisfeiler-Lehman graph convolution”. In: *Asian Conference on Machine Learning.* PMLR. 2020, pp. 49–64.

[57] Christopher Morris et al. “Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.1 (July 17, 2019), pp. 4602–4609. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/4384`.

[58] Ralph Abboud et al. “The Surprising Power of Graph Neural Networks with Random Node Initialization”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21.* Ed. by Zhi-Hua Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 2112–2118. URL: `https://doi.org/10.24963/ijcai.2021/291`.

[59] Alessandro Generale, Till Blume, and Michael Cochez. “Scaling R-GCN Training with Graph Summarization”. In: *arXiv preprint arXiv:2203.02622* (2022).

[60] Jiaxuan You, Tianyu Du, and Jure Leskovec. “ROLAND: Graph Learning Framework for Dynamic Graphs”. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 2022, pp. 2358–2366.

[61] Alex Morehead, Watchanan Chantapakul, and Jianlin Cheng. “Semi-Supervised Graph Learning Meets Dimensionality Reduction”. In: *arXiv preprint arXiv:2203.12522* (2022).

[62] Danh Le-Phuoc et al. “The graph of things: A step towards the live knowledge graph of connected things”. In: *Journal of Web Semantics* 37 (2016), pp. 25–35.

[63] Borui Cai et al. “Temporal knowledge graph completion: A survey”. In: *arXiv preprint arXiv:2201.08236* (2022).

[64] Alberto Garcia-Duran, Sebastijan Dumancic, and Mathias Niepert. “Learning sequence encoders for temporal knowledge graph completion”. In: *arXiv preprint arXiv:1809.03202* (2018).