HOPLoP: Multi-hop Link Prediction over Knowledge Graph Embeddings

by

Varun Ranganathan

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

Abstract

Large-scale Knowledge Graphs (KGs) built from Web resources have become ubiquitous, supporting many applications as a source of data for a plethora of tasks in natural language processing and artificial intelligence. Despite their success, building and maintaining KGs remains a challenge: manual approaches offer high accuracy but very limited coverage while automatic approaches yield low coverage due to the limitations of the state-of-the-art Information Extraction tools. As a result, even the best KGs out there are notoriously incomplete, giving rise to a large crop of reasoning tools for the task of Link Prediction (LP); which aims to find missing links between entities based on structural regularities in the graph. Among the many LP methods, those based on embeddings have become prevalent in the literature. In particular, multi-hop LP algorithms have been found to model complex relations better by exploiting correlations between direct relations and paths connecting the entities related in that way. However, state-of-the-art multi-hop LP methods find paths by traversing the KG, in its original discrete graph representation. Therefore, they are hampered by the incompleteness of the KG and the skewed degree distribution in the KG, causing entities that are involved in disproportionately many facts to be overly represented during training and testing, resulting in poor generalization. We present a simple, efficient and highly effective multi-hop LP method, called HOPLoP, whose main predicate is to perform "traversals" on the embedding space instead of the KG itself. We show how to train and tune our method, and report on experiments with different embedding-based representations of KGs on many benchmarks, showing that HOPLoP improves each LP method on its own and that it consistently outperforms

the previous state-of-the-art multi-hop LP methods by a good margin. Finally, we also describe a method for the interpretation of the paths generated during reasoning by HOPLoP when used with embedding models where entities and relations are mapped to the same space.

Preface

This research stems from my passion for automated reasoning and optimization techniques, along with experimental observations over 4 years. In the coming decades, we will need to use AI-based approaches to tackle world problems and open new opportunities of growth. AI algorithms need better, "information-rich" representations of their world that allow them to learn crucial predicative patterns quicker. This thesis presents HOPLoP, a method for end-to-end differentiable multi-hop link prediction, which effaciously traverses a continuous representation of a knowledge graph to make predictions. Findings from this thesis could be adopted towards a "data-centric" framework for AI. By separating the data representation process from the modeling stage, we may be able to significantly reduce the computation and space required for AI tasks.

Acknowledgements

First, I would like to thank all the scientists and doctors that have helped develop a COVID-19 vaccine so quickly. At the time of writing, Alberta is out of COVID, which is incredible; compared to a year ago, when there was no end in sight. I would also take this opportunity to globally thank all the doctors, nurses and social workers for fighting against COVID.

I would like to thank my supervisor, Prof. Denilson Barbosa, for guiding me throughout my masters. I remember our first conversation, on Feb 27, 2019. Denilson's first question to me was, "How can I get you to join us at to the U of A?". I was taken back, since, this was our very first conversation and he couldn't have gauged much about me from my application and a referral. We had an inspired hour of conversation and since my arrival at the U of A, he has provided me with a great office space with a great office mate (shout-out to Tobias), all the tech needed for me to innovate, ample amounts of creative space to pursue independent curiosities (such as ZORB [1]), and most importantly, developing my ability to critically question so called "truths" of the world.

I would like to thank my former supervisor from PES University, Prof. Natarajan Subramanyam. Back in fall 2017, after working on a project at the Microsoft Innovation Labs, like any novice in Machine Learning, I was terrified of "backpropagation". While attempting my own derivation, I discovered "A new backpropagation algorithm without gradient descent" [2]. I, similar to my peers, was skeptical of my own idea,

[1] https://arxiv.org/abs/2011.08895
[2] https://arxiv.org/abs/1802.00027

but Natarajan guided me through my first innovative-research work, which eventually lead to being one of the most influential data science papers of 2018 [3]. He has guided me through several key decisions in my initial research career (e.g., going to NTU instead of CMU for a summer internship) which, looking back, have had a monumental impact in my life.

I would like to thank both my supervisors for their support of my winning application for the Alberta Innovates Graduate Student Scholarship, which will support me through my next entrepreneurial journey. I also thank my committee members Prof. Eleni Stroulia and Prof. Marek Reformat for taking the time to review my thesis. I thank all staff members at the Alberta Machine Intelligence Institute (shout-out Talat, Shazan, Mara, Johannes, and others who I have worked closely with) and all their partner companies (shout-out to FunnelAI, Digital Public Square, Virtuo, Quickfyre, Provision Analytics) for allowing me to deliver high-impact solutions.

I would like to thank my family (shout-out to Amma, Appa, Patti, Menakshi, Simba) friends (shout-out to Rahul, Vishwas, Aaditya, Pavan, Vignesh) for supporting me from various time-zones. Last but not the least, I thank Navya, for sticking by my side through the ebbs and flows of the last two years of our lives.

[3] http://opendatascience.com/most-influential-data-science-research-papers-for-2018/

Table of Contents

List of Tables

List of Figures

List of Symbols

$(.)^T$	transpose of matrix
D	dimension or dataset, depending on usage
D'	output dimension
E	error function
F	function
H	number of hops or hidden layers, depending on usage
L	loss function
N	data batch size
Q	transition matrix
$Re(.)$	real component
U_l	number of neurons in layer l
W_l	neural layer
X	Inputs (set or matrix)
Y	Outputs (set or matrix)
$[.;.]$	concatenation operation
$\Leftrightarrow$	equivalence relationship
Pr	probability

Θ	parameters or angle, depending on usage
$\Theta^{(t)}$	parameters at time t
$\approx$	approximately equals
$\circ$	composition
δ_l	partial gradients at layer l
$\forall$	for all
γ	margin, in max-margin functions
$\hat{(.)}$	predicted
$\in$	in operation
∞	infinity
ln	natural logarithm
$\mathbb{C}$	Complex values
$\mathbb{R}$	Real values
$\mathcal{E}$	Set of entities
$\mathcal{G}$	Graph
$\mathcal{K}$	Set of tasks
$\mathcal{L}$	Set of links
$\mathcal{O}(.)$	Complexity (Big-O) notation
$\mathcal{P}$	Set of paths
$\mathcal{R}$	Set of relations
$\mathcal{W}$	Core tensor
μ	learning rate
$\nabla_x f(x_0) = f'(x)(x_0)$	Gradient of function f with respect to variable x at x_0

σ	sigmoid function
$\subseteq$	subset or equivalent
$\times$	product
$\times_i$	tensor product along the i^{th} mode
$\vec{(.)}$	vector
$\vec{(.)}^+$	vector related to a positive example
$\vec{(.)}^-$	vector related to a negative example
$\vec{v_h}$	translation vector at hop h
d	counter for dimensionality, used to represent elements in a vector
e_s	source entity
e_t	target entity
$exp(.)$	exponent function
$f'(x) = \frac{\partial f(x,y)}{\partial x}$	Partial derivative of function f with respect to variable x.
h	hops counter, used to represent a particular hop in the path traversal
h_l	hidden layer l representation
i	general counter
l	link
r	relation
x	input
y	output

Abbreviations

AI Artificial Intelligence.

ANN Artificial Neural Network.

BCE Binary Cross-entropy Error.

BFS Breadth First Search.

BP Back-Propagation.

BPTT Back-Propagation Through Time.

DL Deep Learning.

GRU Gated Recurrent Unit.

IE Information Extraction.

KB Knowledge Base.

KBC Knowledge Base Completion.

KBP Knowledge Base Population.

KG Knowledge Graph.

LogR Logistic Regression.

LP Link Prediction.

LR Linear Regression.

LSTM Long Short Term Memory.

MCTS Monte Carlo Tree Search.

ML Machine Learning.

MSE Mean Squared Error.

NAS Neural Architecture Search.

NELL Never Ending Language Learning.

NLG Natural Language Generation.

NLP Natural Language Processing.

NN Neural Network.

PCRA Path-Constraint Resource Allocation.

PRA Path Ranking Algorithm.

ReLU Rectified Linear Unit.

RepL Representation Learning.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

RWR Random Walk with Restart.

SL Supervised Learning.

SOTA state-of-the-art.

SPARQL SPARQL Protocol and RDF Query Language.

SSL Self-Supervised Learning.

UL Unsupervised Learning.

xAI eXplainable Artificial Intelligence.

YAGO Yet Another Great Ontology.

ZORB Zeroth Order Relaxed Backpropagation.

Chapter 1

Introduction

The *Jeopardy!* game show, aired on February 14-15 2011, saw Watson, a question-answering system, win the first place prize [1] of US$1 Million after defeating human champions Brad Rutter and Ken Jennings. To maximize earnings in each round of this quiz show, Watson, like any human player, was tasked to generate questions, in natural language, from clues, also given in natural language. If the question, put forth by a player, contains the required entity [2], that player is said to have won that round. This is a non-trivial task for a human since the game covers a wide variety of topics including history and current events, the sciences, the arts, popular culture, literature, and languages. The plethora of knowledge required to maximize earnings in this game is substantial, making this task difficult for humans without access to external resources. Although computers do not quite share the same issue of storage [3], the task still remains non-trivial, since computers struggle to interface with humans via natural language [1], let alone win a game show that requires interactive communication, in natural language, between the player and the game show host [4].

[1] IBM donated 100% of Watson's winnings to charity, with 50% of those winnings going to World Vision and 50% going to World Community Grid.

[2] The correct entity may refer to famous individuals, locations, companies, etc.

[3] During the game, Watson had access to 200 million pages of structured and unstructured content consuming 4TB of disk storage, including the full text of the 2011 edition of Wikipedia, but was not connected to the Internet.

[4] Although Watson did not have to perform speech-to-text, the questions were given to it in plain text in natural language, which requires *understanding* https://www.ibm.com/cloud/watson-natural-language-understanding

To identify correct entities based on the given clues, Watson utilized more than 100 different algorithms [5] to hypothesize a question that involves the correct entity. Before Watson could do any of that, it required knowledge about various topics of the world, including the knowledge of their representation: the english language. Introducing Knowledge Graphs (KGs) (a.k.a Knowledge Base (KB)): a graph-based data structure that stores knowledge of the world in it's *links*, which can be understood by both machines and humans. Each *link* represents a fact, usually expressed in natural language. A *link*, represented as a triple (e_s, r, e_t), is directed, indicating the relationship r between two entity nodes that connects source entity e_s to target entity e_t.

Watson was powered by open and large-scale KGs such as DBpedia [2], WordNet [3], and YAGO [4] amongst several other data sources. Without this preliminary knowledge, Watson cannot recognize entities and reason about them, eventually failing at question-answering [5]. IBM's Watson, which was just a question-answering system back in 2011, now provides cognitive services to almost 5,000 companies, generating revenues upwards of US$1 Billion [6].

Similar to IBM Watson are Artificial Intelligence (AI)-based assistants such as Apple's Siri [7], Amazon's Alexa [8], Microsoft's Cortana [9] and Google Assistant [10], which interact with humans via natural language. These AI-based assistants are powered by KGs to recognize entities and user intents. For example, if I were to speak to Google Assistant saying, "**hey google** *play* `apollo` by `hardwell` on `spotify`", the program needs to recognize **hey google** (cue), followed by entities such as `apollo` (song), `hardwell` (artist), and `spotify` (app). The program also needs to recognize the user's intent, *play*, and execute instructions accordingly. Additionally, the program

[5] https://www.idga.org/archived-content/whitepapers/watson-a-system-designed-for-answers-the-future-of

[6] https://www.appsruntheworld.com/customers-database/products/view/ibm-watson

[7] https://www.apple.com/ca/siri/

[8] https://en.wikipedia.org/wiki/Amazon_Alexa

[9] https://www.microsoft.com/en-us/cortana

[10] https://assistant.google.com/

should also verify that `apollo` is a song by the artist `hardwell`, and `spotify` is an application installed on the device. This verification process queries various KGs to determine the "truth" value of these *facts*, which are then used to drive real-time decisions [11] autonomously made by the software. Similarly, KGs are extensively used in search [6] (Google Knowledge Graph [12]), chatbots [7], recommender systems [8] and autonomous systems [9]. We, recently, recognize the importance of KGs as a data source for Machine Learning (ML), eXplainable Artificial Intelligence (XAI) [10], and downstream tasks such as question-answering [5] and automated reasoning [11].

KG Accuracy Requirements. The tasks discussed above need KGs that are both *accurate* and *comprehensive*, covering the topics of interest to users. Achieving both goals is difficult. On one hand, manually constructing KGs is time consuming [12]. For example, WordNet, a manually constructed KG that contains expert-annotated information about 155,287 English words and phrases, linking them via semantic relations such as `Hypernymy` and `Synonymy`, took 27 years to develop, and yet it covers only a small fraction of the "ever-growing" language. As a result, most manually created KGs tend to have poor coverage [13], even if their accuracy is high.

On the other hand, KGs automatically constructed with the help of Information Extraction (IE) tools, applied to Web-scale text corpora, have many more entities but are more error-prone, since the state-of-the-art (SOTA) is not sophisticated enough to understand the nuances of natural language [14, 15]. Also, even when IE methods succeed, their coverage is limited to the facts explicitly mentioned in the text. Let's consider this excerpt from Wikipedia describing the movie `TheTerminator` [13]:

> The Terminator is a 1984 science fiction action film released by Orion Pictures, co-written and directed by James Cameron and starring Arnold Schwarzenegger, Linda Hamilton and Michael Biehn. It is the first work in the Terminator franchise. ...

[11] Decision scenarios and appropriate reactions may be *programmed*, such as exception handling.
[12] https://en.wikipedia.org/wiki/Google_Knowledge_Graph
[13] https://en.wikipedia.org/wiki/Terminator_(franchise)#The_Terminator_(1984)

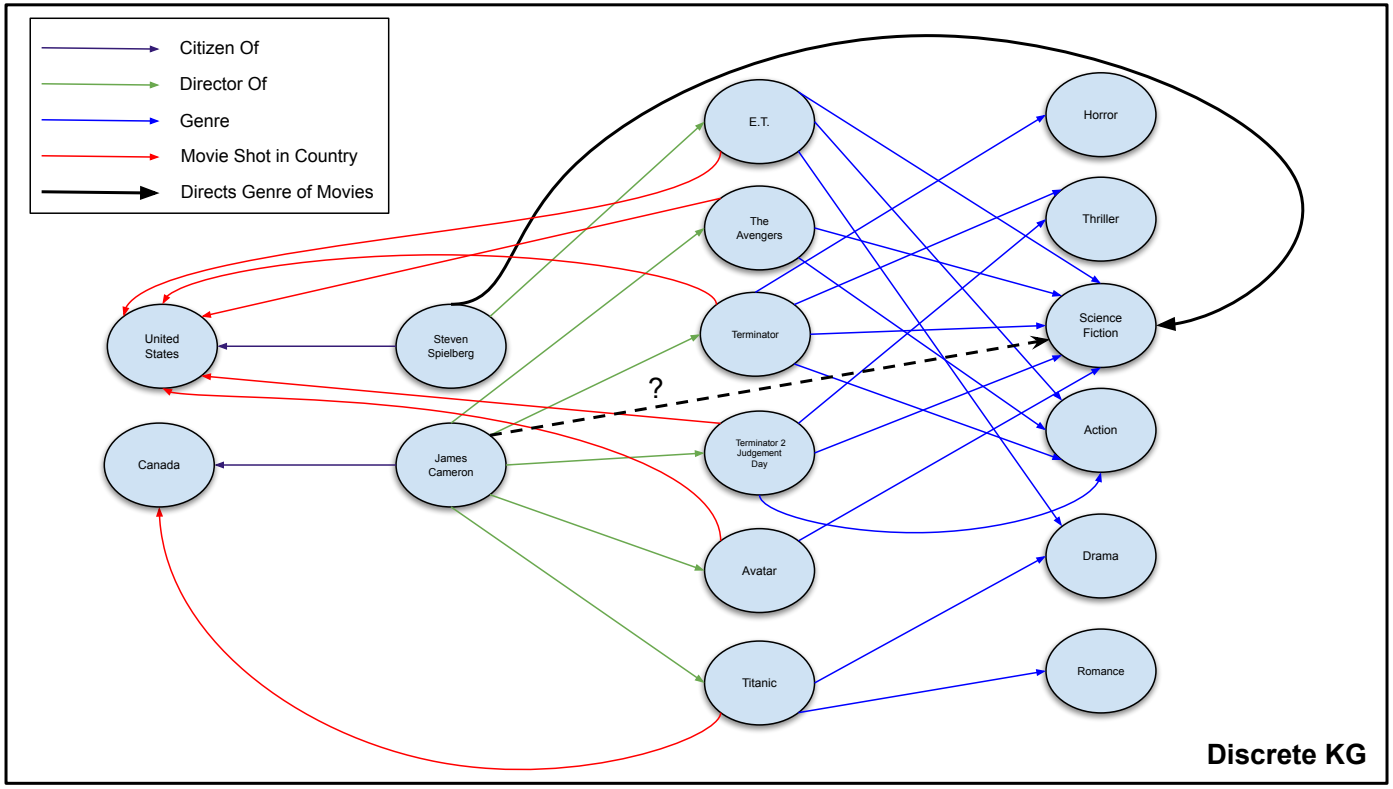


Figure 1.1: (Best viewed in color) An example KG showing a few entities and relations from the `Movies` domain. We color coded the links of the graph to signify specific relations. Notice that entities such as `UnitedStates`, `JamesCameron`, `Action`, and `ScienceFiction` can be consider as supernodes of this graph since they are linked to a higher number of entities. The dashed dark-black line illustrates our question: given this KG, can we find a path that connects `JamesCameron` and `ScienceFiction` to provide us with evidence that (`JamesCameron`, `DirectsGenreofMovies`, `ScienceFiction`).

At the time of writing, the best IE-based KG construction algorithm would be able to extract explicit facts, expressed as links `(JamesCameron, DirectorOf, TheTerminator)`[14] and `(TheTerminator, Genre, ScienceFiction)`, but would miss out an implied fact, expressed as link `(JamesCameron, DirectsGenreOfMovies, ScienceFiction)`, which can be inferred from the other two. As a result, it is well known that even the best KGs in use today are notoriously incomplete [16].

State-of-the-art KG Construction. Building accurate and comprehensive KGs is often done through a combination of *extraction* methods that capture facts explicitly expressed in the source and *inference* methods that can derive implicit facts. A popular strategy for inferring missing facts in recent years is called Link Prediction (LP), which is the task of predicting the *likelihood* that any link exists in the KG, by reasoning over the observed links in the KG.

The field of LP has been dominated by KG embedding models [17–19], since they provide an elegant solution to the incompleteness problem. These methods learn vector representations for discrete entities and relations of a KG and a scoring function such that the score predicted for an observed link is *generally* higher than for an unobserved link. Figure 1.2 illustrates the idea (note: details will be clearer later in section 2.6.2). By tuning the dimensionality of the embedding space, these models are able to generalize to unseen facts [20]. However, KG embedding models consider only two entities and one relation to make a prediction. By doing so, these approaches cannot reason over multiple relations and therefore, fail to model complex relations [21].

Continuing our example, an embedding-based LP algorithm explicitly requires that the relation `DirectsGenreOfMovies` be present in the graph, requiring manual or automatic *population* of the KG. One relatively low-effort "automatic" approach to collect links about the relation, such as `DirectsGenreOfMovies`, requires manual

[14] We follow the standard in the literature and represent multi-word entities and relations with "camelcasing" to avoid confusion.

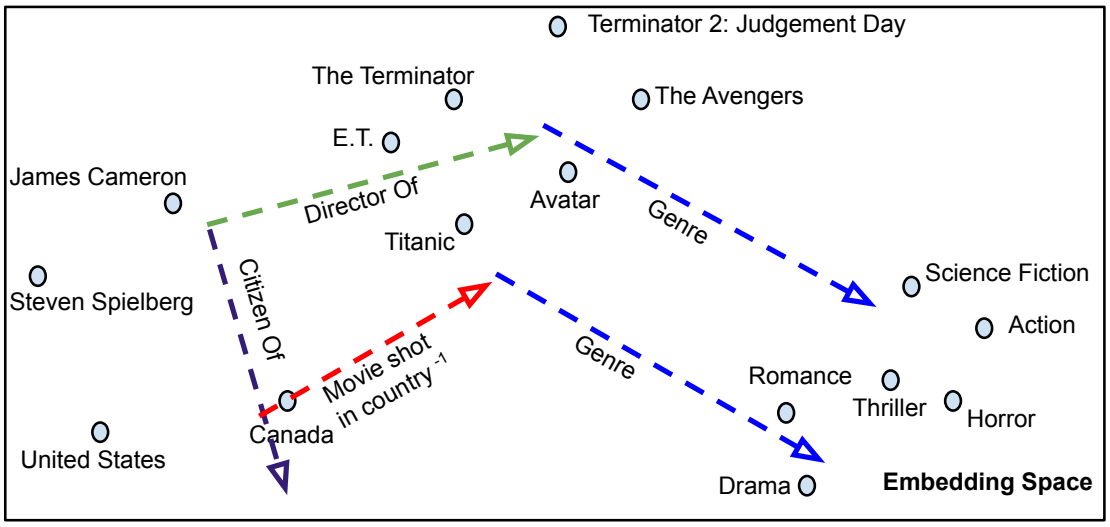


Figure 1.2: (Best viewed in color, refer to the color scheme in figure 1) How might TransE represent our example KG? An illustration showing a few entities embedded in a 2D space using TransE. We can observe how TransE "fills the gap in knowledge" by noticing a link missing between `Avatar` and `Action` in figure 1. Upon observing 1.2, we can intuit that if we added the vector $r_{\vec{Genre}}$ to the point $e_{\vec{Avatar}}$, we would reach close to the required target entity position $e_{\vec{Action}}$, suggesting a missing link. We also see that the resultant point in space is furthest from `Drama`, suggesting a low likelihood for (`Avatar`, `Genre`, `Drama`).

derivation of logical formulae, which is then converted to SPARQL codes to query for information from the KG. For example, the relation `DirectsGenreOfMovies` can be expressed by this logical formula: `DirectorOf` $\circ$ `Genre`, where $\circ$ represents the composition of two relations (see section 3.1.4 for more). A complex relation, such as `AthletePlaysSport`, would require multiple logical formulae to achieve high coverage. For example, the relation `AthletePlaysSport` can be represented by 2 formulae:

- `AthletePlaysForTeam` $\circ$ `TeamPlaysSport`
- `AthletePlaysInLeague` $\circ$ `LeagueStadiums` $\circ$ `SportUsesStadium`$^{-1}$

We can see that, complex relations may be represented by multiple logical formulae. Even if we were to satisfy this laborious requirement of creating links with the relation of interest, KG embedding models will continue to ignore the structural information of the KG, inhibiting their predictive power.

LP Meets Graph Traversals. There is strong evidence in the literature showing that leveraging graph traversals to create features helps with LP as it allows *correlations* between paths and direct relations to be exploited [22]. LP algorithms that leverage structural information gathered from *paths* are called *multi-hop* link prediction algorithms. For example, the Path-Ranking Algorithm (PRA) [23], the first multi-hop LP algorithm, learns to assign scores to pre-computed graph traversals such that, paths that tend to reach correct target entities are scored higher. To *traverse* a path, PRA utilizes a relation-specific transition matrix for each relation in the path and calculates the probability of reaching any entity from a source entity. Through the training process, PRA recognizes the structure of the KG (through the various paths given as input to PRA) and semantics behind each relation (through the transition matrices). It learns to perform LP by tuning its weights for each path such that it's traversal procedure *reaches* the required target entities. By tuning the weights, PRA "scores" or "ranks" the paths in the KG. Paths that are ranked higher provide a higher *support* [15] for a particular relation. Following our previous example of `DirectsGenreOfMovies`, given sufficient number of training instances, a PRA model will learn to score the path `DirectorOf` $\rightarrow$ `Genre` relatively high since `DirectsGenreOfMovies` is the same as the composition of the two other relations, `DirectorOf` and `Genre`. Similarly, for a relation `AthletePlaysSport`, paths such as, `AthletePlaysForTeam` $\rightarrow$ `TeamPlaysSport`, which contain sports-based relations, e.g. `AthleteHomeStadium`, `CoachWonTrophy`, `ChampionshipGameOfTheNationalSport`, will be scored high [23]. These examples show that graph traversals allow LP algorithms to correlate paths with relations in the KG. By doing so, multi-hop LP algorithms learn the global structure of the graph, boosting performance in LP.

Multi-hop LP algorithms are also referred to as multi-hop *reasoning* algorithms since they provide an interpretable path which can be used as a rule to predict links

[15] Support refers to how often a given path leads to the required target entiti(es).

in the KG. These algorithms boost LP performance while simultaneously providing the KG system with interpretable reasoning capabilities. In this thesis, we look to improve the performance of the multi-hop LP process without losing the ability to interpret the reasoning process. Before improving the multi-hop LP process, we first identify what constrains them.

Constraints on multi-hop LP algorithms. Although graph traversals greatly help with LP, they are constrained by the incompleteness of the KG and by the skewed degree distribution of nodes. For example, at the time of writing, about half of all `Citizenship` (`wdt:P27`) facts in Wikidata involve just 10 countries, with the `UnitedStates` (`wd:Q30`) alone having 13.7% of all facts. This data skew results in embedding-based LP methods, attending more to these *supernodes* which appear disproportionately more both in the training and in the validation datasets used in the literature [19]. Moreover, there are also very large discrepancies in node degrees, and therefore, in the number of paths involving them. Keeping with the example of `Citizenship`, the degree of the node corresponding to the `UnitedStates` in Wikidata is 4.8M, while the degree of `TheNetherlands` (`wd:Q29999`), the 10th country with the highest number of facts about `Citizenship`, is approximately 160K. Therefore, LP methods that exploit graph traversals are likely to attend to those paths related to the `UnitedStates` disproportionately more than other countries. These issues inhibit the performance of all multi-hop LP algorithms, since they traverse the discrete KG.

1.1 Thesis Objective

In this thesis, we explore methods for Knowledge Base Completion (KBC), which fill the "information" gaps in KGs through the LP process. Upon exploring relevant literature, we identify issues pertaining to KGs that negatively influence previous LP algorithms. We introduce a novel multi-hop LP framework that learns to traverse an *embedded* space where the issues of skewed node degree and incompleteness are

mitigated. This embedded representation of the KG is derived from KG embedding methods, from which we borrow a few insights:

- KG embedding methods embed structural information of discrete entities onto a point in continuous space, represented by a D-dimensional vector [24]. Uniformly weighting all points in the entity embedding space, i.e. treating all points in space equally, should resolve the skewed node degree issue.

- KG embedding models produce a score to represent the likelihood of a link. Controlling the dimensionality of KG components allows them to generalize to unseen links. We can view the generated entity embedding space to be "more-complete" than the KG in its discrete form.

- KG embedding methods often project the entity embedding onto a relation-specific plane [25–28] to compute a similarity measure between two entities. Creating relation-specific functions does not take into account the structural information of the graph and re-introduces errors caused by skewed node degrees. For example, if we were to create a relation-specific function for the Wikidata relation `Citizenship` in a KG, a KG embedding method would perform the similarity search while biasing towards certain dimensions. In this scenario, the embedding for entity `UnitedStates` may share similarities with embeddings for all individuals in the KG. These similarities maybe abused such that individuals' `Citizenship` is associated with `UnitedStates` with a spuriously high likelihood. Traversing the entity embedding space would allow the agent to discover paths that leads to supporting or refuting a link.

We hypothesize that the performance of a multi-hop LP algorithm can be improved if it is allowed to leverage graph traversals that are *not* constrained by the KG itself. To verify our hypothesis, we introduce a simple yet efficacious multi-hop link prediction framework called HOPLoP in which graph "traversals" and "paths" are

defined over the *embedding* space instead. HOPLoP is an end-to-end differentiable multi-hop LP framework that learns to traverse an embedding space while distinguishing between existent and non-existent links of the KG. By doing so, HOPLoP is able to recognize and account for errors in the embedded representation of the KG and create appropriate decision boundaries, which leads to performance boosts in LP. To evaluate HOPLoP, we follow standard evaluation methodologies for relation prediction and entity prediction. We observe significant improvements over previous SOTA algorithms for LP. Our contributions are 5-fold:

1. We introduce HOPLoP, an end-to-end differentiable multi-hop link prediction framework for LP over large KGs. End-to-end differentiability improves computational efficiency, allowing for simple optimization algorithms such as Gradient Descent (GD) [29].

2. HOPLoP traverses over an embedding space. By doing so, HOPLoP mitigates issues related to incompleteness and skewed degree distributions for nodes. We are the first to demonstrate traversals over a continuous space for LP.

3. We evaluate HOPLoP on 2 popular datasets, and introduce 2 new datasets for the task of multi-hop LP. We evaluate HOPLoP on 2 tasks: entity prediction and relation prediction, and consistently outperform SOTA LP performance across all datasets and metrics. For example, on the relation prediction task, HOPLoP advances previous SOTA methods by reducing errors by 46.53% on NELL-995 and 54.97% on FB15K-237. On the entity prediction task, HOPLoP advances previous SOTA methods with an error reduction of 53.85% on WN18RR and 56.67% on YAGO3-10.

4. We look to further explore advantages of HOPLoP and it's end-to-end differentiability. We aim to see if we can confirm the advantages of multi-task learning and parameter sharing mechanisms [30]. Similar to how *Single-Model* [31] ad-

vanced *Path-RNN* [32], we advance HOPLoP to M-HOPLoP, capable of *reasoning* for multiple relations. M-HOPLoP outperforms HOPLoP across all datasets for relation prediction.

5. We also provide a method to interpret reasoning paths of HOPLoP(TransE). By interpreting negative paths of HOPLoP(TransE), we find negative rules, which can be used to refute the existence of a relation between two entities. We are the first to introduce the notion of a negative path.

1.2 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 introduces all the preliminary knowledge required to understand the remainder of this thesis, including KGs, LP, and ML concepts such as GD, Backpropagation (BP) and Neural Networks (NNs). We also provide an overview of KG embeddings and discuss variants used in experiments. Chapter 3 delves into previous approaches for multi-hop LP. We discuss several key approaches, their advantages and disadvantages, and how they are different from HOPLoP. Chapter 4 presents HOPLoP with its motivations, mathematical formulations of the task, descriptions of the model and the training process. We improve upon HOPLoP by incorporating parameter sharing mechanisms for multi-task learning. We introduce M-HOPLoP which can reason about all relations instead of just one. Chapter 5 describes datasets and experiments performed to evaluate the performance of HOPLoP. Results displayed in tables 5.3, 5.4, 5.5, 5.7, 5.8 and 5.9 show that HOPLoP significantly outperform baseline KG embedding models and SOTA multi-hop LP algorithms in both entity and relation prediction tasks. Table 5.10 shows that HOPLoP is computationally inexpensive compared to SOTA multi-hop LP algorithms. Chapter 6 provides a method to interpret HOPLoP's reasoning paths. Chapter 7 concludes the paper by providing a brief summary of our hypothesis and findings. We also provide a non-exhaustive list of future research directions.

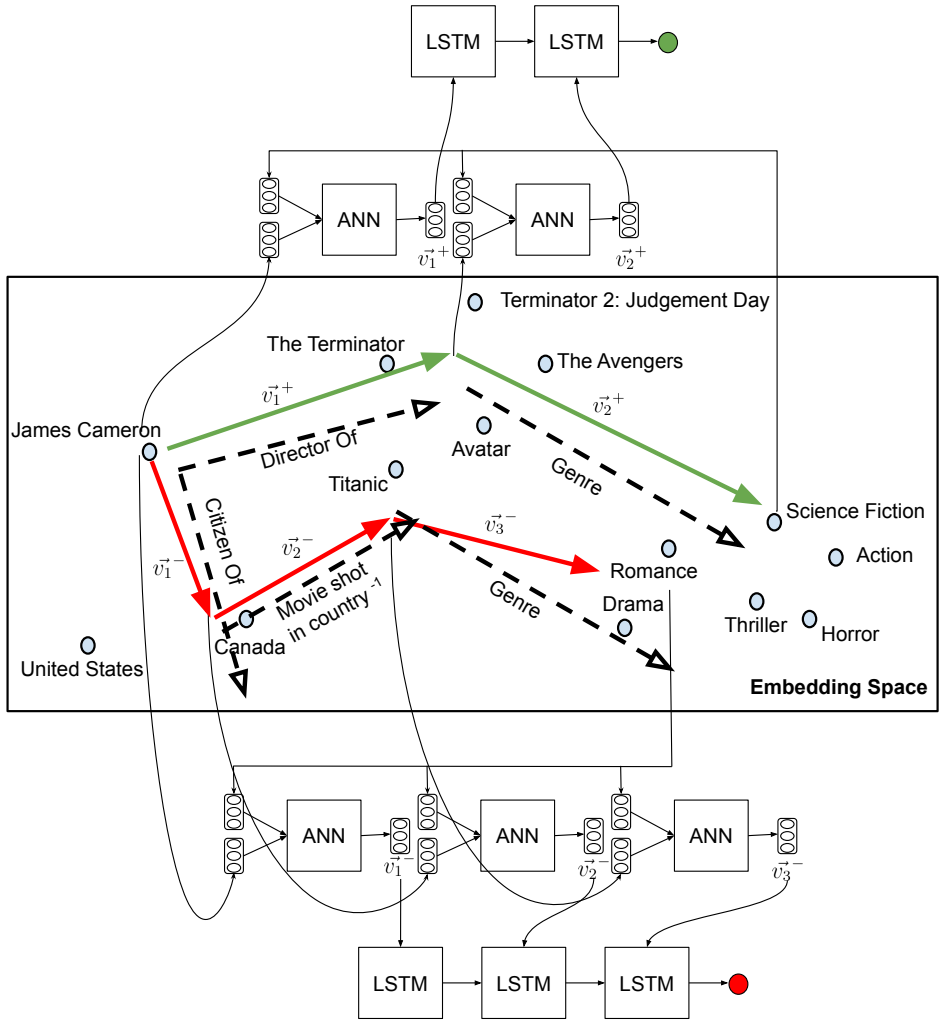


Figure 1.3: (Best viewed in color) Illustration of how a trained HOPLoP may answer the query "What genre of movies does James Cameron direct?". The position of entities in the figure are representative of their position in a TransE-like embedding space (refer 2.6.2). The black dashed lines are vector representations for relations, which *links* 2 entities in embedded space. To answer this query, HOPLoP would take a list of triples $[(\texttt{JamesCameron}, \texttt{DirectsGenreOfMovies}, e_i) \forall e_i \in$ entities in the domain of genre $\mathcal{E}_{Genre}]$ to provide a list of scores corresponding to input list. These scores indicate the probabilities of the corresponding link to exists in the KG. Here, we show 2 examples of paths HOPLoP takes: Green arrows signify the hops of the path traversed by the path-finder that enables the path-reasoner to support the link between `JamesCameron` and `ScienceFiction` (positive pair of entities / positive example). Red arrows signify path traversed by the path-finder that does not support the link between `JamesCameron` and `Romance` (negative pair of entities / negative example). $\vec{v}_i^{(\cdot)}$ is a vector in the embedded space where i indicates the hop number and $(\cdot)$ represents the type of example. HOPLoP learns to traverse an inaccurate representation of the KG to accurately distinguish between positive and negative examples. HOPLoP's reasoning path can be interpreted using vector similarity metrics.

Chapter 2

Preliminaries

2.1 Knowledge Graphs

Knowledge Graphs (KGs) are flexible data models that can represent structured and unstructured data seamlessly. For our purposes here we will define such graphs in terms of a set of entities ($\mathcal{E}$) and a set of relation names ($\mathcal{R}$).

Definition 1 *A Knowledge Graph is a labeled, directed graph* $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{L})$ *consisting of set of entities* $\mathcal{E}$ *which correspond to unique objects, a set of relations* $\mathcal{R}$ *which are labels applied to links and a set of links* $\mathcal{L} \subseteq \mathcal{E} \times \mathcal{E} \times \mathcal{R}$ *which connect one source entity* $e_s \in \mathcal{E}$ *to a target entity* $e_t \in \mathcal{E}$ *and assigns a label* $r \in \mathcal{R}$ *to that link, expressed as a triple* (e_s, r, e_t).

In the wide literature of graphs, entities are often referred to as "nodes" or "vertices" and links are also called "edges". In the literature of link prediction, the source entity is often referred to as the "head", "subject" or "query" entity, while the target entity is also referred to as the "object" or "tail" entity. The set of relations is also referred to as a set of "predicates".

Each link in the graph represents a fact, which is commonly expressed in natural language. Following our previous example, an assertive sentence such as, "James Cameron directed the movie Avatar", expresses a fact, whose link in the KG is represented as triple `(JamesCameron, DirectorOf, Avatar)`. Similarly, the sen-

tence, "Avatar's genre is Science Fiction", is represented as a link `(Avatar, Genre, ScienceFiction)`.

KGs represent facts of the world in a format usable by humans and machines [14]. They organise and integrate data from multiple sources, based on an ontology [1], to capture information about entities of interest in a given domain or task (e.g. people, places or events), and forge connections between them. To remain current, KGs need to be constantly updated with new facts. This is done through two approaches:

1. **Knowledge Base Population (KBP)** [33] involves identifying new entities and their properties from raw unstructured data. Note that properties are represented as relations in the KG whereas attributes are represented as entities in the KG. This is consistent with our KG definition and provides preliminary links to the KG, which is *completed* by a LP system. In our previous example, `JamesCameron` is the entity, while `ScienceFiction` is an attribute, represented by another entity in the KG; `DirectorOf` and `Genre` are properties of entities that connect entities to attributes. KBP can be done manually by human experts or through crowd-sourcing platforms, but arriving at high quality annotated data is expensive. This task can be automated using simple dependency parsers to SOTA Natural Language Processing (NLP) techniques [2].

2. **Knowledge Base Completion (KBC)** [16, 24, 34] involves finding relationships, general or specific, that *link* entities that are already present in the KG. This is done using experts who manually develop rules to perform a task, or by a *reasoner* [3] that looks at the links available in the KG to recognize patterns and derive new links and rules. For example, if the reasoner knows that `JamesCameron` directed movies {`Terminator, Avatar, ...`} and the genre of

[1] Ontology refers to the schema of the KG, which defines the ground truth knowledge and rules to derive more knowledge

[2] https://tac.nist.gov/2020/index.html

[3] This can range from a rule-based static system like Rule-Based Expert Systems [35] to DeepQA [36] which powers IBM Watson.

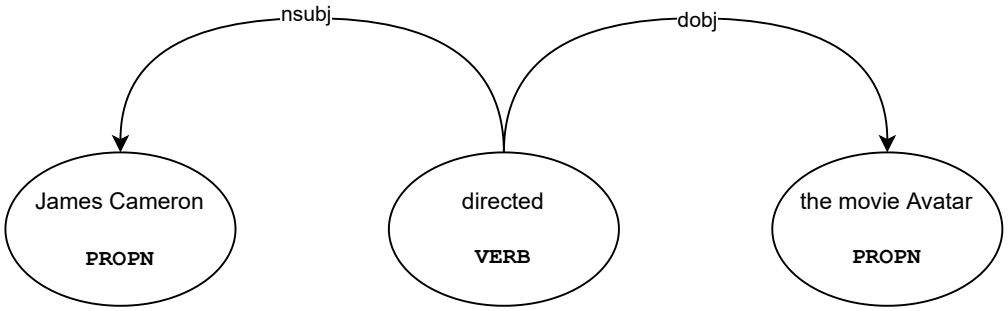


Figure 2.1: Dependency graph of our example sentence "James Cameron directed the movie Avatar" as generated by displaCy Dependency Visualizer [5]

these movies is `ScienceFiction`, then it can derive a new link: (`JamesCameron`, `DirectsGenreOfMovies`, `ScienceFiction`).

In figure 2.1, we show how a KBP system may populate the KG. Observe the dependency graph generated by spaCy [4], an open-source NLP package, for the sentence "James Cameron directed the movie Avatar". We see that the proper noun (`PROPN`) "James Cameron" is the subject, the proper noun "the movie Avatar" is the object, and the relationship between the subject and object is the predicate (`VERB`) "directed". By observing the pattern that the sentence contains a predicate which *depends* on a subject and an object, the KBP can come to the conclusion that a simple fact is being expressed, populating the KG with the link `(JamesCameron, Directed, Avatar)`. The ontology may include, implicitly or explicitly, synonymy rules such as $\texttt{DirectorOf}^{-1} \Leftrightarrow \texttt{Directed}$, where $\Leftrightarrow$ represents a equivalence relationship.

ML techniques are generally complemented by KGs to facilitate explainability [10] and encode domain knowledge that would be costly to learn from data alone. In this thesis, we look to go the other way: using ML to improve KBC, in terms of accuracy and efficiency, by surveying the KBC research literature to spot for successes and areas to improve. We tackle the problem of KBC through the LP task. The *reasoner* of an ontology performs LP to derive and populate the KG with new knowledge.

[4] https://spacy.io/

[5] https://explosion.ai/demos/displacy?text=James%20Cameron%20directed%20the%20movie%20Avatar&model=en_core_web_sm&cpu=1&cph=1

2.1.1 The NELL System

The Never-Ending Language Learning (NELL) project [37] aims at building a system capable of learning both facts and ontological rules by continuously reading the Web. Every cycle in NELL starts with a *current* version of the NELL KG, consisting of facts and inference rules (including multi-hop inference rules), and ends in a *new* version of the KG, built taking into consideration new facts and inferences that may not have been discovered before. While NELL does extract triples as in other KGs, technically, the endpoints of those triples are string literals instead of unique entity identifiers. Nevertheless, datasets originating from the NELL project are very often used in LP and KBC research.

2.2 Link Prediction

Given any link, represented by triple (e_s, r, e_t), Link Prediction (LP) aims at producing a score, indicative of the *likelihood* that link exists in the KG. The input to a LP system is the source entity e_s, which produces a list of "all possible" links $[(e_s, r_i, e_i) \forall r_i \in \mathcal{R}, e_i \in \mathcal{E}]$ ranking them from highest to lowest likelihood. Surveying the literature uncovers 2 variants of the LP task:

1. **Relation prediction** The relation prediction task focuses on predicting the relationship r that exists between a source entity e_s and a target entity e_t. Given an incomplete triple $(e_s, ?, e_t)$, relation prediction aims to fill in the gap in the triple, such that the resulting link exists in the KG. The question, "How is James Cameron related to Avatar?", involves predicting a relationship between two entities: `JamesCameron` and `Avatar`. In this query, the goal is to predict the relation of the incomplete triple (`JamesCameron`, ?, `Avatar`), such that resultant link is exists in the KG. For the given query, a trained relation prediction system should rank the relation `DirectorOf` higher than the relation `ActedIn`.

2. **Entity prediction** Given a source entity e_s and a relation of interest r, entity prediction involves predicting target entities e_t such that the link (e_s, r, e_t) exists in the KG. For example, consider the question: "Who directed the movie Avatar?". This question involves one entity `Avatar` and one relation $\texttt{DirectorOf}^{-1}$, forming an incomplete triple (`Avatar`, $\texttt{DirectorOf}^{-1}$, ?). Now, the aim is to search for an entity, to fill up the '?' in the triple, such that the resultant fact about the world is captured in the KG. Given a trained entity prediction system, we should expect the correct target entity `JamesCameron` to be ranked higher than an incorrect target entity such as `ZoeSaldana`, who was an actress in the movie.

HOPLoP does both! Our work focuses on the latter problem but we will provide a method to perform both entity and relation prediction using HOPLoP. In the coming sections, we further expand on LP and provide a literature survey on multi-hop approaches for LP. In the next sub-sections, we aim to provide pre-cursor knowledge on ML, required to understand the remainder of this thesis. We introduce KG embeddings, relate them to NNs, and will describe a few variants that we use in experiments. Motivated readers may refer to [14, 19, 24, 38] for extensive surveys on LP methods applied on KGs.

2.2.1 Link Prediction to Knowledge Base Completion

How can we use LP, especially multi-hop LP, to fill the gaps in the KG? Amongst several approaches, here are a few:

1. Pick the top-K of predicted links and add them to the KG, or ignore if already present. K would need to be a hyperparameter, which would need tuning.

2. Applying post-processing techniques by finding a score threshold, tuned for a requirement such as precision, recall, accuracy, etc.

3. Keep the probabilities and use the probabilities in downstream tasks.

4. Since multi-hop algorithm traverse paths in the KG, we can use the highly supported paths as rules to deduce new links.

2.3 Machine Learning

Machine Learning (ML) is the sub-branch of Artificial Intelligence (AI) which studies an algorithm's performance on a task taking into account the experience gained, by the algorithm, from data. ML algorithms aim to model a *population* [6] based on a subset of data from that population, often known as training data. Once the population of data-points is *modeled*, predictions and decisions can be made about unseen data-points, without explicitly programming for them. These decisions can be made autonomously, in real-time, to execute a specific set of instructions, emulating an "intelligent" software.

Before making predictions, ML algorithms first *learn*, through a process called *training*, to recognize patterns within the inputs that can be leveraged to produce an output that satisfies an objective. During the training phase, ML algorithms look at samples from the training data and gain experience by updating their mathematical model of the population based on an *objective* function, which captures the goal that needs to be achieved. Due to the stochastic nature of ML [7], there does not exist one ultimate algorithm that can solve all prediction problems. Therefore, practioners, through several iterations, develop several ML algorithms that tackle the same task. To select one model for deployment, practioners need a fair and objective comparison of all algorithms. Between the training and deployment phases, there is a testing phase, which evaluates and compares the performance of multiple ML algorithms de-

[6] In statistics, a population is a set of similar items or events which is of interest for some question or experiment. In our case, the population of links representing facts, is the KG, in which, each data-point in any sample of a KG is a link. https://en.wikipedia.org/wiki/Statistical_population

[7] Many aspects of ML, such as initial parameter values and shuffling for batching, rely on the randomness of numbers.

veloped for a specific task on specific datasets. Selection of the model for deployment is based on the predictions that model makes for a *test* split of the data available at development time, which is hidden from the model during training. Relevant metrics are calculated to measure and compare the performance of each algorithm.

HOPLoP is a framework for KBC which uses ML to discover new links in the KG. In the coming sections, we discuss all the pre-requisite concepts required for the future chapters, including a brief overview of the taxonomy of ML algorithms, GD, BP and Deep Learning (DL) based neural architectures used by HOPLoP. At the end of this chapter, we provide a discussion about KG embeddings as well as describe the KG embedding models used in our experiments.

2.3.1 Supervised Learning

Supervised Learning (SL) is a type of ML in which the goal is to find a mathematical function that best maps an input to the required output based on input-output pairs contained in the training data. A major distinction from other types of ML is that SL requires labeled data, i.e., data for which all inputs have corresponding outputs. During the training phase, both the input and the required output must be provided to the SL algorithm, which updates the underlying mathematical model such that it improves at mapping that input to that required output. The most common SL approaches use GD to update the parameters that make up the underlying mathematical function, which we will delve deeper in the coming sections. In the next chapter (refer to section 3.1), we discuss about SL approaches for multi-hop LP.

2.3.2 Unsupervised Learning

Unsupervised Learning (UL) [39] is a class of ML algorithms that learn to find patterns from unlabeled data. UL algorithms aim to create a mathematical function that maps inputs to itself. By generating an identity mapping function, UL algorithms, through parameter tuning [20, 25, 40, 41], force their underlying mathematical model to build

a compact internal representation of the population such that the model generalizes [8] to unseen examples. UL concentrates on clustering (e.g. image segmentation), anomaly detection (e.g. credit card fraud detection), and dimensionality reduction (e.g. visualizing higher dimensional vectors in 2D or 3D space).

Representation Learning

Representation Learning (RepL), or Self-Supervised Learning (SSL), is a sub-branch of UL that seeks to obtain latent representations for objects based on interactions with other objects. This is similar in the sense that UL and SSL algorithms aims to find different representations for objects. However, SSL aims to prepare the model by *pre-training* it for a general task. These models can be *fine-tuned* for specific downstream tasks ranging from predicting properties of data instances (e.g. sentiment analysis) to generate new data instances that mimic the population (e.g. document summarization), a technique used for data augmentation [42]. The general task usually involves predicting missing parts of the input. For example, consider the Transformer [43] model, applied on text [44] and images [45]. The general task to train these transformer models is using SSL, where parts of the inputs are *masked* [9] and the model is made to guess the missing portion. This model is then fine-tuned by re-training on domain or task specific data, allowing the model to adapt well to a specific task, such as Natural Language Generation (NLG) [46], without having to train it on large corpuses from scratch.

A similar approach has been followed by the KG embedding research community, which uses RepL techniques to generate embeddings for components of a KG. Instead of predicting for the hidden or *masked* parts in the input, KG embedding models are tasked to differentiate between "positive" links that are present in the KG and "negative" links that have been generated, through randomized procedures [47]. This

[8] Generalization refers to the model's ability to adapt properly to new, previously unseen data, drawn from the same population as the one used to create the model. https://developers.google.com/machine-learning/crash-course/generalization/

[9] With respect to language, a few tokens are left "blank" `[MASK]` in the input

differentiation is generally done through classification or *learning to rank* approaches. In the last section in this chapter, we discuss KG embeddings in detail and describe the different baselines used in HOPLoP experiments.

2.3.3 Reinforcement Learning

Reinforcement Learning (RL) is a type of ML where the algorithm is provided feedback in the form of rewards. RL algorithms aim to maximize the cumulative rewards it receives from an environment. Given a mathematical function that observes the environment and outputs an action, RL algorithms optimize their underlying mathematical model to, sequentially, take actions that would maximize the cumulative rewards received at the end of the episode. RL is similar to SL in the sense that both types of algorithms require a feedback from the environment. However, in SL, this feedback is immediately generated by an objective function whereas, in RL, the feedback, in the form of a reward, is received at the end of an *episode* [10].

In the context of multi-hop LP, RL, in recent times, has received special attention, since it has helped mitigate skewed node degree issues, enabling multi-hop LP algorithms to efficiently traverse large and noisy KGs [48–50]. In the next chapter, we will dive deeper into multi-hop approaches for LP. Specifically, we discuss both SL and RL approaches for multi-hop LP, highlighting modeling improvements in the literature, such as the switch from SL to RL for multi-hop reasoning.

2.4 Gradient Descent

Gradient Descent (GD) [29] is a simple algorithm at the core of many artificially intelligent systems [51]. GD is used to *train* the underlying mathematical model of several ML algorithms to perform a task, which is defined in terms of an *objective*

[10] An episode is one sequence of states, actions and rewards, ending at a terminal state. State: Describes the current situation the RL algorithm is in, whose *actions*, within that environment, is determined by a mathematical model.

function. In ML, we aim to create a predictive function. GD allows us to *find* [11] a function that performs the required task, such as input-output mapping, which may be expressed by a mean square error function, depending on the task. We will refer to the term "objective function" as a function to be *optimized*, and refer to the term "error function", "cost function" or "loss function" as a function to be *minimized*. Note that *optimized* could refer to the objective of *maximizing*, as done in RL, or *minimizing*, as done in SL. Given a dataset containing examples of input and output pairs, a parameterized differentiable function and an objective, expressed as an error function, GD attempts to find optimal parameter values that satisfies the objective, by reducing the error. Before delving further into GD, we shall explain what gradients are.

Derivatives and their Gradients

A derivative of a differentiable function [12] is a function that takes a point in space as input to calculate the slope, or the gradient, of the function at that point. Throughout this thesis, when we refer to a *derivative* of a function, we refer to its partial derivative, as opposed to its total derivative [13]. Given any differentiable function $f(x)$, its derivative function with respect to the variable x is $f'(x) = \frac{\partial}{\partial x}(f(x))$ and its gradient at a point x_0 is $\nabla_x f(x_0) = \frac{\partial(f(x))}{\partial x}(x_0)$. The gradient is a vector representing the direction and rate of fastest increase of a function. If the gradient vector is non-zero, the direction of the gradient [14] is the direction in which the function increases most quickly from x_0, and the magnitude of the gradient is the rate of increase in that direction. GD uses this gradient vector to "move", or update, the parameter

[11] GD assumes that the function has a constant sequence of differentiable computations. This is usually determined by the practioner, although recent years have witnessed growth in Neural Architecture Search (NAS) [52] methods. GD tunes the parameter values of the function such that the function satisfies the objective.

[12] https://en.wikipedia.org/wiki/Differentiable_function

[13] The difference is that a partial derivative with respect to some variable x treats all other variables as constants https://en.wikipedia.org/wiki/Total_derivative

[14] The direction of a vector can be visualized as a line starting from the origin $0 \in \mathbb{R}^D$ to the point represented by the gradient vector $\nabla_x f(x_0) \in \mathbb{R}^D$

values of the model towards a local minima of an error function in the direction of steepest descent.

2.4.1 Regression using Gradient Descent

Regression refers to a function that maps inputs to corresponding required outputs. During the training phase, the parameters of the regression function are tuned using GD. Given a dataset for SL, a mapping or regression function and an objective, we can proceed to tune the parameters of the function using GD. Detailed description of GD and it's variants is available in Appendix A.1. The GD approach for training regression models proceeds as follows:

Algorithm 2 *Training a regression model using GD*

Input: *Dataset* $D = \{..., (x_i, y_i), ...\}$, *Regression function* F, *Error function* E, *Learning rate* μ, *Initial Parameters* $\Theta^{(t=0)}$

$\Theta^{(t)} \leftarrow \Theta^{(t=0)}$;

Repeat:

$(x_i, y_i) \sim D$; ▷ *Sample one data-point*

$\nabla_\Theta E(F(x_i, \Theta^{(t)}), y_i) \leftarrow \frac{\partial(E(F(x_i, \Theta^{(t)}), y_i))}{\partial \Theta}(\Theta^{(t)})$; ▷ *Calculate gradients*

$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \mu \times \nabla_\Theta E(F(x_i, \Theta^{(t)}), y_i)$; ▷ *Update parameters*

Until *convergence;* ▷ *Time-step increments by 1*

Return $\Theta^{(t)}$;

Linear Regression

The *Linear* Regression (LR) is a simple regression model, which captures the linear relationship between the inputs and the outputs. A LR function is parameterized by a vector of weights; one weight parameter for each feature in the input and one bias parameter. The LR function can be given as follows:

$$
\begin{aligned}
LR(x, \Theta) &= \Theta_0 + \sum_{d=1}^{D} x_d \cdot \Theta_d \\
&= \sum_{d=0}^{D} x_d \cdot \Theta_d \\
&= \vec{x} \cdot \vec{\Theta}
\end{aligned}
\tag{2.1}
$$

where x is an input vector with $x_0 = 1$, D is the input dimensionality, and Θ_0 represents the bias term and $\Theta_{1..D}$ is the vector of weights for LR. To train a LR function, the most common choice for the error function is the mean squared error, given as follows:

$$
MSE(\hat{y}, y) = \frac{1}{N} \sum_{d=1}^{D'} (\hat{y}_d - y_d)^2 \tag{2.2}
$$

where N is the size of the dataset, D' is the output dimensionality, y is required output vector and $\hat{y} = LR(x, \Theta)$ is the predicted output vector.

Logistic Regression

Logistic regression (LogR) is similar to LR, which has been modified to fit the classification setting. The aim is to model a probability function, which outputs values between 0 and 1, to predict whether a data-point belongs to a particular class of objects. Similar to LR, the LogR function can also be parameterized by a vector of weights and is given as follows:

$$
LogR(x, \Theta) = \sigma(LR(x, \Theta)) \tag{2.3}
$$

where σ represents the sigmoid function that *squashes* the result from LR between 0 and 1. To train a LogR function for classification, a common choice for the error function is the binary cross-entropy function, given as follows:

$$
BCE(\hat{y}, y) = -\frac{1}{N} \sum_{d=1}^{D'} y_d \ln \hat{y}_d + (1 - y_d) \ln (1 - \hat{y}_d) \tag{2.4}
$$

where ln represents the natural logarithm and $\hat{y}_d = LogR(x, \Theta)$ is the predicted, generally probability value for the input vector. Although the MSE function can

be used for the classification setup, it often leads to poor convergence results. This is because the MSE function applied on a "sigmoid-squashed" function [15] leads to exponentially, with respect to number of features, many local minimas [53].

2.5 Backpropagation and Deep Learning

The current Deep Learning (DL) paradigm employ neural architectures, in conjunction with optimization objectives and a learning algorithm [61], to achieve SOTA performance on various tasks [62]. The workhorse of this paradigm is the Back-Propagation (BP) algorithm [63], which can be viewed as GD with effective caching [29], applied on a chain of differentiable operations. A major advantage of using the DL paradigm is automatic feature engineering, i.e., the ML algorithm can automatically extract important features from raw inputs such as images or text. In this work, when we refer to "backpropagation", we refer to the GD-based Backpropagation algorithm [63], as opposed to the Zeroth Order Relaxed Backpropagation (ZORB) [64], which does not require gradients, loss function selection and hyperparameter tuning to efficiently train neural architectures [17]. Before delving into GD-based BP, we need to introduce NNs.

2.5.1 Neural Networks

Neural Networks (NNs), based on a collection of connected units or nodes called artificial neurons [61], are differentiable mathematical functions capable of modeling complex, non-linear relationships between the input and target features. Non-linearity is achieved through the use of non-linear activation functions at each layer, which projects an input to a higher dimensional non-linear space. A neural network is termed "deep" if it has more than 1 hidden layers where the layers may be customized

[15] Output values for sigmoid-squashed functions are in the range $(0, 1)$.

[16] Slope of a line is given by $tan(\Theta)$ where Θ is the angle between the error axis and the gradient direction. https://en.wikipedia.org/wiki/Slope

[17] In theory, HOPLoP should be compatible with ZORB, but due to ZORB's immaturity (https://github.com/varunranga/zorb), we do not experiment HOPLoP on this algorithm.

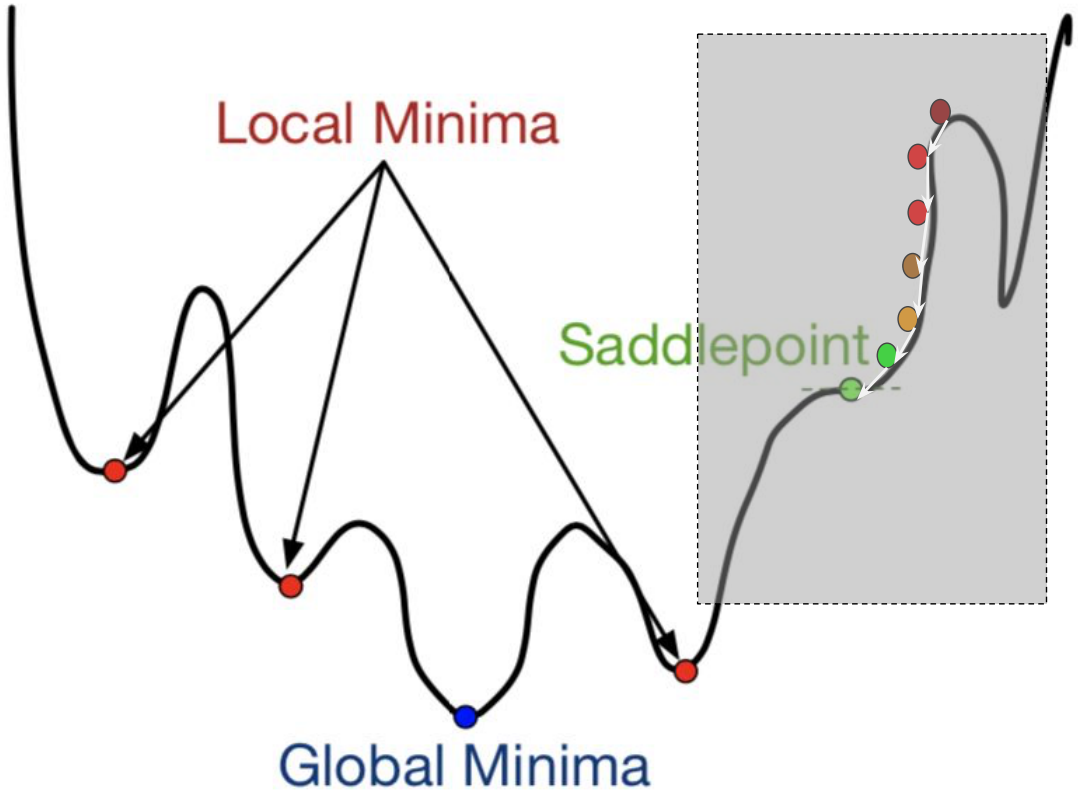


Figure 2.2: (Best viewed in color) Illustration of a non-convex error function surface; image taken from [53]. GD moves the parameter values towards the local minima of the error function. As the arrows indicate, the weights are updated such that the error function moves towards the local minima. The ball represents the error caused by weights at that time. The color gradient from red to yellow represents the change in time step, i.e., time moves forward as the color of the ball changes from red to yellow. The color of the ball also represents the gradient at that time. Usually, we observe that as the system is trained more, naturally consuming more time, the gradients decrease. Darker colors signify high gradient values and lighter colors signify low gradient values. GD can be susceptible to undesirable stationary points, such as saddle points. Saddle points are analogous to plateaus on the error surface. As seen in the figure, the slope at the saddle point, i.e. the gradient, is $tan(0) = 0$ [16], which does not provide GD with information about the curvature of the error function or the direction to move the weights towards. This inhibits vanilla GD from finding a local minima. Several modifications have been proposed to the update rule [54–60], in order to speed up or stabilize training.

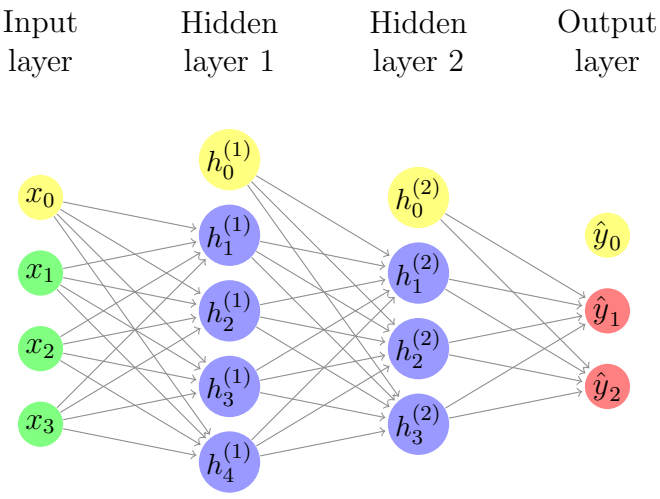


Figure 2.3: (Best viewed in color) Illustration of a Fully-Connected Neural Network (FCNN) with 3 input neurons and 2 hidden layers, each non-linearly activated with 4 and 3 neurons respectively. The output layer contains 2 output neurons. Yellow nodes represent the '1' input for the bias term; Green nodes represent real-valued inputs; Purple nodes represent hidden neurons; Red nodes represent output neurons; Each directed edge represent a weight between 2 neurons. We maintain this color scheme throughout.

for specific tasks, such as Recurrence for sequences (see Section 2.5.2, Convolutions for images [65, 66]). In this work, we term a single-(hidden)-layered fully-connected neural network as an Artificial Neural Network (ANN) [18]. The performance of neural networks are backed by the Universal Approximation Theorem [67] and the development of parallelized computation.

To intuit BP, let us consider a simple feed-forward neural network with H hidden layers [19], each layer l having U_l neurons, each layer activated by function f_l, which accepts input $x_i = [..., x_{ij}, ...] \in X$ is an input vector and output $y_i = [..., y_{ij}, ...] \in Y$ is a required output vector. This neural network function can be represented as:

[18] This will result in H hidden layers and 1 output layer to a total number of layers $L = H + 1$.
[19] See footnote 18.

$$
\begin{aligned}
NN(X) &= f_L(h_{L-1}W_L) \\
&= f_L(f_{L-1}(h_{L-2}W_{L-1})W_L) \\
&= ... \\
&= f_L(f_{L-1}(...(f_1(XW_1)W_2)...W_{L-1})W_L)
\end{aligned}
\tag{2.5}
$$

where $W_l \in \mathbb{R}^{(U_{l-1}+1)\times U_l}$ are matrices representing a neural layer and $U_0 = D$ is the dimension of each input vector. For simplicity, we model the bias term by adding a row of weights in weight matrices W_l and concatenating the input with a column of 1s. We refer to each W_l as a "neural layer".

Alternatively, we need not explicitly model the bias term. Through normalization [68], the bias term is eliminated (see Appendix A.5). This comes at the cost of computation, since normalization of data before a layer is more expensive ($\mathcal{O}(NM), N$: data batch size, M : number of features), than addition of the bias vector ($\mathcal{O}(M)$), or multiplication of a vector with ones ($\mathcal{O}(1)$).

Backpropagation Algorithm

Although GD can be applied to any differentiable function, it is very inefficient if the function is complex. This inefficiency arises from repeated gradient calculations in each update step. The Back-Propagation (BP) algorithm [63] extends GD for a computational graph [20]. The idea involves storing the "partial gradients" at each layer to be reused while calculating the gradients for other layers. We present the derivation of the BP algorithm on a simple feed-forward neural network in appendix A.6.

In general, for most deep neural architecture, the partial gradient δ_L for the output layer L is:

[20] https://www.tutorialspoint.com/python_deep_learning/python_deep_learning_computational_graphs.htm

$$\begin{aligned}\delta_L &\propto (NN(X) - Y) \qquad \text{(error made)} \\ &\propto \frac{\partial}{\partial O_L} f_L(O_L) \qquad \text{(derivative of activation function at output layer)}\end{aligned} \tag{2.6}$$

and the partial gradient δ_l for a hidden layer l is:

$$\begin{aligned}\delta_l &\propto \delta_{l+1} \qquad \text{(partial gradient from upper layer)} \\ &\propto W_{l+1}^T \qquad \text{(transpose of weight matrix of top layer)} \\ &\propto \frac{\partial}{\partial O_l} f_l(O_l) \qquad \text{(derivative of activation function at current layer)}\end{aligned} \tag{2.7}$$

To arrive at the actual gradient, we need to multiply the partial gradients δ_l of a layer l with the input to that layer h_{l-1}, which is already computed during the *forward propagation* stage. Upon obtaining the gradient, GD is used to update the parameters. See appendix A for a detailed derivation of BP and the derivatives of activation functions.

2.5.2 Recurrent Neural Networks and Backprop Through Time

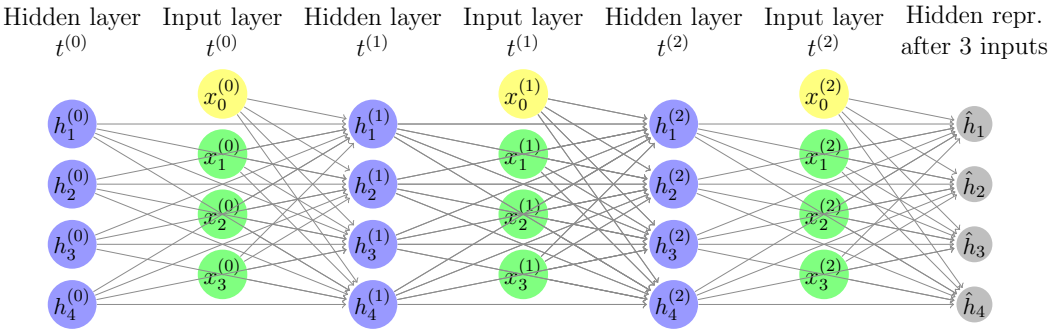


Figure 2.4: (Best viewed in color) Illustration of a vanilla RNN, containing 3 input neurons and 4 hidden neurons, unrolled upto 3 time-steps. Gray nodes represent the hidden state representation of the RNN after processing the input sequence of vectors. In this diagram, the number of hidden units is 4 and the input dimension at each timestep is 3. At each time step, the concatenation of the input vector and the hidden state representation is used to generate a new hidden state representation.

Recurrent Neural Networks (RNNs) [63, 69] are a class of deep neural networks that learn to model temporal dynamic behaviour by utilizing an internal memory component that stores the state of the network. In vanilla RNNs, as illustrated in figure 2.4, a neural layer, activated by the tanh function, models the temporal dynamic behaviour of a sequence of data-points in the dataset. At each time-step, the RNN is provided with an input vector $x_i^{(t)}$, which is concatenated with the current hidden state of the RNN, represented as h^{t-1}, and sent to the tanh-activated neural layer to form a new hidden state:

$$h^{(t)} = tanh(W_h \times [x_i^{(t)}; h^{(t-1)}; 1]) \tag{2.8}$$

where $[.;.]$ signifies the concatenation operation. This hidden state vector $h^{(t)}$ can be viewed as an embedding that represents the sequence of vectors $[x_i^{(1)}, ..., x_i^{(t)}]$. At each step, $h^{(t)}$ may be used to generate an output $y_i^{(t)}$ at time-step t, using another neural layer W_o:

$$y_i^{(t)} = W_o \times [h^{(t)}; 1] \tag{2.9}$$

For optimization purposes, RNNs are often unrolled into a feed-forward NN during implementation. For example, assuming a maximum sequence length of T, a RNN is unrolled to a feed-forward NN with T hidden layers This feed-forward NN takes input $[x_i^{(1)}; h^{(0)} = [0 \in \mathbb{R}^D]]$ and outputs the final representation of the sequence of input vectors $h^{(T)} \in \mathbb{R}^D$. Weights W_h are shared at each hidden layer l and input $x_i^{(l)}$ is concatenated with the hidden representation of the sequence $h^{(l-1)}$ to produce a new hidden representation of the sequence $h^{(l)}$. RNNs are optimized using the Back-Propagation Through Time (BPTT) algorithm, which can simply be viewed as the GD-based BP algorithm applied on an unrolled RNN.

Due to problems such as vanishing and exploding gradients [70], vanilla RNNs have been superseded by gated networks such as Long Short Term Memory (LSTM) [71]

and Gated Recurrent Unit (GRU) [72]. Gates allow recurrent networks to "choose" the information that should be add to or taken from the hidden state representation. LSTM networks, which has been successfully applied for a variety of sequence related problems, learn to ignore noisy temporal patterns while paying a soft-attention to important patterns of the sequence, making them more powerful that vanilla RNNs. For this reason, we parameterize HOPLoP with an LSTM, but conceptually, a vanilla RNN would suffice. Figure 2.5 illustrates the connections within 1 time-step of an LSTM layer.

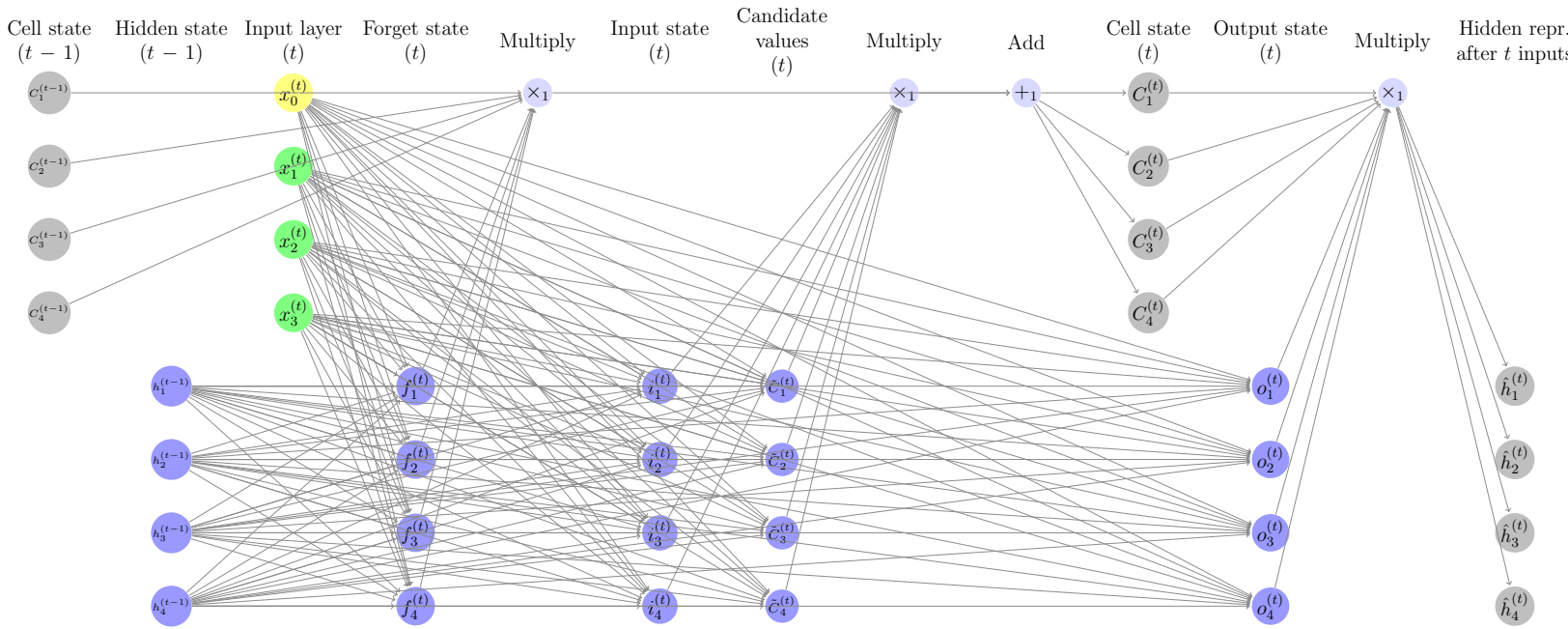


Figure 2.5: (Best viewed in color) Illustration of the connections in 1 time-step of a LSTM layer. C_d refers to the cell state, f_d denote neurons for the forget gate, i_d denote neurons for input gate, $\tilde{C}_d$ denote neurons for the candidate input values, and o_d represent neurons for the output value. For a comprehensive explanation of LSTMs, please refer to https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

2.6 Knowledge Graph Embeddings

KG embedding algorithms are SSL algorithms that learn to map discrete objects, such as entities and relationships, to their corresponding vector representations. These algorithms aim to model the existence of a link in a KG. To avoid confusion, when we refer to KG embedding "model", we talk about the KG embedding algorithm's underlying mathematical scoring function [21]; when we refer to KG embedding "space", we talk about the space of tensors that captures the latent structure of the KG [24], distributed over the representations for each component in the KG. Given a link, represented by the triple (e_s, r, e_t), a KG embedding model takes as input $\vec{e_s}$, $\vec{r}$, $\vec{e_t}$, and computes a score representing the likelihood that the given link exists in the KG. This is a non-trivial task, since a KG contains only positive examples of links. This requires us to generate negative examples. For example, we can generate a negative link `(JamesCameron, DirectsGenreOfMovies, Comedy)` from the positive link `(JamesCameron, DirectsGenreOfMovies, ScienceFiction)` by replacing the target entity `ScienceFiction` with another target entity `Comedy`, such that the resultant link does not exist in the graph. A practitioner may generate a static set of negative examples, as done by Xiong *et al.* [48], or may generate negative examples while training the model, as done by Bordes *et al.* [21]. Intuitively, KG embedding models are trained to distinguish between positive and negative examples of links. By doing so, the gradients used to train the parameters of the scoring function are backpropagated to update representations for the components of a link. It can be seen that the structural information required to predict whether a link exists in the KG is distributed over all parameters of the KG embedding model [24]: parameters that represent each component in the KG and parameters for the scoring function. These representations can be later used to power downstream tasks such as question answering and automated reasoning, or even KBC as done by HOPLoP.

[21] This includes a mapping and an objective function

2.6.1 From the perspective of a neural network

A simple way to represent any object as a vector is through one-hot encoding. For example, let us assume `JamesCameron` is the 450^{th} entity discovered by the KBP system. The system has discovered a total of 15,000 entities. In the one-hot encoding scheme, `JamesCameron` would be represented as a sparse vector, in which the 450^{th} element is 1 while the rest of the elements are 0. Therefore,
$OneHotEncoding(\texttt{JamesCameron}) = \begin{bmatrix} 0 & ... & 0 & 1 & 0 & ... & ... & 0 \end{bmatrix}$ where the one-hot vector is in $\mathbb{R}^{1\times|\mathcal{E}|}$. One can quickly realize that the size of this vector scales linearly with the number of entities, i.e., the size of this vector will increase by 1 if the KBP system discovers a new entity. This exacerbates the curse of dimensionality [73], requiring more data-points to achieve optimal performance.

Taking inspiration from neural networks, a simple approach to reduce the dimensionality used to represent the object would be to pass the one-hot encoding through a neural layer consisting of number of neurons significantly lesser than the number of objects. This neural layer is represented as a matrix $W \in \mathbb{R}^{(|\mathcal{E}|+1)\times D}$ where $|\mathcal{E}|$ represents the number of entities, $+1$ signifies a bias vector, D is the final dimensionality of the vector representing the object. Continuing our `JamesCameron` example, this computation can be shown as follows:

$$Embedding(\texttt{JamesCameron}) = \begin{bmatrix} 0 \\ ... \\ 0 \\ 1 \\ 0 \\ ... \\ ... \\ 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} w_{(1,1)} & w_{(1,2)} & .... & w_{(1,D)} \\ ... & & & \\ w_{(449,1)} & w_{(449,2)} & .... & w_{(449,D)} \\ w_{(450,1)} & w_{(450,2)} & .... & w_{(450,D)} \\ w_{(451,1)} & w_{(451,2)} & .... & w_{(451,D)} \\ ... & & & \\ ... & & & \\ w_{(15000,1)} & w_{(15000,2)} & .... & w_{(15000,D)} \\ b_{(1)} & b_{(2)} & .... & b_{(D)} \end{bmatrix} \tag{2.10}$$

where $w_{(i,j)}$ and b_k is the weight and bias parameter values of the neural layer. It can be observed that a row i in the weight matrix W is a vector representing the entity i. Since the one-hot encoding for `JamesCameron` is a sparse vector, where all but one elements are 0, calculating the embedded representation for `JamesCameron` can be simplified:

$$Embedding(\texttt{JamesCameron}) = \begin{bmatrix} w_{(450,1)} & w_{(450,2)} & .... & w_{(450,D)} \end{bmatrix} + \begin{bmatrix} b_{(1)} & b_{(2)} & .... & b_{(D)} \end{bmatrix} \tag{2.11}$$

In the example, vector $\begin{bmatrix} w_{(450,1)} & w_{(450,2)} & .... & w_{(450,D)} \end{bmatrix}$ from the weight matrix partially represents the entity `JamesCameron`, while the vector $\begin{bmatrix} b_{(1)} & b_{(2)} & .... & b_{(D)} \end{bmatrix}$ represents the bias terms for each of D neurons in the neural layer. Therefore, instead of multiplying a one-hot vector with a matrix, which is computationally very expensive, neural layers such as the embedding lookup layer [22] extracts the row of a matrix given the row number, which is usually an ID used to reference the required object. To completely represent the entity `JamesCameron` via the vector $\begin{bmatrix} w_{(450,1)} & w_{(450,2)} & .... & w_{(450,D)} \end{bmatrix}$, we would like to eliminate the bias vector, i.e., $\begin{bmatrix} b_{(1)} & b_{(2)} & .... & b_{(D)} \end{bmatrix} = 0 \in \mathbb{R}^D$. As mentioned in the previous section, this is accomplished by $l2$-normalizing the rows of the weight matrix, which is followed by almost all KG embedding models. This implicitly models the bias, which is distributed amongst the weight parameters of the matrix. After $l2$-normalization, the resultant embedding for `JamesCameron` can be given as follows:

$$Embedding(\texttt{JamesCameron}) = \frac{\begin{bmatrix} w_{(450,1)} & w_{(450,2)} & .... & w_{(450,D)} \end{bmatrix}}{\sqrt{w^2_{(450,1)} + w^2_{(450,2)} + ... + w^2_{(450,D)}}} \tag{2.12}$$

By eliminating the bias term [23], we now arrive at an embedding for `JamesCameron`

[22] https://www.tensorflow.org/api_docs/python/tf/nn/embedding_lookup

[23] Alternatively, we can choose *not* to model the bias terms, but this new model without the bias terms can be viewed as a subset of the previous model with bias terms since, a function $f(x) = xW$ can be modeled by another function $g(x) = xW + b$, where $b = 0$. But, it is not possible to model function $g(x)$ using function $f(x)$ for any value $b \neq 0$, making $f(x)$ a subset of $g(x)$.

that fully describes the entity `JamesCameron`. By controlling the dimensionality D of the resultant embeddings, these models tend to generalize to unseen links of the KG [20]. The resultant embedded representation of the KG, modeled by the vector representations for components of the KG and the scoring function, is more-complete but less accurate than the discrete KG [17].

We believe that the performance in multi-hop LP can be improved if it is not restricted by the links in the discrete KG. To verify our hypothesis introduced in chapter 1, we use KG embedding models to create an embedded space that can be traversed by HOPLoP. We experiment with three KG embedding models: TransE [21], ComplEx [74], and TuckER [75].

2.6.2 TransE

Bordes *et al.* [21] introduced TransE, which is the first and the most competitive translational distance KG embedding model [76]. Given a link (e_s, r, e_t), TransE views the relation r as a vector which maps, via vector addition or translation, from the source entity e_s to the target entity e_t in the vectorial space, i.e., $\vec{e_t} \approx \vec{e_s} + \vec{r}$. The idea behind TransE is that, if we are at the source entity, expressed by point $\vec{e_s}$, and if were to move by $\vec{r}$, we should reach the required target entity, represented by $\vec{e_t} \approx \vec{e_s} + \vec{r}$. Therefore, the closer $\vec{e_s} + \vec{r}$ is to $\vec{e_t}$, it is more likely that the link (e_s, r, e_t) exists in the KG. The score for a link is given by: $-\Sigma_{d=1}^{D}(\vec{e_{s_d}} + \vec{r_d} - \vec{e_{t_d}})^2$ where d is the dimension of entity and relation embeddings and i represents the position of an element in the vector. Since $(\vec{e_{s_d}} + \vec{r_d} - \vec{e_{t_d}})$ represents a distance metric, the negative sign promotes the intuition that links scored or *ranked* higher are more likely to be present in the KG. During the training phase, TransE is given a positive link (e_s, r, e_t^+) and a negative link (e_s, r, e_t^-). TransE uses a margin-based loss function to update the embeddings for e_s, r, e_t^+, and e_t^- such that, $\vec{e_s} + \vec{r}$ is closer to the correct target entity $\vec{e_t^+}$ than the incorrect target entity $\vec{e_t^-}$. If the predicted score for a given positive link is higher, by a pre-defined margin γ, than the predicted score for the

corresponding negative link, the margin-based loss function returns 0, assuming no errors from TransE's perspective.

Given a relation r_{1-M} of type 1-M [24], one can realize that $\vec{e_s}+\vec{r_{1-M}}$ cannot represent multiple entities that are linked from e_s via the relation r_{1-M}. This can be viewed as a mode collapse [77, 78], where the vector $\vec{e_s} + \vec{r_{1-M}}$ represents the average of all correct target entities that complete the incomplete link $(e_s, r_{1-M}, ?)$, such that the link exists in the KG. Therefore, TransE does not perform very well on complex relation types [21].

Another underlying assumption made by TransE is that all entities and relations lie on the same plane. We use this to our advantage by allowing HOPLoP to traverse over the TransE embedding space, where both entities and relations are represented. We compare HOPLoP(TransE)'s translational vectors with vector representations of entities and relations in the same plane, allowing us to interpret it's traversed path (see Chapter 6).

2.6.3 ComplEx

A binary tensor representation of a KG is a 3D tensor in $\mathbb{R}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$, where the value at the any index (i, j, k) corresponds to the truth value of the link (e_i, r_j, e_k). $\mathcal{E}$ represents the set of entities, $\mathcal{R}$ represents the set of relations, $|.|$ represents the "size of" operator. The binary tensor representation of the KG is very large to store and continues to suffer from the incompleteness problem. Therefore, tensor factorization models [79], such as RESCAL [25] and DistMult [80], learn to decompose this binary tensor in an attempt to compress the KG while generalizing to unseen facts. Trouillon *et al.* [74] introduced ComplEx, which extends DistMult to improve asymmetric relation modelling. ComplEx represents each components of the KG using both real and complex valued parameters. The scoring function is based on the dot function for complex numbers [25]. The goal is to minimize the real part of the dot product of a

[24] Connects 1 source entity to (M)any target entities

[25] https://en.wikipedia.org/wiki/Dot_product#Complex_vectors

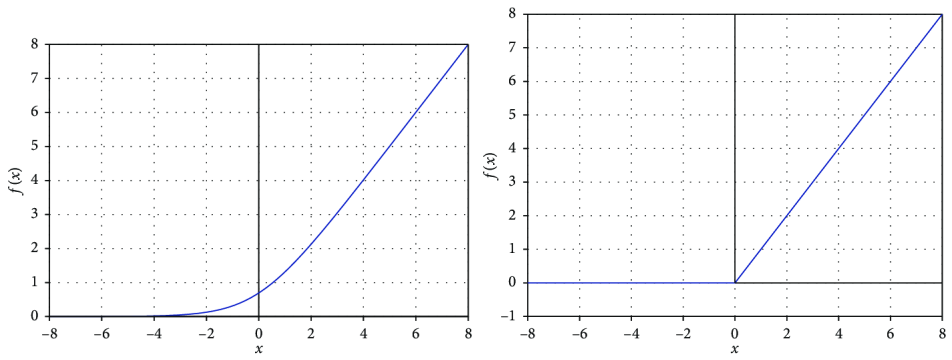


Figure 2.6: Softplus (Left) and ReLU (Right) function visualization taken from Aceves-Fernandez *et al.* [81].

embeddings for a link $\vec{e_s} \cdot \vec{r} \cdot \vec{e_t}$ such that, positive links result in a lower dot product value than negative links:

$$\begin{aligned} & E([\vec{e_s}^+; \vec{r}; \vec{e_t}^+], [\vec{e_s}^-; \vec{r}; \vec{e_t}^-]) \\ & = Softplus(-y \cdot score(e_s, r, e_t)) \end{aligned} \tag{2.13}$$

where E refers to the error function, $y \in \{+1, -1\}$ is the label assigned for the link and score is given by:

$$score(e_s, r, e_t) = Re(< \vec{e_s}, \vec{r}, \overline{\vec{e_t}} >) \tag{2.14}$$

where $\vec{(.)} := \vec{(.)}^+$ if $y = +1$ else $\vec{(.)} := \vec{(.)}^-$, $:=$ is the assignment operator and $\overline{\vec{e_t}}$ is the vector representing the complex conjugate of e_t.

The $Softplus$ function, often viewed as a "smooth"-Rectified Linear Unit (ReLU) [26], is: $Softplus(x) = ln(1 + exp(x))$ compared to $ReLU(x) = max(0, x)$.

The training process, via backpropagation, aims is to find embedding values, such that the error reduces. To reduce errors, BP will tune the parameters such that, scores for links tend towards $-\infty$. Entities represented in the real number space has different latent properties compared to entities represented in the complex number space. By enabling the representation of real and imaginary numbers, entities can

[26] https://software.intel.com/sites/products/documentation/doclib/daal/daal-user-and-reference-guides/daal_prog_guide/GUID-FAC73B9B-A597-4F7D-A5C4-46707E4A92A0.htm

behave differently based on their role (subject or object) in a fact. During the testing phase, the negative of the *score* is used to rank candidate entities for LP. Links that result in lower *score*s are considered more likely to be true.

2.6.4 TuckER

TuckER [75] is a fully expressive bilinear tensor factorization model based out of the Tucker decomposition [82]. It looks to decompose the binary tensor representation of the KG into a set of matrices and a small core tensor. The score for a link is computed using the embeddings for the link (e_s, r, e_t) and another 3D tensor that learns to combine the embeddings such that TuckER produces a higher score for positive links. The score function is given as:

$$score(e_s, r, e_t) = \mathcal{W} \times_1 \vec{e_s} \times_2 \vec{r} \times_3 \vec{e_t} \tag{2.15}$$

where $\mathcal{W}$ is the core tensor and $\times_i$ indicates the tensor product along the i^{th} mode. The score is then treated similar to the output of a multiple logistic regression model, i.e., a score for each possible target entity is squashed by the sigmoid function and TuckER minimizes the binary cross-entropy loss. TuckER is *fully expressive* [27] and achieves SOTA performance in LP [75].

[27] Given any ground truth over the triples, there exists an assignment of values to the entity and relation embeddings that accurately separates the true triples from false ones

Chapter 3

Related Work: Multi-hop algorithms for Link Prediction

3.1 Supervised Learning with PRA and its successors

Introduced by Lao *et al.* [23], the Path Ranking Algorithm (PRA) is the first to tackle multi-hop link prediction. It uses intuition from PageRank [83] which we explain briefly, since it forms the basis for random walking.

3.1.1 PageRank

PageRank is a simple yet highly efficient [1] algorithm based on a recursive definition:

$$\underset{t+1}{\Pr} = Q \times \underset{t}{\Pr} \tag{3.1}$$

where $\Pr_t \in \mathbb{R}^{|\mathcal{E}|}$ is the current probability distribution over entities at time-step t and Q is the transition matrix of the graph, which is a 2D matrix in $\mathbb{R}^{|\mathcal{E}|\times|\mathcal{E}|}$. In matrix Q, higher values signify a higher likelihood of transitioning from one node to another. At $t = 0$, $\Pr_{t=0}$ is a one-hot vector representing the source node [2]. Upon convergence,

[1] It is also known as the "trillion dollar algorithm" https://en.wikipedia.org/wiki/PageRank

[2] During implementation, the input vector need not be one-hot. A binary input vector may represent multiple source nodes. For didactic purposes, we proceed to explain PageRank with only one source nodes. However we can initialize the distribution over multiple source nodes, and, if the adjacency matrix is ergodic https://en.wikipedia.org/wiki/Ergodic_theory, the process always converges.

$\Pr_{t=\infty}$ represents a distribution over the set of target entities. To illustrate this, visualize a structure in which sinks behave as entities, and pipes that connect two sinks can be viewed as the links of the graph. PageRank can be understood by posing this question: "Consider you have sinks interconnected with pipes. If we pour 1 litre of water in a particular sink, how much water will spread to other sinks?". Similar to PageRank is the Random-Walk with Restart (RWR) which has one extra component called the "restart" probability. This component essentially indicates that for every step taken in any direction, there is a probability associated in going back to the initial starting position, which in our case is the source entity e_s. Equation 3.1 becomes:

$$\underset{t+1}{\Pr} = c(Q \times \underset{t}{\Pr}) + (1 - c)o_s \tag{3.2}$$

where $c \in (0, 1)$ is a hyperparameter that controls the probability of *restarting*, and o_s is the one hot encoded representation of the source entity e_s. The restart term makes sure that the PageRank distribution, at any time-step, is *not* distributed throughout the graph. This biases the process to explore only a neighbourhood of the source node.

3.1.2 Path Ranking Algorithm

Path Ranking Algorithm (PRA) tackles the entity prediction problem using a Random Walk with Restart (RWR)-based inference mechanism [22] that ranks related target entities. A single PRA model is trained to perform LP for a single relation of interest in the KG. To avoid confusion with other relations in the graph, we interchangeably call this relation of interest as our *task*. The set of relations of interest or tasks $\mathcal{K} \subseteq \mathcal{R}$ is a subset of the relations in $\mathcal{R}$. Given a set of source entities and pre-defined paths, PRA performs a "weighted random walk" over all paths. During the training phase, the weight associated with each path is tuned using GD, such that, paths that are more *likely* to reach the required target entities will be scored or ranked higher, in support of the task.

PRA uses bounded RWRs [3] to calculate the likelihood of reaching all entities after traversing all pre-defined paths. To explain the "traversal" [4] process, let's look at this example: Let $\mathcal{P}$ be a random variable [5], representing the set of paths traversed in the KG. Given a path $p \in \mathcal{P}$ and $p = \texttt{DirectorOf} \rightarrow \texttt{Genre}$, PRA calculates a distribution over all entities, such that, entities with high scores are more likely to be related with the source entity. To calculate this distribution, PRA explores the graph using RWR over each path. In our example, RWR requires the transition matrices Q_r for both relations `DirectorOf` and `Genre`:

$$\begin{aligned}
&\Pr_{t=0}(e_t|e_s, r, p) = o_s \\
&\Pr_{t=1}(e_t|e_s, r, p) = c(Q_{\texttt{DirectorOf}} \times \Pr_{t=0}(e_t|e_s, r, p)) + (1-c)o_s \\
&\Pr_{t=2}(e_t|e_s, r, p) = c(Q_{\texttt{Genre}} \times \Pr_{t=1}(e_t|e_s, r, p)) + (1-c)o_s
\end{aligned} \tag{3.3}$$

where $\Pr_{t=i}(e_t|e_s, r, p)$ is a distribution over all entities representing the probability of reaching any entity from source entities e_s at time-step $t = i$. At $t = 0$, $\Pr_{t=0} = o_s$ and at $t = H$ where H is the number of relations in the path, $\Pr_{t=H}(e_t|e_s, r, p)$ is the final distribution over all target entities e_t upon traversing the path p. Each pre-defined path is associated with a trainable likelihood parameter to perform a weighted averaging of the RWR probabilities for each path. Once RWR inference is performed on all paths, the resulting entity distributions are averaged by:

$$\Pr(e_t|e_s, r) = \sum_{p=0}^{|\mathcal{P}|} \Theta_p \times \Pr_{t=H_p}(e_t|e_s, r, \mathcal{P}_p) \tag{3.4}$$

where $\Pr_{t=H_p}(e_t|e_s, r, \mathcal{P}_p)$ represents a RWR distribution, after traversing path $\mathcal{P}_p$ of length H_p, over all entities in the graph and Θ_p represents the weight associated with path $\mathcal{P}_p$. During the training phase, the vector $\Pr(e_t|e_s, r)$, representing the

[3] RWRs are bounded by the relations in the path, i.e., the number of times the RWR inference mechanism 3.2 is applied is determined by the path length, and each RWR inference step depends on the relation to be traversed.

[4] Notice that, the RWR process, over all paths, can be pre-computed, emulating the act of traversal.

[5] https://en.wikipedia.org/wiki/Random_variable

distribution of links from source entities to the target entities via the relation r, is compared against a ground truth vector that contains the correct target entities. Through GD, PRA aims to reduce the binary cross-entropy error between the two vectors. The error gradients are backpropagated to tune the weights associated with each path. In the beginning, PRA would randomly traverse and expand nodes in the graph. As GD tunes the weight parameter for each path, PRA starts to prefer paths that have a higher probability of reaching the required target entities. At the end of the training process, PRA can perform LP by traversing all pre-defined paths and combining their RWR distributions in a weighted manner. The resultant vector gives a distribution over all entities representing the likelihood that the relation of interest r exists between the source entities and target entities.

The path weight can be viewed as a score to *rank* paths that support the task. Paths that are scored or ranked high can be used as a "reasoning" path, to search for equivalent paths and fill missing links. This provides an alternate approach to LP: gather highly ranked paths, as deemed by PRA, and search for these paths in the graph to predict the underlying relation.

Advantages PRA is an incredibly powerful reasoning tool, which learns to rank "reasoning paths" higher amongst a set of pre-defined paths. This is done through a supervised learning process: given source entities for a task and a set of pre-defined paths, predict the existence of the relation of interest between the queries entity and all other entities. Similar to training a classifier, PRA tunes the weights for each path such that, the weighted entity distribution *ranks* positive target entities [6] higher than negative target entities [7]. Since all processes takes place over a discrete space, the model is explainable. This was the first algorithm to promote explainable automated reasoning that could operate over large-scale KGs.

[6] The relation of interest exists between the source and "positive" target entity.

[7] Similarly, the relation of interest does not, currently, exist between the source and "negative" target entity in the KG.

Disadvantages Since PRA operates on the discrete KG, it is affected by the KG issues, specifically, skewed node degree caused by *supernodes*, incompleteness or missing links in the KG, semantically duplicate entities and relations [84, 85]. These issues inhibit the traversal process by either, prioritizing supernodes disproportionately and spuriously ranking them higher than correct target entities; or by, limiting the number of links that can be traversed; or by, treating similar things differently.

Differences with HOPLoP HOPLoP inherently tackles relation prediction, whereas PRA tackles entity prediction. HOPLoP does not require pre-defined paths nor transition probabilities between entities since traversals are performed in the continuous space. During one training step, PRA requires all positive target entities, and assumes all other entities to be "negative". On the other hand, HOPLoP requires one positive and one negative target entity, which may be hand-picked or automatically generated. PRA's reasoning process is explainable. Although we cannot explain the reasoning behind HOPLoP's LP score, we can interpret HOPLoP(TransE)'s reasoning paths.

3.1.3 Path-RNN and Single-Model

Frameworks that followed PRA used continuous representations of the KG, which helped the model understand the global structure of the graph [86]. Neelakantan *et al.* [32] introduced Path-RNN to tackle relation prediction by reasoning about composition of binary relations connected in a path [8]. Similar to PRA, one Path-RNN model was used to model one relation. For a given task, Path-RNN uses PRA, in a pre-processing step, to generate relational paths relevant to the task. To process a traversed path, vector representations of relations were composed using a RNN, $RNN_r(p)$. After "traversing" a path [9], the RNN_r generates a vector representation

[8] In their paper [32], they refer to the compositing two relations as *conjunction*.

[9] The traversal is done by RNN_r, which takes in a new relation at each step and "moves" to a next state.

for that path, which is compared against the embedded representation for the underlying relation of interest. The dot product of the relation embedding and the path embedding, generated by the RNN_r, is squashed by a sigmoid function to output the probability that relation r connects the source and the target entity:

$$\Pr(R = r|\mathcal{P} = p) = \sigma(\vec{r} \cdot RNN_r(p)) \tag{3.5}$$

Das *et al.* [31] introduced a multi-task framework called Single-Model that outperforms Path-RNNs in relation prediction by introducing three modeling changes: include entity type information at each step of the reasoning process; parameter sharing within a *Single-Model* to *reason* about all relations of interest; using a LSTM over RNN.

Differences with HOPLoP HOPLoP does not acquire reasoning paths using PRA; instead it learns to traverse the embedding space. Although both HOPLoP and Single-Model use LSTMs, we do not explicitly provide entity type information. HOPLoP acquires latent information from the structure of the embedding space, which may contain entity type information [87–89]. Although we derive HOPLoP to reason about one relation, we extend our formulation to reason about all relations of interest, using M-HOPLoP.

3.1.4 Compositing KG Embeddings

Composition is the act of creating something new from the combination of two or more objects. In the context of KGs, researchers have fine-tuned pre-existing KG embeddings to encode structural information gained from traversal paths. They do these through several different composition operations [90] or by taking advantage of certain "composable" KG embeddings.

Guu *et al.* [20] tackles the entity prediction problem by introducing a novel compositional training objective that produces a structural regularization effect in the

LP process. Following the previous methods, Guu *et al.* [20] used PRA to generate paths. Given pre-trained composable KG embeddings and paths that connect source entities to target entities, their methodology involves fine-tuning embeddings such that the composition of relations will represent the underlying relationship. This is viewed as a form of structural regularization, since information about the graph structure, captured by path traversals, is embedded into the representations for components of the KG. For example, given a path $\texttt{DirectsGenreOfMovies} \implies \texttt{DirectorOf} \rightarrow \texttt{Genre}$, Guu *et al.* [20]'s approach looks to fine-tune TransE such that $\vec{e}_{\texttt{JamesCameron}} + \vec{r}_{\texttt{DirectorOf}} + \vec{r}_{\texttt{Genre}} \approx \vec{e}_{\texttt{ScienceFiction}}$. Similarly, their methodology is followed to fine-tune RESCAL [25] KG embeddings: $M_{\texttt{Genre}} \times M_{\texttt{DirectorOf}} \times \vec{e}_{\texttt{JamesCameron}} \approx \vec{e}_{\texttt{ScienceFiction}}$ where M_r is a function, in matrix form, that models the relation r. They also fine-tune DistMult [80] KG embeddings and modify the loss function in a similar manner: $Softplus(-y \cdot (\vec{e}_{\texttt{JamesCameron}} \times \vec{r}_{\texttt{DirectorOf}} \times \vec{r}_{\texttt{Genre}} \times \vec{e}_{\texttt{ScienceFiction}}))$ [10].

PTransE developed by Lin *et al.* [90], was a parallel study on the compositionality of KG embeddings. They investigate the compositionality of TransE, and how to incorporate path information. They introduce a novel scoring function that called Path-Constraint Resource Allocation (PCRA) [11], to account for the probability of reaching the target entity from the source entity via any of the pre-defined paths. They experiment with the composition in the scoring function, by replacing addition with multiplication and RNN, and found that addition works best on the TransE space.

Differences with HOPLoP These methods explicitly require paths to be fed into their model, whereas HOPLoP does away with this requirement and searches for paths on its own. Their approach looks to re-train KG embeddings to provide a form of

[10] Notice that this loss function is very similar to ComplEx. Similar to how DistMult performs multiplication of real vectors, ComplEx perform multiplication of complex vectors, but keeps only the real part.

[11] This algorithm is similar to PageRank.

structural regularization or train from scratch, whereas HOPLoP utilizes separate neural architectures that learns to traverse over a pre-existing embedding space. These approaches are only compatible with composable KG embeddings such as RESCAL [25], DistMult [80] and TransE [21], whereas HOPLoP can work with any KG embedding method.

3.1.5 Highlights

Approaches that trained models in a supervised fashion over paths collected using PRA were affected by large fan-out areas caused by certain entities known as supernodes [91], making the LP process inefficient [48]. In contrast, the HOPLoP framework does not utilize PRA to collect relational paths. Instead, it learns to traverse an embedded representation of the KG on it's own. It does so by modifying the vector representation for the source entity such that, at the end of the traversal process, the resultant embedding is equal to the target entity embedding, thereby "reaching" the target entity. By treating every point in space equally, HOPLoP's traversal process isn't affected by supernodes and their skewed node degrees.

3.2 Reinforcement Learning with DeepPath and its successors

To overcome the bottleneck caused by supernodes, successor approaches sought out techniques from Reinforcement Learning (RL) [92]. Xiong *et al.* [48] introduced DeepPath, which is the first algorithm that applies RL for KBC. DeepPath tackles relation prediction by learning to traverse the graph such that, the path taken by the RL agent expresses the underlying relation. Similar to PRA and Path-RNN, one DeepPath model is trained to model one relation. The training phase is broken down into two parts. The first involves training the mathematical model of the RL algorithm using Supervised Learning (SL) and paths collected by a Breath First Search (BFS) procedure. During the second phase of training, DeepPath uses REINFORCE [93], a

RL algorithm, to efficiently search for paths, based on a reward function that takes into account the accuracy, diversity and efficiency of a traversed path. The KG environment operates on a discrete KG and provides the agent with rewards, which the RL algorithm aims to increase. The RL agent, parameterized as a 2-layered ReLU-activated neural network, takes as input the embedded representation of its current position e_c (current entity) and direction to move towards, expressed. by the vector $(\vec{e_t} - \vec{e_c})$. At each time-step, the RL agent outputs the relation to traverse, which is used by the environment to discretely search for the "next-hop" entity. Once the DeepPath model has been trained for a particular relation, it traverses the KG from the source entity to the target entity. The path used to traverse the graph is the compared against the set of paths DeepPath finds during the training phase.

Das *et al.* [49] introduced MINERVA, which uses RL to tackle the entity prediction task. Their model utilizes a policy network that finds a path that connects the source entity with a target entity such that the underlying relation of interest is expressed. Rather than choosing only a relation at each time-step, MINERVA can choose a specific link, involving a relation and next-hop entity. Rather than pretraining the RL model using SL, MINERVA is an "only-RL" solution that uses a LSTM network to generate a path embedding at each step. This path embedding is sent to a NN model to predict the link to traverse at that time-step. MINERVA's environment uses the discrete KG to search and extend the path towards the target entity. Since MINERVA assigns a constant reward of +1 to all paths that reach the required target entity, it ignores the quality of the path [50, 84].

To improve the quality of traversed paths, Lin *et al.* [50] introduced a reward shaping strategy that gives a reward of +1 if the agent reaches the correct target entity; but if the agent did not reach the correct target entity, the reward was a function of a pre-trained KG embedding model. This ensures that the RL agent is *not* spuriously penalized for predicting links that are not present in the KG. The underlying assumption is that the KG embedding method is more complete than the

discrete KG and contains information about links that are not present in the discrete KG. Shen *et al.* [94] introduced a general graph traversal algorithm called M-Walk and applies it in the context of KBC. M-Walk employs a RNN network and Monte Carlo Tree Search (MCTS) guiding the search to pick entities that help the agent move towards the target entity.

3.3 Variational Inference for multi-hop LP

Ranganathan *et al.* [78] describe two phases in the multi-hop link prediction process: Path-finding and Path-reasoning. Path-finding is the process of searching for a sequence of entities and relations that connect two entities. Path-reasoning is the process of evaluating whether a traversal path represents an underlying relationship. Approaches prior to Chen *et al.* [84] did not facilitate adequate interactions between the two phases of multi-hop LP.

For example, DeepPath and MINERVA can be interpreted as enhancing the Path-finding step while compositional reasoning [31, 32] algorithms can be interpreted as enhancing the Path-reasoning step. DeepPath is trained to find paths more efficiently between two given entities while being agnostic to whether the link exists between the two entities. MINERVA learns to reach target nodes given a source entity-relation pair while being agnostic to the quality of the searched path. Compositional reasoning models learn to predict the underlying relation given paths, while being agnostic to the path-finding procedure.

The lack of interaction prevents the model from understanding more diverse inputs and make the model very sensitive to noise and adversarial samples. To this end, Chen *et al.* [84] introduced DIVA to tackle the relation prediction problem using neural architectures coupled with the variational auto-encoder algorithm [95] for training and inference to cope with complex link connections in a KG.

Differences with HOPLoP While HOPLoP and DIVA aim to improve the interaction between the two subtasks of multi-hop LP process, HOPLoP operates entirely over an embedding space. By operating over an embedding space, HOPLoP is not affected by the incompleteness problem since it operates over a more-complete representation of the KG and has the ability to create traversal functions that can account for errors in the embedding space.

3.4 Representation Learning for Multi-hop LP with HOPLoP

Previous approaches traverse a discrete representation of the KG, abstracted by either the environment [48, 49, 94] or a discrete latent variable [84]. The traversal operation in previous multi-hop approaches is restricted by the links in the KG. These links are skewed, causing LP algorithms to spuriously favour supernodes over actual target entities [91], eventually inhibiting performance gains in the LP task. HOPLoP is not constrained to follow links in the KG, allowing it to traverse anywhere in an embedded space, while boosting performance of LP. During the training phase, HOPLoP learns to traverse the embedded space while classifying positive and generated negative pairs of entities. Operating over an embedding space enables end-to-end differentiability. This allows HOPLoP to be optimized efficiently using GD methods [29], unlike RL approaches or variational inference based optimization methods, which are computationally expensive [96].

Link Prediction Algorithm	ML Type	Additional Requirements	#Relations Modelled	Trainable Parameters	
				KG Components	Learning Function
TransE [21]	SSL	Negative Samples	$\|\mathcal{R}\|$	$d(\|\mathcal{E}\| + \|\mathcal{R}\|)$	0
ComplEx [74]	SSL	Negative Samples	$\|\mathcal{R}\|$	$d(\|\mathcal{E}\| + \|\mathcal{R}\|)$	0
TuckER [75]	SL	—	$\|\mathcal{R}\|$	$(d_e\|\mathcal{E}\|) + (d_r\|\mathcal{R}\|)$	$d_e \times d_r \times d_e$
PRA [23]	SL	Paths	1	$\|\mathcal{P}\|$	0
Path-RNN [32]	SL	Paths	1	$d\|\mathcal{R}\|$	$d(2d+1)$
Single-Model [31]	SL	Entity Types, Paths	$\|\mathcal{K}\|$	$d(\|\mathcal{E}\| + \|\mathcal{R}\|)$	$4d(3d+1)$
Guu *et al.* [20]	SL	Paths, Negative Samples	$\|\mathcal{K}\|$	$(d_e\|\mathcal{E}\|) + (d_r^{(1-2)}\|\mathcal{R}\|)$	0
DeepPath [48]	RL	Entity Embeddings, Paths	1	0	$512(2d+1) + 525312 + 1025(\|\mathcal{R}\| - \|\mathcal{K}\|)$
MINERVA [49]	RL	—	$\|\mathcal{K}\|$	$d(\|\mathcal{E}\| + \|\mathcal{R}\|)$	$4d(2d+1) + 4d(2d+1) + 2d(4d+1)$
M-Walk [94]	RL	—	1	$d(\|\mathcal{E}\|)$	$64(d_e + d_r + 1) + 5(65 \times 64) + 3d(128 + d_e)$
Reward Shaping [50]	RL	KG Embedding Model	$\|\mathcal{R}\|$	$d(\|\mathcal{E} + \mathcal{R}\|)$	$4d(2d+1) + 2d(3d+1) + 2d(2d+1) + 2d(2d+1)$
DIVA [84]	PGM	Negative Samples	$\|\mathcal{K}\|$	$d(\|\mathcal{E} + \mathcal{R}\|)$	$12d + \|\mathcal{R}\|(3d+1) + 4d(2d+1) + 2d(2d+1) + 2d(3d+1)$
HOPLoP	SSL	Entity Embeddings, Negative Samples	1	0	$1000(2d+1) + 1001d + 4d(2d+1) + (d+1)$
M-HOPLoP	SSL	Entity Embeddings, Negative Samples	$\|\mathcal{K}\|$	0	$1000(2d+1) + 1001d + 4d(2d+1) + \|\mathcal{K}\|(d+1)$

Table 3.1: Comparison of various KG embedding and multi-hop algorithms for LP. d_e refers to the entity embedding dimension, d_r refers to the relation embedding dimension, $d = d_e = d_r$ is the dimension of all components, $\mathcal{K} \subseteq \mathcal{R}$ refers to the set of tasks in the dataset. Since DeepPath, HOPLoP, M-HOPLoP do not train the entity embeddings, they are considered as constant and are not counted in the set of trainable parameters. Note: The formulae describing parameter sizes for each model are best approximates of their publicly available model descriptions and reproducible material.

Chapter 4

HOPLoP: Multi-hop Link Prediction over Knowledge Graph Embeddings

4.1 Motivation

- **Traversing an embedded space?** Our hypothesis is: performance in multi-hop LP can be improved if the KG is more complete than its discrete representation. —What does this mean? (r.i) There should be more links in the KG representation, allowing the path-finder to better explore the graph. (r.ii) There should be more entities, allowing the path-finder to "hop and stop" at various entities. KG entity embedding spaces fit these requirements: (i) In "space" (say, $\mathbb{R}^D$), an "one-hop" *unconstrained* path-finder, should be allowed to traverse from any point to any another point in space. This is possible through simple vector addition [21]. (ii) An *unconstrained* path-finder should be allowed to pause or stop anywhere in space, to model those entities that may have been pretermitted during KBP [78]. HOPLoP's path-finder is a one-hop function that learns to output a sequence of *translation vectors*, moving itself from the source entity to the target entity. We allow this one-hop function to perform multiple traversals to anywhere in the space, but we *control* its traversal towards the target entity.

- **Now, because we are traversing an embedded space**, we have ∞ paths from the source to the target entity. This allows the path-finder to be more expressive. DL architectures such as LSTMs can form appropriate compressed representations for all these paths, which then can be used by a simple logistic regression function that models the LP task. A "distance-to-target" loss L_{D2T} controls the path-finder to find a traversal path, or a sequence of traversal vectors, that link the source to the target entity, irrespective of whether a link exists or not in the discrete KG.

- **Since all operations (traversal as a sequence of translations and neural architectures) are connected and differentiable**, we can use simple GD-based BP to tune the parameters of the model. This will be preferred from using computationally expensive techniques such as, RL (which usually requires pre-training with SL), and variational inference techniques [96]. Through back-propagation, a logistic regression model, which predicts the existence of the link, can propagate error gradients to the path-reasoner, which in turn can propagate errors to the path-finder, facilitating adequate interactions between the path-finder and path-reasoner [84].

- **KG entity embedding space contains neural representations of entities.** A neural path-finder [48] can utilize this information to model the relationship between the 2 entities using *paths* in the space that connect the source to the target entity. The LSTM network can efficiently compress long range sequences such as paths in a KG [31, 49, 50, 84]. Since we do not alter the existing embedding space and use seperate neural architectures to traversal any embedded space, HOPLoP does not restrict it to any specific KG embedding method and can be used in parallel with existing embedding use-cases.

4.2 Task

We formally describe our task of link prediction over a KG. To recap, a Knowledge Graph (KG) is a collection of facts, represented as the links of a graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{L})$ where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relations that label the links, $\mathcal{L} \subseteq \mathcal{E} \times \mathcal{E} \times \mathcal{R}$ is the set of links, each *linking* a source entity to a target entity via a relation. An example of a link in the YAGO3 KG [97] is represented as
`(James_Cameron, directed, Avatar_(2009_film))`.

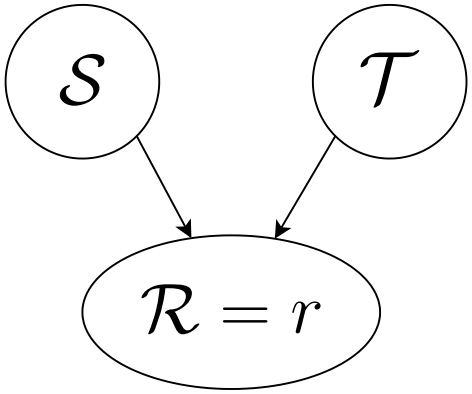


Figure 4.1: Our LP task, represented as a PGM.

Our task is to fill in the incompleteness of the KG by predict missing links in the knowledge graph $\mathcal{G}$. Specifically, we tackle the problem of relation prediction $(e_s, ?, e_t)$, i.e., find the relation between the given source and target entity. To promote differentiability, we convert this problem into the stochastic setting by modeling the probability that a relation of interest or task $k \in \mathcal{K} \subseteq \mathcal{R} = r$ [1] exists between two entities e_s and e_t. To derive HOPLoP, we begin by representing our LP task with a Probabilistic Graphical Model (PGM). Figure 4.1 represents our LP task, which formally represents equation 4.1.

$$\Pr(\mathcal{R} = r | \mathcal{S}, \mathcal{T}) \tag{4.1}$$

where $\mathcal{S}$ and $\mathcal{T}$ are random variables representing all source and target entities

[1] This means that HOPLoP can be applied to any relation in the KG, not only relations of interest. To avoid confusion, when we refer to relation r, we refer to our task.

respectively.

HOPLoP learns to predict links by traversing an embedded representation of a KG from a source entity to a target entity. This will result in a sequence of "steps", forming a path p. Let $\mathcal{P}$ be a random variable, indicative of all paths in the embedded KG space. On incorporating path information $\mathcal{P}$ in the LP process, equation 4.1 becomes:

$$\Pr(\mathcal{R} = r|\mathcal{S}, \mathcal{T}) = \Pr(\mathcal{R} = r|\mathcal{S}, \mathcal{T}, \mathcal{P}) \times \Pr(\mathcal{P}|\mathcal{S}, \mathcal{T}) \tag{4.2}$$

Equation 4.2 is represented by figure 4.2.

Since HOPLoP operates on an embedded space, we represent each hop as a traversal vector. A traversal path $\mathcal{P} = p$ is a sequence of traversal vectors $p = (\vec{v_1}, ..., \vec{v_H})$ where $H = |p|$ is the length of the sequence. Each traversal vector characterizes a "hop" from one point in the embedding space to another point in the same embedding space. In the next section, we describe how we attain a path p. We continue to simplify and formulate our LP task.

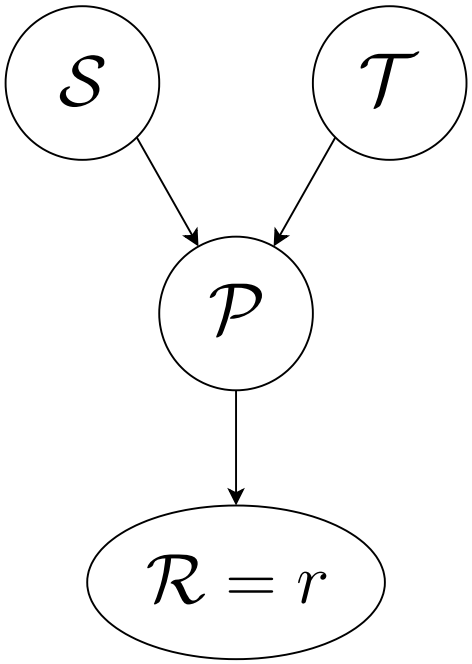


Figure 4.2: Upon incorporating "paths" in the LP process.

Our goal is to combine the generalization ability of KG embeddings with the predictive power of global structural information. We allow the path-finder to explore "paths", unconstrained, over an embedded space. By doing so, there will be trade-off between the completeness of a KG and the accuracy of the KG representation. Therefore, HOPLoP aims to accurately complete an imperfect representation of the KG. To do this, we make the assumption that any path $p \in \mathcal{P}$ exists in the KG embedding space. This assumption is represented by equation 4.3.

$$\Pr(\mathcal{P} = p|\mathcal{S}, \mathcal{T}) = 1 \tag{4.3}$$

Equation 4.3 formalizes the assumption that there exists a path p given any source entity $\mathcal{S}$ and any target entity $\mathcal{T}$. This assumption holds since HOPLoP's path finder traverses the KG in a continuous space, and there always exists a line $(\vec{e_t} - \vec{e_s})$ that connects the source entity to the target entity in embedded space. As we will see, this assumption will allow us to separate the path-finding and the path-reasoning

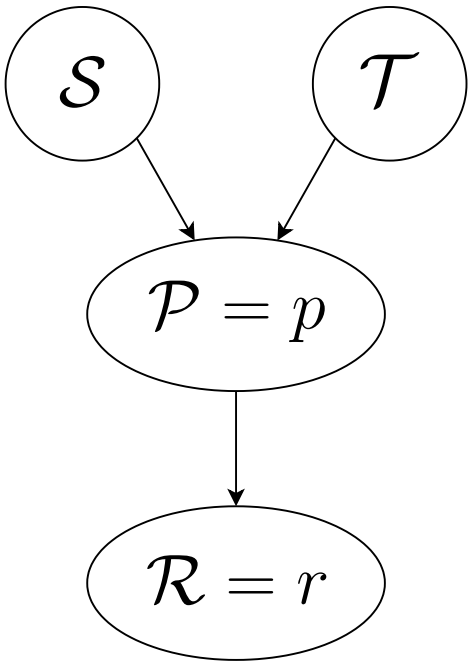


Figure 4.3: An intermediate PGM illustration of our LP process. Once a path p is traversed, the parent random variables $\mathcal{S}$ and $\mathcal{T}$ does not directly influence the child random variable $\mathcal{R}$.

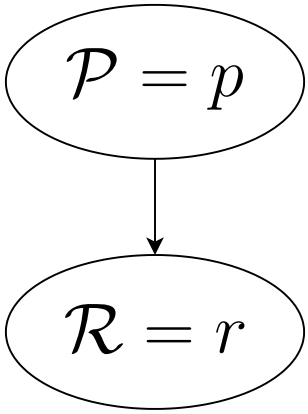


Figure 4.4: Our LP task is simplified to predict the probability that a generated path p represents the relation $\mathcal{R} = r$.

processes, while facilitating adequate interactions between the two processes. This assumption is reflected in equation 4.4.

$$\Pr(\mathcal{R} = r|\mathcal{S}, \mathcal{T}) = \Pr(\mathcal{R} = r|\mathcal{S}, \mathcal{T}, \mathcal{P} = p) \tag{4.4}$$

Since the path p is generated based on the source and the target entities, the random variable $\mathcal{P}$ is a *child* of random variables $\mathcal{S}$ and $\mathcal{T}$. After traversing the KG, we know that $\mathcal{P} = p$. Since the parent of the random variable $\mathcal{R}$ is known, the non-descendents of $\mathcal{R}$ do not influence $\mathcal{R}$ [98]. Therefore, equation 4.4 reduces to equation 4.5, illustrated by figure 4.4.

$$\Pr(\mathcal{R} = r|\mathcal{S}, \mathcal{T}, \mathcal{P} = p) = \Pr(\mathcal{R} = r|\mathcal{P} = p) \tag{4.5}$$

4.3 Model

In the previous section, we have simplified and divided our task into 2 parts: generating a path or "path-finding" and reasoning about a path to predict a link or "path-reasoning". To solve this problem of LP, we employ two neural network architectures [62]. The path-finder is a simple feedforward neural network, consisting of a hidden layer with 1000 ReLU-activated neurons, that learns to traverse the con-

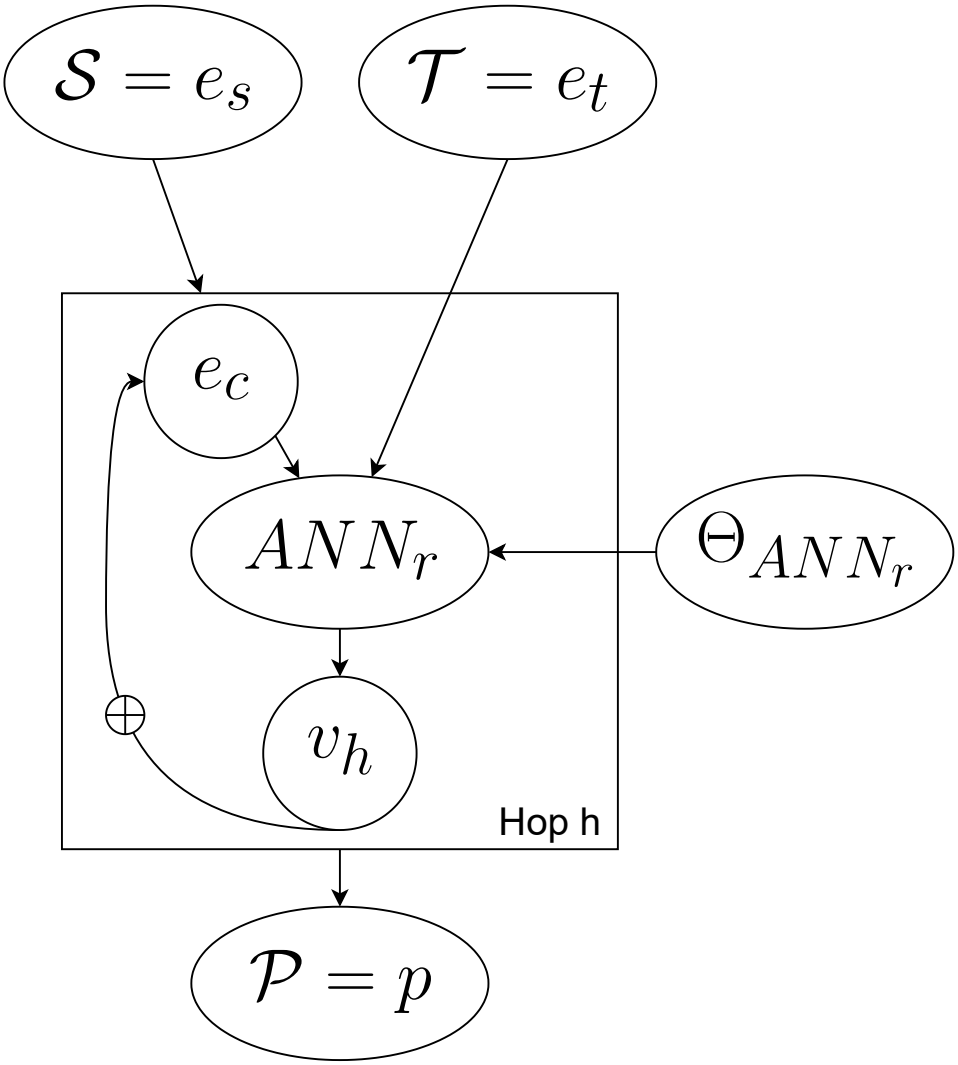


Figure 4.5: PGM representing the Path-finding process. The rectangle in the figure represents a plate model, which is repeated for H hops. e_c represents the current position in the entity embedding space, v_h represents the traversal vector at hop h, $\oplus$ represents pointwise addition. Finding a path p from e_s to e_t is performed by an ANN, whose parameters are controlled by Θ_{ANN_r}.

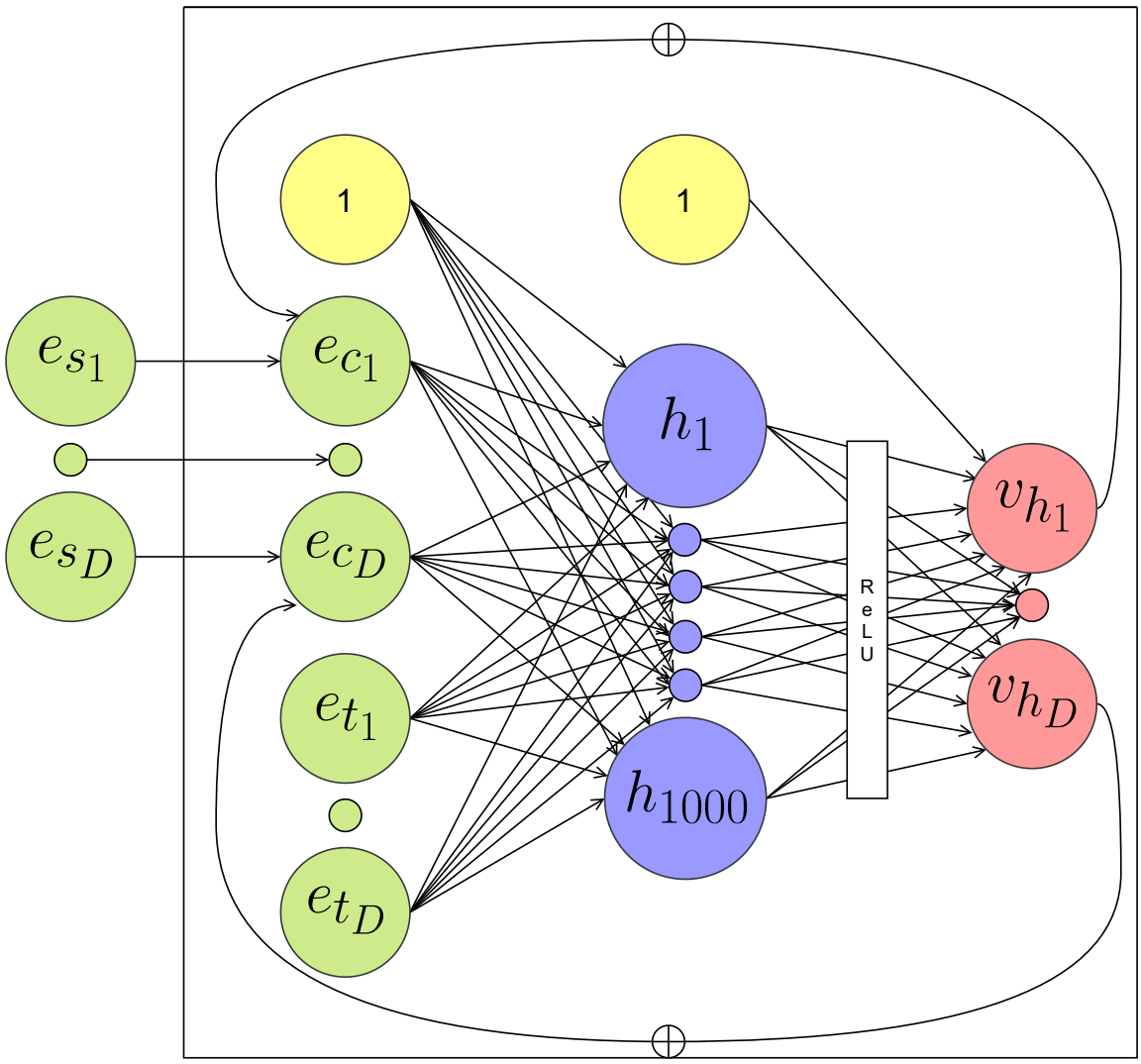


Figure 4.6: (Best viewed in color) The connections in the path-finder model. An ANN that accepts the concatenated input of the current and target entity $[e_c; e_t] \in \mathbb{R}^{2D}$ and outputs a translation vector $v_h \in \mathbb{R}^D$ which is added to the current entity representation. The ANN consists of 2 layers: a hidden layer with 1000-ReLU activated neurons and an output layer with 100 linear neurons.

tinuous representation of the KG. The output layer contains neurons corresponding to the dimension of the KG embedding space. This neural network accepts, as input, concatenated vector representations of the source and target entity. It outputs a translation vector, which is added to the source entity embedding. This resultant vector is concatenated with the target entity embedding and fed into the neural network, to perform another hop over the embedded KG. This process continues until a maximum number of hops H is reached.

Our framework also includes a path-reasoner, which is an LSTM [71] network that analyses the sequence of translation vectors. The LSTM network is single-layered, consisting of LSTM cells corresponding to the entity embedding dimension. The hidden state vector of the LSTM at the end of the traversal can be interpreted as the path embedding. This path embedding is sent to a sigmoid-activated neuron that performs logistic regression. The LSTM network takes, as input, the sequence of translation vectors, outputted by the path-finder. The output aims to predict the probability that the relation $\mathcal{R} = r$ holds between the source entity and the target entity.

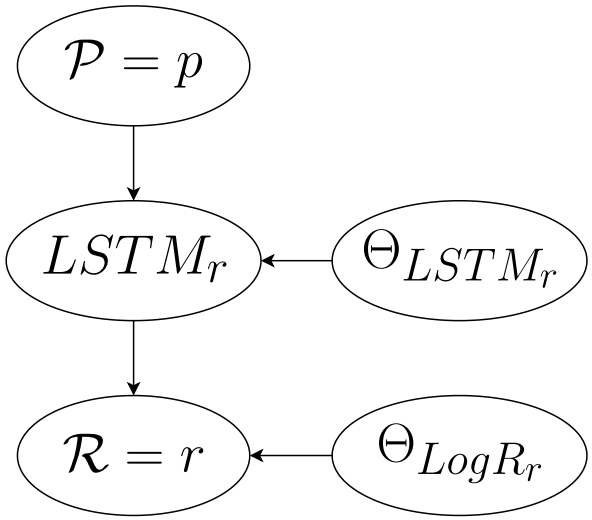


Figure 4.7: PGM diagram illustrating the path-reasoning process. The path is analyzed by the LSTM network, whose parameters are controlled by Θ_{LSTM_r}. The path representation from the LSTM is used to perform logistic regression, parameterized by Θ_{LogR_r}, to predict whether the relation of interest exists between the two entities.

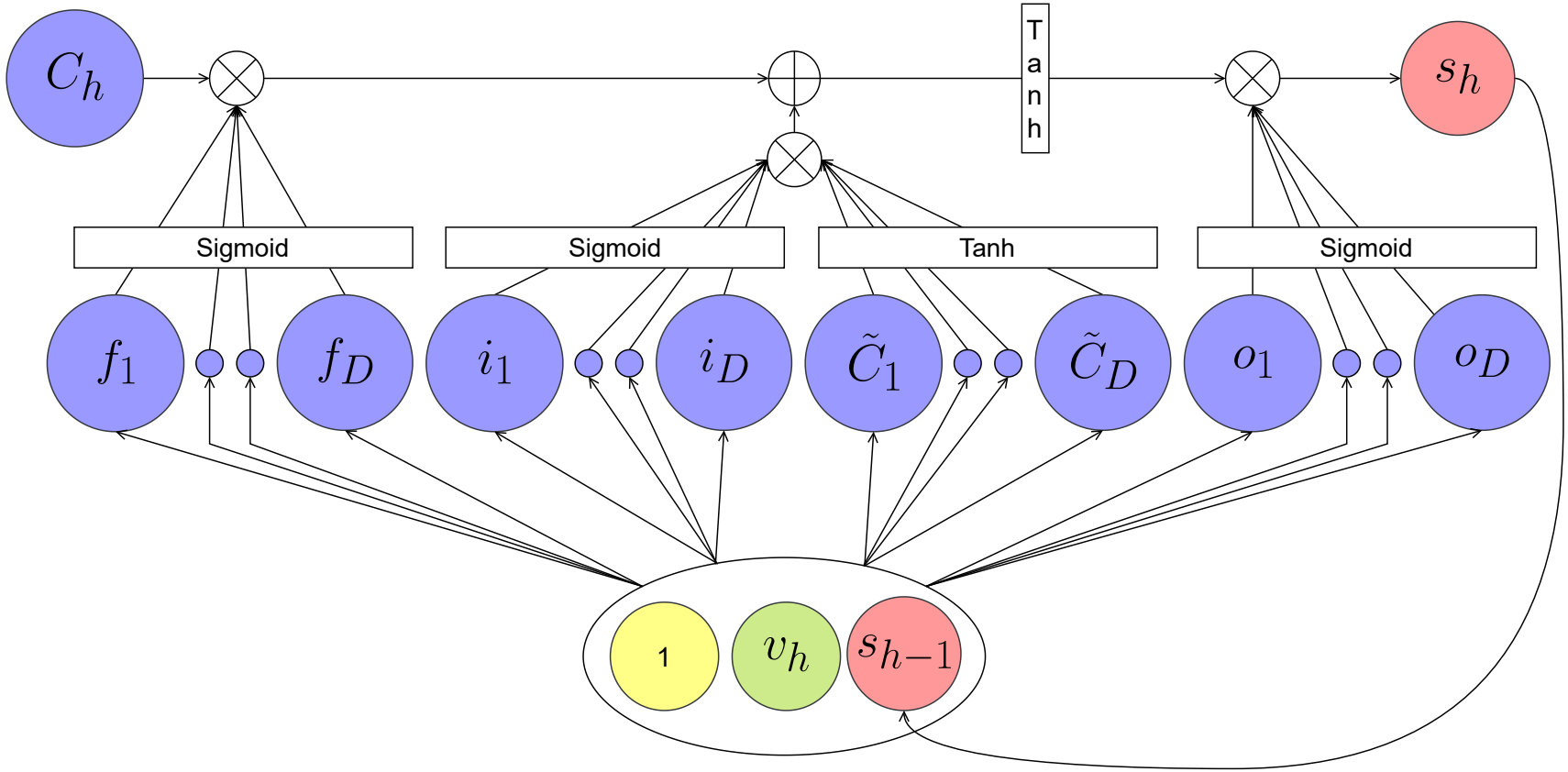


Figure 4.8: (Best viewed in color) Illustration of the connections in 1 hop of the LSTM network. To avoid confusion with hop h, s_h denotes the hidden representation of the LSTM at hop h and C_h denotes the cell state at hop h. At each hop, the LSTM refers input as v_h and changes it's hidden state from $s_{(h-1)}$ to s_h. See section 2.5.2 for more on LSTMs.

To summarize, our path-finder traverses the embedding representation of the KG to form a path $\mathcal{P} = p$ as given by:

$$p = (\vec{v_1}, ..., \vec{v_H}) \tag{4.6}$$

where $\vec{v_i}$ is defined as:

$$\vec{v_i} = ANN_r \left([e_s + \sum_{h=1}^{i-1} \vec{v_h}; e_t] \right) \tag{4.7}$$

Finally, our link prediction problem is formulated by:

$$\Pr(\mathcal{R} = r | \mathcal{P} = p) = \sigma(u_r^T \times LSTM_r(p) + b_r) \tag{4.8}$$

where u is a trainable vector that is multiplied with the last hidden state of the LSTM, and b represents a trainable bias. The result is squashed using the sigmoid function σ to obtain a value between 0 and 1 that represents the probability that a link with relation r connects the source entity e_s to the target entity e_t. Subscript r indicates relation-specific functions and parameters.

4.4 Training

The path-finder consists of a feedforward neural network that aims to traverse the KG, from the source entity to the target entity. To learn this traversal function, we control the path-finder by employing a distance-to-target (D2T) loss. At the end of the traversal, we expect that the path-finder should reach the target entity position. This loss can be given as follows.

$$L_{D2T} = \sum_{d=1}^{D} \left(\vec{e_{t_d}} - \left(\vec{e_{s_d}} + \sum_{h=1}^{H} \vec{v_{h_d}} \right) \right)^2 \tag{4.9}$$

where $\vec{e_t}$ is an embedding representing the target entity, $\vec{e_s}$ is an embedding representing the source entity, $\vec{v_i}$ is the traversal vector, outputted by the neural network,

at the i^{th} hop, d represents the element at that position in the vector and D is the embedding dimension.

We formulate the LP task as a classification problem. The model should learn to classify between positive and negative pairs of entities for a relation r. A positive entity pair involves two entities which are related via the relation r and the link (e_s, r, e_t) is observed in the KG. To perform this classification task, we use the binary cross-entropy loss function. The classification loss can be given as follows.

$$L_C = -y \times \log(o) + (1 - y) \times \log(1 - o) \tag{4.10}$$

where $o = \Pr(\mathcal{R} = r|\mathcal{P} = p)$ is the value outputted by the LSTM network, and y is a boolean variable that signifies if the relation r holds between the source entity and the target entity. Since our model learns to classify between positive and negative entity pairs, following [48], we train the model with positive and negative examples in a 1:1 ratio.

The two objectives, described in equation 4.9 and 4.10, are jointly optimized using GD [29]. The overall objective function can be given as follows.

$$\min_{\Theta_r} L = L_{D2T} + L_C \tag{4.11}$$

where $\Theta_r = \Theta_{ANN_r} \cup \Theta_{LSTM_r} \cup \Theta_{LogR_r}$ represents the parameters of the model. The differentiability of our framework allows us to jointly optimize the parameters of the path-finder and the path-reasoner.

Algorithm 3 *Training HOPLoP*

Input: *Dataset* $D_r = \{.., (e_s, [.., e^+_{t_i}, e^-_{t_i}, ..]), ..\}$, *Function* $\mathsf{HOPLoP}_r(ANN_r, LSTM_r, LogR_r)$, *Hops* H, *Loss functions* L_C *and* L_{D2T}, *Initial Parameters* $\Theta_r^{(t=0)}$

$\Theta_r^{(t)} \leftarrow \Theta_r^{(t=0)}$;

Repeat:

$(e_s, e^+_t, e^-_t) \sim D_r$; ▷ *Sample a source entity, positive and negative target entity*

$p^+ \leftarrow []$; $p^- \leftarrow []$; ▷ *Path is initially empty*

$e^+_c \leftarrow e_s$; $e^-_c \leftarrow e_s$; ▷ *Initial position is source entity*

for h *in* $1...H$ ***do***

Append $v^+_h \leftarrow ANN_r(e^+_c, e^+_t)$ *to* p^+ *and* $v^-_h \leftarrow ANN_r(e^-_c, e^-_t)$ *to* p^-;

$e^+_c \leftarrow e^+_c + v^+_h$; $e^-_c \leftarrow e^-_c + v^-_h$;

end for

$o^+ = LogR_r(LSTM_r(p^+))$; $o^- = LogR_r(LSTM_r(p^-))$;

$\nabla_{\Theta_r} L_C \leftarrow \frac{\partial(L_C(o^+,1)+L_C(o^-,0))}{\partial\Theta_r}(\Theta_r^{(t)})$; ▷ *Calculate* L_C *gradients*

$\nabla_{\Theta_r} L_{D2T} \leftarrow \frac{\partial(L_{D2T}(e^+_c,e^+_t)+L_{D2T}(e^-_c,e^-_t))}{\partial\Theta_r}(\Theta_r^{(t)})$; ▷ *Calculate* L_{D2T} *gradients*

/* $\delta_{LogR_r} \propto \nabla_{\Theta_r} L_C$;

$\delta_{LSTM_r} \propto \delta_{LogR_r}$;

$\delta_{ANN_r} \propto \nabla_{\Theta_r} L_{D2T} \times \delta_{LSTM_r}$; */ ▷ *Gradient proportions for parameters*

$\Theta_r^{(t+1)} \leftarrow \Theta_r^{(t)} - \mu \times \nabla_{\Theta_r}(L_C + L_{D2T})$; ▷ *Update parameters*

Until *convergence;* ▷ *Time-step increments by 1*

Return $\Theta_r^{(t)}$;

4.5 Discussion

HOPLoP framework essentially performs a logistic regression [99] which learns a decision boundary that can accurately distinguish positive and negative entity pairs of entities for a particular relation of interest. This classification is based on a vector representation of the traversed path, generated by the LSTM that analyses the sequence of translation vectors used to traverse from source to the target entity. The traversal over an embedding space involves modifying the source entity embedding several times, through vector addition, such that, at the end of the traversal, the modified source entity should represent the target entity and the sequence of modifications applied on the source entity by the ANN should allow the LSTM to form effective path embeddings that characterizes the traversal process. Through backpropagation [63], the LSTM network controls how the neural network learns to traverse the embedding space, thus providing adequate interaction between the path-finding and path-reasoning processes.

4.6 M-HOPLoP: Modeling all relations at once

The inherent differentiability of HOPLoP can pique the interest of an experienced ML practioner towards multi-task learning and parameter sharing mechanisms [30]. Similar to the advancement made by the Single-Model [31] over Path-RNNs [32], we explore multi-task learning capabilities of HOPLoP for KBC.

To this end, we make a simple yet highly effective change to HOPLoP. Given a dataset and the set of relations of interest or tasks $\mathcal{K} \subseteq \mathcal{R}$, rather than performing simple logistic regression at the end, we changed HOPLoP's classification heads from 1 to $|\mathcal{K}|$, to perform multiple logistic regression [99]. This results in a change to the logistic regression layer, which is now parameterized by a matrix in $\mathbb{R}^{(D+1,|\mathcal{K}|)}$. In HOPLoP, this layer is parameterized by a vector in $\mathbb{R}^{(D+1)}$. For example, since we are interested in 12 relations from the NELL-995 KG dataset, the number of classification

heads for M-HOPLoP will be 12. We don't change the parameter size for the path-finder and the LSTM layer, and continue to train M-HOPLoP to reduce the loss as defined in equation 4.11 for HOPLoP.

By modeling all relations of interest using one M-HOPLoP model, information regarding all relations are distributed amongst the parameters of path-finder and the path-reasoner. Similar to HOPLoP, M-HOPLoP traverses the KG embedding space from the source entity to the target entity. Different from HOPLoP, M-HOPLoP is trained to recognize which relation links the two entities. This allows M-HOPLoP to model different relations, while predicting whether a link involves a particular relation or not.

Chapter 5

Experiments and Results

5.1 Datasets

To test our framework on the task of (multi-hop) link prediction, we evaluate on standard datasets such as NELL-995 [48] and FB15K-237 [100] and their statistics have been provided in Table 5.1. Every dataset contains a set of tasks, relations of interest for which links must be predicted. These tasks were extracted from different domains like `Sports`, `People`, `Locations`, `Film`, etc. For each task, there is a set of source entities. Each source entity e_s is related to a set of positive target entities {..., $e^+_{t_i}$, ...} via a specific task r. The datasets also include several negative target entities {..., $e^-_{t_i}$, ...} that are not linked to the source entity via the relation of interest. For example, given a relation task `DirectsGenreOfMovies`, there may be a source entity `JamesCameron` for which there is a positive target entity `ScienceFiction` and many

Dataset	#Entities	#Relations	#Triples	Node Degree
NELL-995	75,492	400	308,426	(1, 1.0, 4.09, 2411)
FB15K-237	14,505	474	620,158	(1, 26.0, 42.75, 8642)
WN18RR	40,714	22	173,670	(1, 3.0, 4.27, 482)
YAGO3-10	123,143	74	2,158,080	(1, 10.0, 17.52, 61044)

Table 5.1: Statistics of KG datasets used in experiments. The 4-tuple in the node degree column represents aggregate statistics (min, median, mean, max) over all entities in the graph.

Dataset	#Tasks	Source Entities	Target Entities	
			Positive	Negative
NELL-995	12	(259, 599.5, 685.92, 1405)	(1.0, 1.0, 1.15, 1.83)	(2.67, 7.90, 7.08, 9.46)
FB15K-237	20	(67, 285.5, 630.9, 3245)	(1.0, 1.13, 1.56, 2.94)	(7.69, 10.06, 10.02, 10.72)
WN18RR	10	(25, 2222.0, 6255.6, 34087)	(1.02, 1.84, 4.62, 25.16)	(5.44, 28.71, 44.80, 184.16)
YAGO3-10	23	(357, 2850.0, 13019.91, 66163)	(1.0, 2.61, 4.35, 21.41)	(1.0, 26.15, 42.96, 211.88)

Table 5.2: Statistics of tasks in KG datasets used in experiments. 4-tuples in the source entities column are aggregate statistics (min, median, mean, max) over all tasks. The 4-tuples in the target entities column are aggregate statistics over all source entities.

negative target entities such as {`Horror, Drama, Comedy, ...` }. Following Guu *et al.* [20] and Xiong *et al.* [48], negative target entities are picked from the same domain as the positive target entities, allowing for a fair evaluation of our framework. Following our previous example, the set of negative target entities will not contain entities such as `Canada`, `Titanic`, `Football`, etc., since these entities are out of the range of the relation `DirectsGenreOfMovies`, whose range is equivalent to the range of `Genre`. Futhermore, following Lin *et al.* [90], we include the reverse relations for each relation in the KGs. Specifically, for each link (e_s, r, e_t), we add another link (e_t, r^{-1}, e_s) to the KG. This technique has known to improve the performance of KG embeddings in the LP task [101], allowing us to compare HOPLoP and M-HOPLoP to stronger baseline KG embedding models. This technique helps previous multi-hop algorithms by enabling bi-directional traversal of a link, i.e., a RL agent is able to step backward in the KG [48]. This allows them to correct for mistakes in the graph, due to incompleteness, or mistakes in the traversal process.

To further evaluate our model, we introduce the WN18RR and YAGO3-10 [102] datasets for the task of relation prediction. These datasets have been previously used for entity prediction [34], in the literature of KG embeddings. To maintain consistency with well-known datasets in this literature of multi-hop LP, we pick several tasks from each dataset and generate negative target entities for links involved in these relations:

5.1.1 WN18RR

The WN18RR dataset was derived from the WN18 dataset introduced by Bordes *et al.* [21], which had 18 relations scraped from WordNet for roughly 41,000 synsets, resulting in 141,442 triplets. As first stated by Toutanova *et al.* [100] and confirmed by Dettmers *et al.* [102], the WN18 dataset suffered from informative value, because more than 80% of the test triples (e_s, r_1, e_t) could be found in the training set with another relation (e_s, r_2, e_t) or (e_t, r_2, e_s). Dettmers *et al.* [102] used a rule-based model which learned the inverse relation and achieved SOTA results on the WN18 dataset. Therefore, Dettmers *et al.* [102] introduced WN18RR by removing 7 spurious relations from the WN18 dataset. These relations were either identical to or the inverse of the remaining 11 relations in the KG. For example, the test set frequently consisted of triples such as (e_s, `Hyponym`, e_t) while the training set contained its inverse (e_s, `Hypernym`, e_t).

We take advantage of the hierarchical structure of WordNet to generate negative examples. Given a positive entity pair, we include all hyponyms of the hypernyms of the positive target entity as negative target entities for that source entity. This ensures negative target entities are from the same domain as the positive target entity. We pick 10 tasks from 11 relations from the KG such that the relation is involved in atleast 1% of all facts. This covers 99.9% of the WN18RR KG.

5.1.2 YAGO3-10

YAGO3-10 [97] is a subset of YAGO3 which consists of entities which have a minimum of 10 relations each. In this dataset, most of the triples deal with descriptive attributes of people, such as `Citizenship`, `Gender` and `Profession`.

Tasks	M-HOPLoP (TransE)	HOPLoP (TransE)	M-Walk	MINERVA	DeepPath	PRA	TransE (Bernoulli)
AthletePlaysForTeam	**0.911**	**0.953**	0.847	0.827	0.721	0.547	0.727
AthletePlaysInLeague	0.973	**0.997**	0.978	0.952	0.927	0.841	0.726
AthleteHomeStadium	**0.963**	**0.930**	0.919	0.928	0.846	0.859	0.798
AthletePlaysSport	0.969	0.929	0.983	**0.986**	0.917	0.474	0.805
TeamPlaysSport	**0.940**	**0.980**	0.884	0.875	0.696	0.791	0.759
OrgHeadquateredInCity	**0.963**	**0.956**	0.950	0.945	0.790	0.811	0.912
WorksFor	**0.966**	**0.993**	0.842	0.827	0.699	0.681	0.901
BornLocation	**0.956**	**0.965**	0.812	0.782	0.755	0.668	0.744
PersonLeadsOrg	**0.929**	**0.962**	0.888	0.830	0.790	0.700	0.899
OrgHiredPerson	**0.967**	**0.930**	0.888	0.870	0.738	0.599	0.868
Overall	**0.946**	**0.934**	0.899	0.876	0.788	0.697	0.828

Table 5.3: Performance of (M-)HOPLoP against baseline path-based and embedding-based approaches to the relation prediction task on the NELL-995 dataset. Values represent MAP scores. DIVA attained an overall MAP score of 0.886.

Since the YAGO is not hierarchically structured, we used Breadth First Search (BFS) to obtain negative examples. BFS traversed the links of the graph, while remaining agnostic to the relation, to reach entities that are connected to a source entity. We made sure that the negative target entities, picked by BFS, were in the range of the task. We used BFS rather than random sampling techniques to make the dataset more competitive, following the intuition that baseline KG embedding models will be able to distinguish between positive and negative entity pairs simply based on the distance between the entities [27]. We pick 23 tasks from 37 relations from YAGO such that there are atleast 3,000 facts per relation to ensure adequate representation of the relation in embedded representation of the KG. This covers 99% of the YAGO3-10 KG. Following previous work, we extracted approximately 10 negative target entities per positive target entity. For the relation `hasGender` in the YAGO3-10 dataset, there was only 1 negative target entity per positive target entity. This is because the `hasGender` relation in YAGO has a binary range of entities {`male`, `female`}. The datasets, and codes for building them, codes for experiments, are publicly available at https://github.com/varunranga/HOPLoP. Please refer to Appendix B for more on hyperparameter tuning and reproducibility.

5.2 Experimental Setup

Although our framework is agnostic to the KG embedding model, HOPLoP relies on a base KG embedding model to provide an embedding space to traverse over. We experiment with 3 popular KG embedding models as described in section 2.6.2. In short:

- TransE [21] was the first, yet extremely competitive [76], translational distance LP model that capture relations between entities such that $\vec{e_s}+\vec{r} \approx \vec{e_t}$. ComplEx [74] and TuckER [75] are derived from tensor factorization methods (ComplEx: [25, 80]), TuckER: [82]) which looks to decompose the binary tensor representa-

tion of the KG. The optimization goals between the two methods are very different: translational distance models look to minimize the distance, in euclidean space, between two related entities whereas tensor decomposition methods look to maximize the probability that a link exists in the KG through classification.

- Although ComplEx and TuckER are both tensor factorization methods, ComplEx represents embeddings in complex space and computes dot products for complex vectors such that positive links have a lower dot product value than negative links. This generates embeddings that generally have more negative values, due to the minimization procedure explained in section 2.6.3, which is leveraged later by Ding *et al.* [103] to introduce sparsity and interpretability to the embeddings. On the other hand, TuckER decomposes the binary tensor representation of the KG $B \in \{0,1\}^{|\mathcal{E}|\times|\mathcal{R}|\times|\mathcal{E}|}$ into $B \approx \mathcal{W} \times_1 \mathcal{E} \times_2 \mathcal{R} \times_3 \mathcal{E}$, such that $\mathcal{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$, $\mathcal{E} \in \mathbb{R}^{|\mathcal{E}|\times d_e}$, $\mathcal{R} \in \mathbb{R}^{|\mathcal{R}|\times d_r}$, and the product of matrices should be result in higher dot product for positive links. The objective between two tensor factorization methods differs due to their view of their KG model.

We perform hyperparameter tuning for those hyperparameters introduced by HOPLoP. To leverage computation gains from compiled static graphs generated by ML packages such Tensorflow [104], HOPLoP requires a pre-determined path length, which allows for compile time optimizations [1]. Therefore, we tune the maximum number of hops in a traversal H choosing from $\{1, 3, 5, 10, 15, 20\}$.

We provide a fair comparison between HOPLoP, M-HOPLoP and KG embedding models, we set the dimensionality of all embeddings to 100. We re-train for embeddings following the hyperparameters selected by the creators of the KG embedding. Embeddings and network parameters were optimized using Adam [59] with the default hyperparameter settings (initial learning rate $\mu = 0.001$).

[1] In later chapters, we see that HOPLoP learns to *not* hop; this is non-trivial because we do not explicitly provide HOPLoP with feedback regarding when not to hop, nor do we set up any constraints in the traversal process.

Tasks	M-HOPLoP (TransE)	M-HOPLoP (ComplEx)	M-HOPLoP (TuckER)	HOPLoP (TransE)	HOPLoP (ComplEx)	HOPLoP (TuckER)	DeepPath	PRA	TransE (Bernoulli)	ComplEx (Bernoulli)	TuckER
Team/Sport	**0.992**	0.904	0.883	**0.989**	**0.993**	**0.995**	0.955	0.987	0.924	0.985	0.953
Person/PlaceOfBirth	**0.967**	0.898	0.746	**0.986**	**0.960**	**0.958**	0.531	0.441	0.842	0.928	0.569
Person/Nationality	**0.993**	0.924	0.914	**0.964**	**0.977**	**0.981**	0.823	0.846	0.849	0.900	0.934
Film/Director	**0.670**	**0.657**	0.523	**0.654**	**0.679**	**0.671**	0.441	0.349	0.534	0.553	0.604
Film/WrittenBy	**0.945**	**0.867**	0.733	**0.994**	**0.978**	**0.972**	0.457	0.601	0.770	0.783	0.789
Film/Language	**0.931**	**0.908**	**0.781**	**0.971**	**0.971**	**0.935**	0.670	0.663	0.720	0.698	0.780
TvProgram/Languages	**0.992**	**0.985**	**0.971**	**0.984**	**0.987**	**0.979**	0.969	0.960	0.935	0.934	0.971
CapitalOf/Location	**0.916**	0.777	0.673	**0.906**	0.851	0.838	0.783	0.829	0.599	0.905	0.560
OrgFounder/OrgFounded	0.780	0.764	0.608	0.812	0.796	0.757	0.309	0.281	0.711	**0.864**	0.473
Artist/Origin	**0.937**	0.844	0.656	**0.966**	0.883	0.866	0.514	0.426	0.744	0.903	0.519
Overall	**0.914**	**0.826**	0.723	**0.864**	**0.859**	**0.849**	0.572	0.541	0.709	0.809	0.690

Table 5.4: Performance of (M-)HOPLoP against baseline path-based and embedding-based approaches to the relation prediction task on the FB15K-237 dataset. DIVA attained a MAP score of 0.598.

We evaluate HOPLoP on 2 tasks: Relation prediction and Entity prediction. We describe how HOPLoP can be used to perform both entity and relation prediction while comparing it to the state-of-the-art (SOTA) in both tasks.

5.2.1 Relation Prediction

The relation prediction task is commonly used to evaluate multi-hop LP algorithms [23, 48, 49, 94]. The reason for evaluating HOPLoP on relation prediction stems from its mathematical derivation, making relation prediction the intrinsic evaluation method. To recap, the relation prediction task is: given 2 entities, predict whether a particular relation exists between them.

Methodology Given a test source entity e_s, a relation r and a set of target entities $e_{t|q,r} = \{e_{t1}^+, e_{t2}^+, e_{t3}^-, e_{t4}^-, ...\}$, we compute the scores $\mathsf{HOPLoP}_r(e_s, e')$, computed by the logistic regression layer, for each entity in $e' \in e_{t|s,r}$. e_{ti}^+ refers to a positive target entity, whereas e_{ti}^- refers to a negative target entity. In the ideal situation, the scores for all positive target entities will be higher than all negative target entities, i.e., all positive target entities will be ranked higher than all negative target entities.

Following previous literature, our evaluation metric is the Mean Average Precision (MAP) score. Given a task, the MAP score is the mean AP across all source entities. AP is a strict metric since it penalizes when a negative target entity is ranked above a positive target entity.

We evaluate HOPLoP and M-HOPLoP in the relation prediction task on four datasets: NELL-995, FB15K-237, WN18RR and YAGO3-10.

5.2.2 Entity Prediction

The entity prediction task is commonly used to evaluate KG embedding-based LP algorithms [34]. Although HOPLoP is designed for relation prediction, we describe a procedure for the extrinsic evaluation of HOPLoP on the more-common entity pre-

diction task. To recap, the entity prediction task is: given a relation and a source entity, predict the set of target entities linked to the source entity via that relation.

Methodology Given a test triple (e_s, r, e_t), we calculate the score $\mathsf{HOPLoP}_r(e_s, e')$ for all possible e' where $e' \in \mathcal{E}$ is an entity, such that, (e_s, r, e') is not a link in the KG, unless $e' = e_t$. This allows us to compute metrics in the filtered setting [21], which does not penalize the model for ranking other correct target entities higher than the correct target entity e_t in question. We sort this list of scores in decreasing order and compute the rank for the correct target entity e_t. Since we do not train a HOPLoP model for every relation in the dataset, we consider the worst case prediction possible, i.e., the rank for the correct target entity e_t at worst is $|\mathcal{E}|$.

Following previous literature, we compute 2 types of metrics: Hits@k (H@k) $k \in \{1, 3, 10\}$ and Mean Reciprocal Rank (MRR) and evaluate on two standard datasets: WN18RR and YAGO3-10. Due to the high coverage of our datasets, we evaluate HOPLoP(TransE) on WN18RR and YAGO3-10, since HOPLoP models most of the relations in the KG. In the case of the previously introduced NELL-995 and FB15K-237 [48], the set of tasks is considerably smaller than the set of relations in the KG [2]. HOPLoP does not model many relations from these KGs and therefore, we cannot objectively compare HOPLoP's performance on these datasets on the entity prediction task. Nevertheless, we compare HOPLoP(TransE) against several SOTA KG embeddings on WN18RR and YAGO3-10, which have been extensively used for entity prediction.

[2]NELL-995: 12 tasks out of 200 relations (excluding inverse relations); FB15K-237: 20 tasks out of 237 relations

Tasks	M-HOPLoP (TransE)	M-HOPLoP (ComplEx)	M-HOPLoP (TuckER)	HOPLoP (TransE)	HOPLoP (ComplEx)	HOPLoP (TuckER)	TransE (Bernoulli)	ComplEx (Bernoulli)	TuckER
Hypernym	**0.962**	**0.743**	**0.590**	**0.968**	**0.860**	**0.865**	0.556	0.496	0.472
DerivationallyRelated	**0.989**	0.931	0.604	**0.993**	**0.977**	0.909	0.955	0.953	0.460
InstanceHypernym	**0.983**	**0.892**	0.814	**0.966**	**0.966**	**0.995**	0.844	0.700	0.811
AlsoSee	**0.864**	**0.695**	**0.486**	**0.938**	**0.841**	**0.844**	0.411	0.301	0.264
MemberMeronym	**0.656**	**0.339**	**0.334**	**0.706**	**0.643**	**0.898**	0.152	0.112	0.112
SynsetDomainTopic	**0.972**	**0.882**	0.814	**0.976**	**0.941**	**0.979**	0.856	0.524	0.771
HasPart	**0.760**	**0.651**	**0.510**	**0.832**	**0.738**	**0.906**	0.345	0.382	0.307
MemberDomainUsage	**0.656**	**0.645**	0.327	**0.571**	**0.518**	**0.482**	0.388	0.201	0.288
MemberDomainRegion	**0.673**	**0.575**	0.459	**0.669**	**0.652**	**0.696**	0.468	0.507	0.237
VerbGroup	0.957	0.843	0.548	**0.991**	**0.992**	0.816	0.843	0.965	0.347
Overall	**0.847**	**0.720**	0.549	**0.848**	**0.793**	**0.811**	0.582	0.514	0.407

Table 5.5: Performance of (M-)HOPLoP against baseline embedding-based approaches for relation prediction on the WN18RR dataset.

5.3 Results

Hypothesis: If a multi-hop LP algorithm is allowed to traverse the graph, unconstrained, then this will boost performance in the LP task. To this end, we proposed HOPLoP, which traverses the KG embedding space in a unconstrained yet controlled manner. In the context of HOPLoP, our hypothesis can be re-worded: If HOPLoP is trained, then it learns to uncover correlations between the controlled path traversals and the relation of interest, thus boosting LP performance. Table 5.6 observes that, without any training, HOPLoP does *not* perform as well as the baselines. This shows that the training process helps HOPLoP correlate traversal paths to a relation. This boost in LP performance can also be observed across all datasets and metrics.

Embedding	TransE	ComplEx	TuckER
Dataset			
NELL-995	0.465 | 0.828 | **0.934**	—	—
FB15K-237	0.403 | 0.709 | **0.864**	0.388 | 0.809 | **0.859**	0.371 | 0.690 | **0.849**
WN18RR	0.415 | 0.582 | **0.848**	0.403 | 0.514 | **0.793**	0.386 | 0.407 | **0.811**
YAGO3-10	0.422 | 0.545 | **0.908**	0.396 | 0.564 | **0.861**	—

Table 5.6: Performance comparison of `untrained HOPLoP (MAP)` | `baseline KG embedding (MAP)` | `trained HOPLoP (MAP)`.

5.3.1 Relation Prediction

We compare HOPLoP with embedding-based algorithms [21, 74, 75], supervised path traversal approaches [23], reinforced path traversal approaches [48, 49, 94]. We also compare with DIVA [84], a probabilistic approach tackling the relation prediction task, but report their results in the captions of tables since their paper does not disclose MAP scores for each relation. Table 5.3 reports the results from experiments involving the NELL-995 dataset, a simple dataset for which many existing algorithms observe very high accuracies. We evaluate HOPLoP over the FB15K-237

dataset, which is considered to be more challenging arguably more relevant for real-world scenarios than the NELL dataset [84]. Results from this experiment have been reported in Table 5.4, showing that our approach performs better than SOTA approaches. From Table 5.3 and Table 5.4, we can also observe that HOPLoP improves the performance of all KG embedding models. On the NELL-995 dataset, HOPLoP traverses over TransE's embedding space to boost performance by +0.106 MAP. HOPLoP boosts performances of TransE, ComplEx, TuckER by +0.137, +0.050 and +0.159 MAP on the FB15K-237 dataset. We also observe that less performant KG embedding models, such as TransE, are highly benefited by HOPLoP since their simple representation allows HOPLoP to easily understand the structure of the KG and create appropriately complex decision boundaries.

We also evaluate HOPLoP on two new datasets introduced for the task of multi-hop LP. We compare HOPLoP to baseline KG embedding models and observe improvements in MAP scores. Specifically, on the WN18RR dataset, HOPLoP improves the performance of TransE, ComplEx, and TuckER by +0.266, +0.279, +0.404 MAP respectively. Experiments on the YAGO3-10 dataset show that HOPLoP improves the performance of baseline KG embedding models TransE and ComplEx by +0.363 and +0.297 MAP respectively. Due to the enormous size of the YAGO3-10, coupled with computational constraints, we do not run experiments with TuckER.

M-HOPLoP

We observe that in M-HOPLoP (TransE) performs the best at modelling relations for the entire dataset, although, this cannot be said for individual tasks, where HOPLoP(TransE) usually outperforms. We observe that M-HOPLoP does not work as efficiently on complex embeddings spaces such as TuckER. The performance slightly degrades even when we shift M-HOPLoP from TransE to ComplEx. This trend is different compared to HOPLoP trained on individual tasks. M-HOPLoP's degradation in performance can be attributed to low parameter space, i.e, the path-finder and the

Tasks	M-HOPLoP (TransE)	M-HOPLoP (ComplEx)	HOPLoP (TransE)	HOPLoP (ComplEx)	TransE (Bernoulli)	ComplEx (Bernoulli)
IsAffiliatedTo	**0.970**	**0.921**	**0.941**	**0.965**	0.815	0.843
PlaysFor	**0.899**	**0.863**	**0.955**	**0.904**	0.778	0.814
IsLocatedIn	**0.873**	**0.830**	**0.918**	**0.824**	0.643	0.593
HasGender*	**0.967**	**0.963**	**0.973**	**0.971**	0.946	0.940
WasBornIn	**0.820**	**0.792**	**0.883**	**0.766**	0.415	0.390
ActedIn	**0.746**	**0.810**	**0.861**	**0.839**	0.369	0.397
IsConnectedTo	**0.826**	0.715	**0.979**	**0.948**	0.694	0.734
HasWonPrize	**0.807**	**0.790**	**0.843**	**0.878**	0.540	0.561
Influences	**0.745**	**0.754**	**0.869**	**0.714**	0.354	0.374
DiedIn	**0.824**	**0.800**	**0.842**	**0.776**	0.451	0.397
HasMusicalRole	**0.901**	**0.911**	**0.923**	**0.935**	0.646	0.538
GraduatedFrom	**0.850**	**0.763**	**0.853**	**0.891**	0.456	0.385
Created	**0.864**	**0.859**	**0.916**	**0.837**	0.523	0.604
WroteMusicFor	**0.768**	**0.795**	**0.896**	**0.940**	0.368	0.434
Directed	**0.831**	**0.864**	**0.942**	**0.840**	0.517	0.558
ParticipatedIn	**0.623**	**0.683**	**0.790**	**0.770**	0.476	0.552
HasChild	**0.765**	**0.815**	**0.966**	**0.841**	0.396	0.330
HappenedIn	**0.877**	**0.806**	**0.955**	**0.877**	0.600	0.628
IsMarriedTo	0.907	0.845	**1.000**	**0.990**	0.524	0.940
IsCitizenOf	**0.888**	**0.857**	**0.934**	**0.975**	0.634	0.548
WorksAt	**0.903**	**0.817**	**0.968**	**0.997**	0.581	0.621
Edited	**0.867**	**0.773**	**0.988**	**0.948**	0.328	0.426
LivesIn	**0.843**	**0.778**	**0.967**	**0.965**	0.484	0.356
Overall	**0.842**	**0.818**	**0.908**	**0.861**	0.545	0.564

Table 5.7: Performance of HOPLoP against baseline embedding-based approaches for relation prediction on the YAGO3-10 dataset. * Due to the biased nature of this task, a LP algorithm that always predicts `male` will achieve a MAP score of 0.963 in this task.

LSTM from HOPLoP remain the same in M-HOPLoP. The parameters used to model a relation still remains the same, but most of the parameters are shared across all relations of interest. Increasing the power of the neural architecture will definitely help, but M-HOPLoP (TransE) already achieves SOTA performance across the NELL-995, FB15K-237, WN18RR datasets with parameter sharing mechanisms. HOPLoP occucpies a parameter space of 381,601 to model 1 relation whereas M-HOPLoP occupies a parameter space of 383,116 (+1,515) (averaged $|\mathcal{K}|$ across datasets, see table 3.1) to model multiple relations.

5.3.2 Entity Prediction

We compare HOPLoP(TransE) with several SOTA KG embedding models such as [21, 74, 80, 102, 105–107] and multi-hop LP approaches [49, 50, 94] on WN18RR and YAGO3-10 dataset [3]. From tables 5.8 and 5.9 we see that HOPLoP(TransE) consistently outperforms previous SOTA approaches by a good margin. On the WN18RR dataset and YAGO3-10, HOPLoP(TransE) achieves an error reduction (with respect

[3]Results were obtained from https://paperswithcode.com/task/link-prediction

Model	MRR	H@1	H@3	H@10
M-Walk	0.437	0.414	0.445	—
ComplEx	0.440	0.410	0.460	0.510
MINERVA	0.448	0.413	0.456	0.513
TransE	0.466	0.423	—	0.556
TuckER	0.470	0.443	0.482	0.526
Reward Shaping (ComplEx)	0.472	0.437	—	0.542
ComplEx-N3	0.480	—	—	0.570
HOPLoP(TransE)	**0.760**	**0.753**	**0.767**	**0.790**

Table 5.8: Performance of HOPLoP(TransE) in the entity prediction task on the WN18RR dataset compared against SOTA multi-hop LP algorithms and KG embedding models.

Model	MRR	H@1	H@3	H@10
DistMult	0.340	0.240	0.380	0.540
ComplEx	0.360	0.260	0.400	0.550
ConvE	0.440	0.350	0.490	0.620
RotatE	0.495	0.402	0.550	0.670
RefE	0.577	0.503	0.621	0.712
ComplEx-N3	0.580	—	—	0.710
HOPLoP(TransE)	**0.818**	**0.817**	**0.818**	**0.820**

Table 5.9: Performance of HOPLoP(TransE) in the entity prediction task on the YAGO3-10 dataset compared against SOTA KG embedding models.

to MRR) of 53.85% and 56.67% respectively.

5.4 Analysis

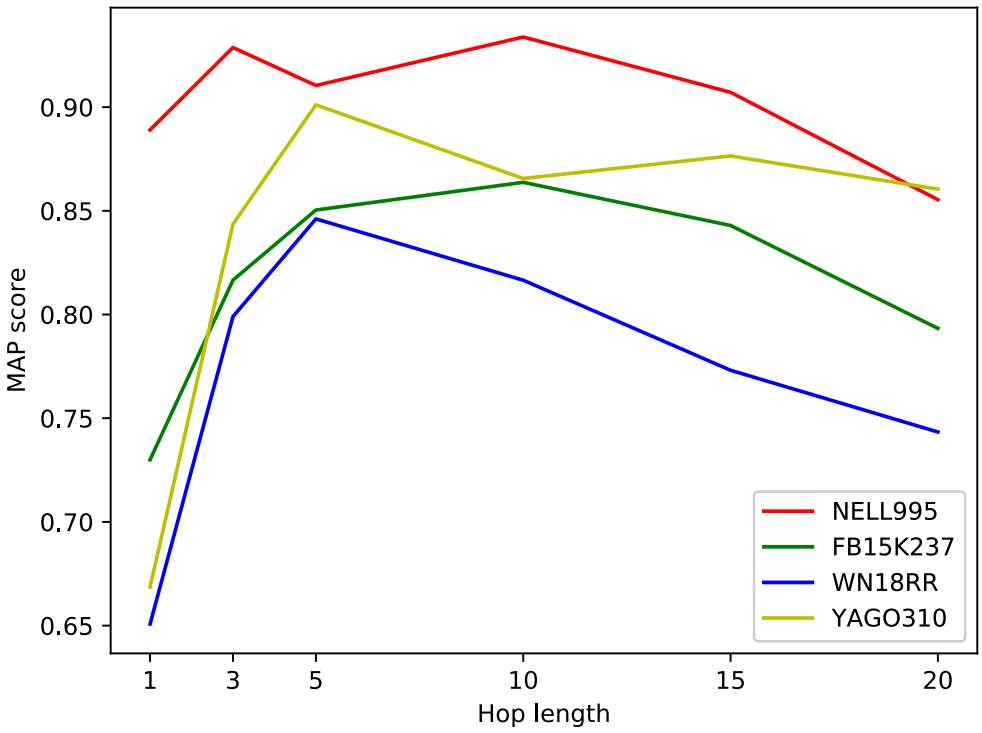
The main hypothesis of this paper is that the performance of a LP algorithm can be improved if it is allowed to leverage graph traversals that are not constrained by the KG. To verify this hypothesis, we introduced HOPLoP, a multi-hop link prediction framework that learns to traverse an embedding space, provided by a base KG embedding model. From Tables 5.3, 5.4, 5.5, and 5.7, we can observe that HOPLoP

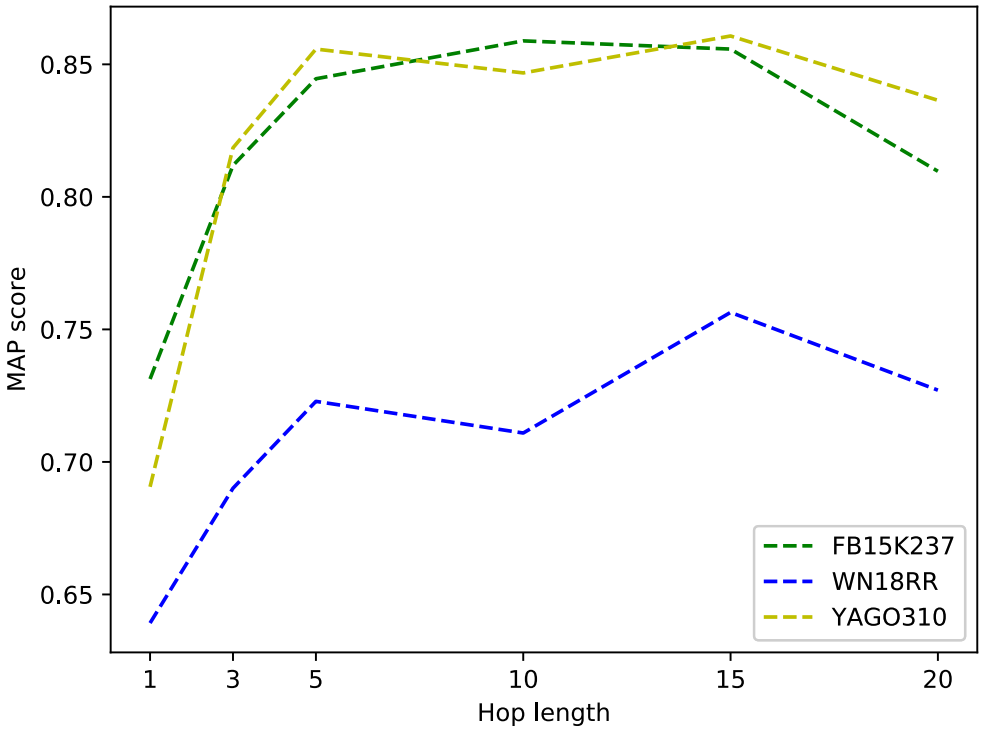
Model	Training (hrs.)	Testing (sec./sample)
MINERVA (S=3, L=100)	3	2×10^{-2}
M-Walk (S=5, M=128)	14	6×10^{-3}
HOPLoP(TransE) (H=1), fastest	3.7*	3.6×10^{-3}
HOPLoP(TransE) (H=5), best perf.	0.5	5.9×10^{-3}
HOPLoP(TransE) (H=20), worst	1.1	1.4×10^{-2}

Table 5.10: Runtime of HOPLoP(TransE) compared to M-Walk and MINERVA on the WN18RR dataset. S indicates search horizons, L indicates number of rollouts and M indicates the number of MCTS simulations. * Early stopping mechanism with a high patience of 100 epochs did not allow quick termination of the training process.

consistently outperforms baseline KG embedding models and previous SOTA multi-hop LP algorithms. We support our hypothesis with the intuition that traversing an embedding space uncovers correlations between paths and relations facilitating better understanding of the global structure through KG embeddings. To verify this, table 5.6 compares untrained versions of HOPLoP to baselines and the trained versions of HOPLoP to show that untrained HOPLoP models are not as performant as baseline KG embedding models and trained HOPLoP models. This shows that the training process helps HOPLoP discover correlations between paths and relations that help boost performance in the LP task. Futhermore, we show that HOPLoP is computationally inexpensive. In Table 5.10, we observe that the runtime of HOPLoP (Tensorflow-gpu) is lower than that of M-Walk (C++ & Cuda) and MINERVA (Tensorflow-gpu). We observe a 17x improvement in training times and 2x improvements in testing times.

Figure 5.1 shows us a pattern: there is a increase in LP performance as hops increases, but only to a certain extent. Further increasing the number of hops does not improve LP performance, it either stabilizes or degrades. This can be attributed to "overfitting". Since HOPLoP can traverse more expressive "edges", increasing the number of hops only increases it's expressivity, causing HOPLoP to overfit at higher hop values.


0.90
0.85
0.80
0.75
0.70
0.65
MAP score
1
3
5
10
15
20
Hop length
NELL995
FB15K237
WN18RR
YAGO310


0.85
0.80
0.75
0.70
0.65
MAP score
1
3
5
10
15
20
Hop length
FB15K237
WN18RR
YAGO310

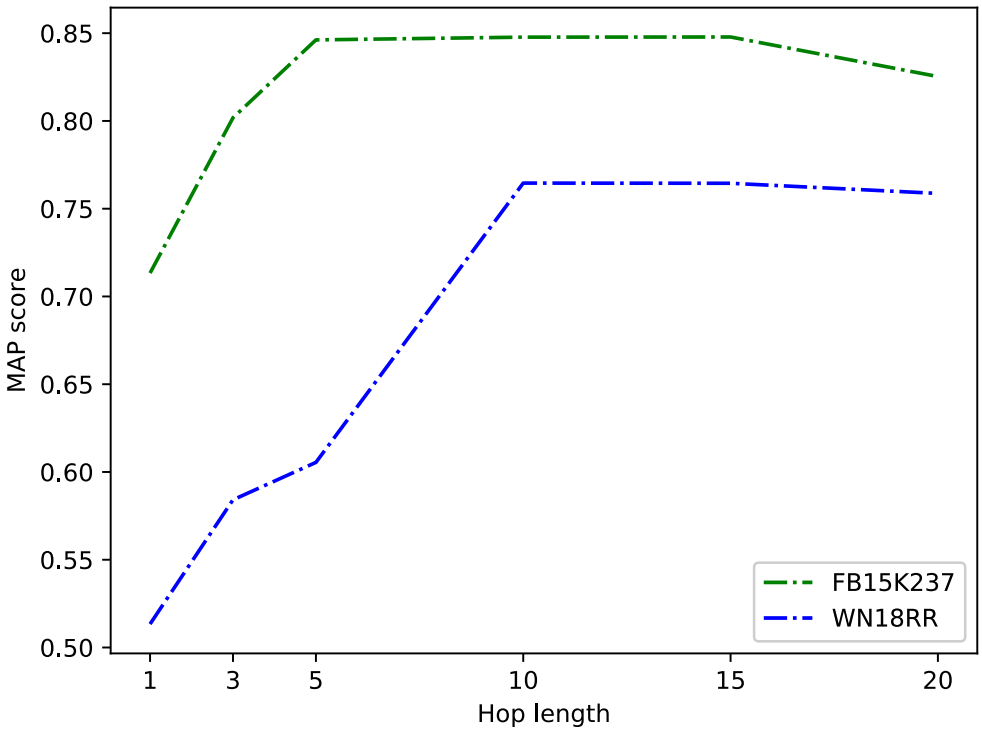


Figure 5.1: (Best viewed in color) Number of hops H vs MAP score plots for each dataset and embedding space. Red: NELL-995, Green: FB15K-237, Blue: WN18RR, Yellow: YAGO3-10. Top: TransE, Center: Complex, Bottom: TuckER. We see that there is a increase in LP performance as hops increases only to a limit. Further increasing the number of hops does not improve LP performance, it either stabilizes or degrades.

Chapter 6

Interpretability of HOPLoP

In general, a limitation of operating over an embedded space is that reasoning paths cannot always be easily interpreted because they result from latent factors that form the KG embedding space [10] and the weights learned by HOPLoP. HOPLoP(TransE), however, operates on the TransE embedding space, where both entities and relations are represented [21], geometrically allowing one to interpret it's reasoning process by observing the translation vectors $\vec{v_1} \dots \vec{v_H}$. This is not possible in the case of ComplEx and TuckER since, for those, HOPLoP traverses the embedded space where only entities are represented. Following the intuition that similar relationships will be represented by similar vector representations [10], we use a function of the euclidean distance as a similarity measure between a translation vector $\vec{v_h}$ and an embedding for a relation. Let $s(\vec{v_h}, \vec{r_i})$ be the similarity function.

$$s(\vec{v_h}, \vec{r_i}) = -\sqrt{\sum_{d=1}^{D} (\vec{v_{h_d}} - \vec{r_{i_d}})^2} \tag{6.1}$$

where D is the dimensionality of the vectors. Negating the euclidean distance allows us to produce a higher similarity score for more similar vectors.

Methodology Now, we shall describe our process for interpreting the traversal paths of HOPLoP(TransE). To generate interpretable relational paths for a given task r, we remove the relations r and r^{-1} from the graph. This forces beam-search

[108] to find other single-hop [1] or multi-hop relational paths that may represent the task. For the task `hasGender` in the YAGO3-10 dataset, we do not remove the relation `hasGender` from the dataset since it's range {`male`, `female`} is connected to other entities only via the `hasGender` relation. We also add a "NO-OP" relation, represented by a 0-vector, to interpret the scenario where HOPLoP(TransE) does not hop to a new entity. At each hop, we compute the similarity between the translation vector $\vec{v_h}$ and all relation embeddings $\vec{r_i} \in R$ as given by equation 6.1. The similarity scores are used to beam-search links, first exploring relations in the graph whose embedded representations are most similar to the translation vector and then exploring entities close to HOPLoP(TransE)'s next-hop position.

6.1 Example Paths and Their Interpretation

We provide several examples of interpreted HOPLoP(TransE) paths, describing relations of interest from different KGs. In each case, a positive path is one that connects a positive pair of entities, i.e., if a positive path exists between two entities, HOPLoP(TransE) tends to predict that a link, of relation type r, exists between them. Similarly, if a negative path exists between two entities, HOPLoP(TransE) tends to refute the existence of a link or relation type r between the two entities.

One-hop paths

`PersonLeadsOrganization` (NELL-995)

- `CeoOf`
- `PersonLeadsGeopoliticalOrganization`

`WorksFor` (NELL-995)

- `TopMemberOfOrganization`

[1] This would imply that the single-hop relation r' present in the graph is semantically similar to the task r.

- `OrganizationHiredPerson`$^{-1}$

`PersonBornInLocation` (NELL-995)

- `PersonBornInCity` (Positive)
- `PersonMovedToStateOrProvince` (Negative)

Multi-hop paths

`AthletePlaysSport` (NELL-995)

- **Positive**: `AthletePlaysForTeam` → `TeamPlaysSport`.

  Meaning: if an athlete A plays for team B and if team B is known to play sport C, then athlete A plays that sport C.

- **Negative**: `PersonHasCitizenship` → `SportFansInCountry`$^{-1}$

  Meaning: just because an athlete A has citizenship in a country B which contains fans of sport C, it does not imply athlete A plays sport C, since a country may contain fans of multiple sports.

`PersonBornInLocation` (NELL-995)

- **Positive**: `PersonHasCitizenship` → `CountryAlsoKnownAs`

  Meaning: if a person A has citizenship in a country B, it is highly probable that person A was born in country B.

- **Negative**: `PersonBelongsToOrganization` → `OrganizationAlsoKnownAs`$^{-1}$ → `AtLocation`

  Meaning: if a person A belongs to an organization B and it is located at C, it does not mean person A was born at location C, since a large number of people move for work.

`Ethnicity/LanguagesSpoken` (FB15K-237)

- **Positive**: `Ethnicity/GeographicDistribution` → `Country/OfficialLanguage`

  Meaning: if an ethnic group A is from a country B and country B's has an official language C, it is highly probable that the ethnic group B speaks language C.

- **Negative**: `Ethnicity/People` → `Actor/DubbingPerformances/Language`

  Meaning: if a dub actor A performs in language B, it does not mean that the ethnic group C of actor A can speak the language C, since dub actors learn to speak multiple languages.

`Event/Locations` (FB15K-237)

- **Positive**: `NcaaBasketballTournament/Team` → $\texttt{SportsTeamLocation/Teams}^{-1}$

  Meaning: If an NCAA Basketball Tournament A hosts a team B, and team B plays at location C, this implies that event A happened at location C.

- **Negative**: $\texttt{Film/FilmFestivals}^{-1}$ → $\texttt{NetflixGenre/Titles}^{-1}$ → $\texttt{Location/Contains}^{-1}$

  Meaning: If a Film Festival A hosts Film B and the film B is part of the `NetflixGenre` C and C contains location D, it does not imply that event A happened at location D.

`hasGender` (YAGO3-10)

- **Positive**: `playsFor` → $\texttt{isAffiliatedTo}^{-1}$ → `hasGender`

  Meaning: If an athlete A plays for team B, another player C is affiliated with team B and player C has gender D, then player A also has gender D. This is because clubs form different teams for each gender.

- **Negative**: `isMarriedTo`$^{-1}$ $\rightarrow$ `hasGender`

  Meaning: If a person A is married to another person B who has gender C, person A most likely does not have the gender C of their spouse.

`graduatedFrom` (YAGO3-10)

- **Positive**: `hasAcademicAdvisor` $\rightarrow$ `worksAt`

  Meaning: If a person A has an academic advisor B who works at organization C, the likelihood of person A graduating from C is high.

- **Negative**: `wasBornIn` $\rightarrow$ `isLocatedIn` $\rightarrow$ `isLocatedIn`$^{-1}$

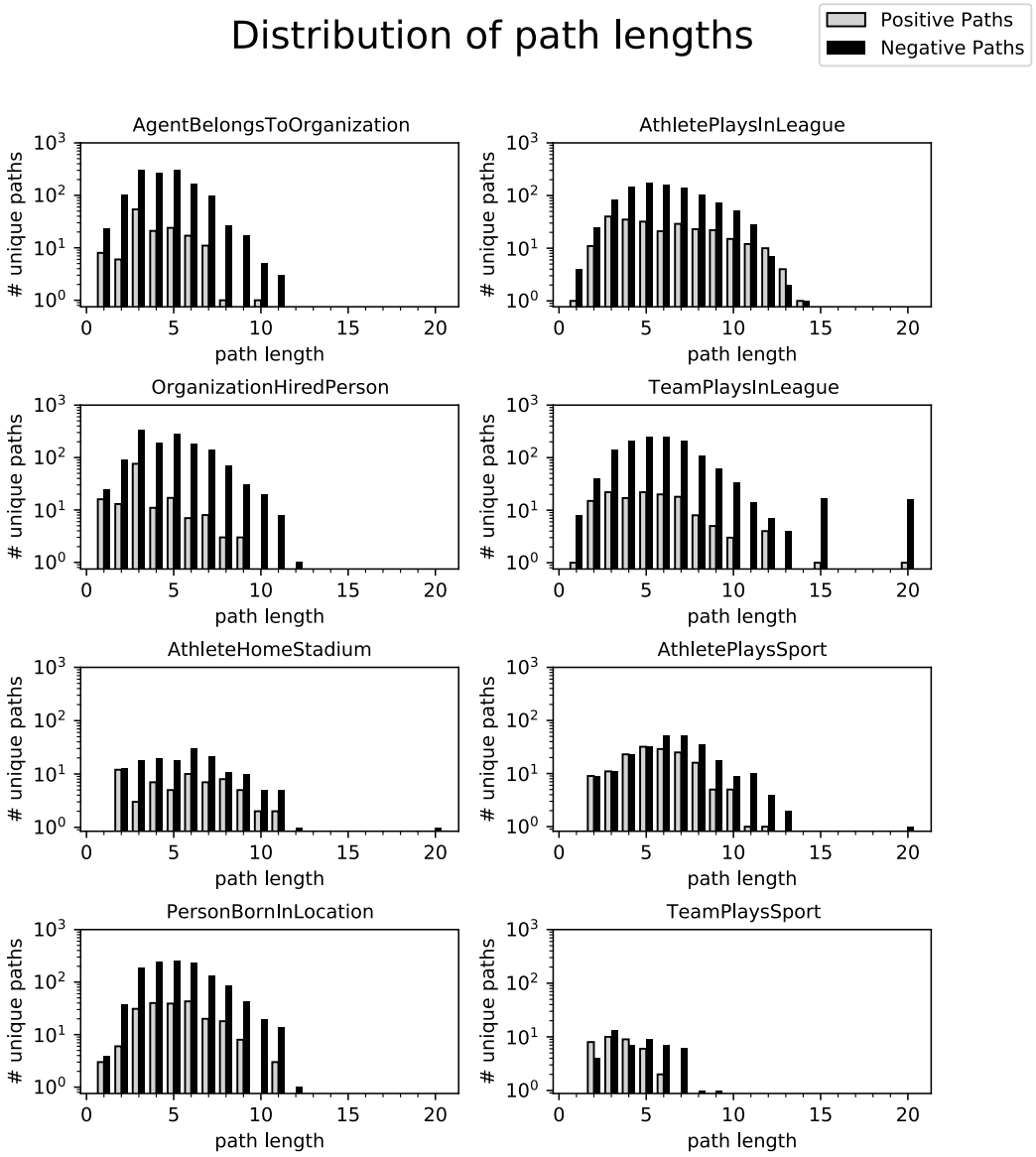
  Meaning: If a person A was born in location B and a particular school C is located at B, it does not imply that person A graduated from school C, since many students graduate from universities far from their birth place.

6.2 Distribution of Path Lengths

Figure 6.1 presents the distribution of number of unique paths by path length of all tasks from the NELL-995 dataset. In this case, a unique path is a unique sequence of *both* relations and entities, i.e., $p = [e_s, v_1, e_1, ..., v_H, e_t]$. This examples the high number of unique paths observed, as compared to previous methods, which only look at relations in the path. We observe that the number of unique negative paths is higher than the number of unique positive paths. We also observe that the average path length for positive paths is consistently lower than the average path length for negative paths across all tasks.

These observations show the extent to which HOPLoP "explores" the embedded KG space in search for paths with strong support either way. Figure 6.1 also shows that some tasks have a fairly unique distributions, which indicates that HOPLoP can adapt to them. We observe that HOPLoP(TransE) utilizes the "NO-OP" relation, which does not change it current entity position. This explains the observation that

path lengths rarely cross 10, since "NO-OP" is *not* a relation in the KG. We also observe that, for a few tasks, a substantial number of paths have been found exceeding path lengths 15. This can be attributed to the "over-fitting" scenario (see section 5.4), in which HOPLoP(TransE) creates unique traversal patterns that tend to overfit for the data provided.



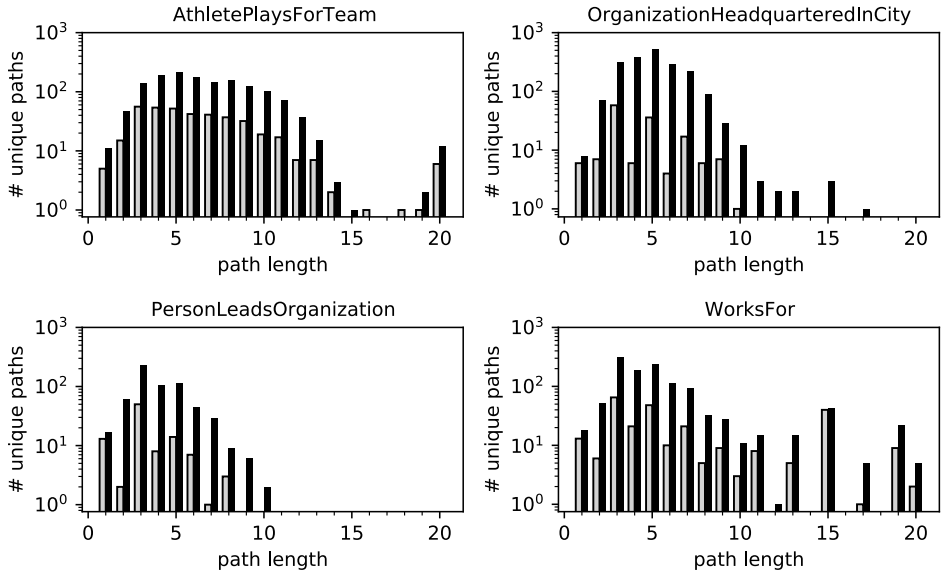


Figure 6.1: Distribution of number of unique paths founds by path length for each task in the NELL-995 dataset. The y-axis is log-scaled. The light grey bars represent number of unique positive paths. The black bars represent number of unique negative paths. We observe that HOPLoP(TransE) finds more negative paths than positive paths. We also observe that path length rarely cross 10. This shows that HOPLoP(TransE) is utilizing the "NO-OP" operation, which is *not* counted as a relation in the KG. We also observe that, for a few tasks, a substantial number of paths have been found exceeding path lengths 15. This can be attributed to the "over-fitting" scenario (see section 5.4).

Chapter 7

Conclusion and Future Work

In this thesis, we explored the space of automated reasoning over large-scale KGs. Specifically, we looked into the problem of Knowledge Base Completion (KBC), which aims to fill the informational gaps by reasoning over the observed links in the KG. Adopted from graph theory, LP is a popular technique used to tackle KBC. A few issues that hinders LP are: incompleteness [14], duplicate entities and relations [84] and skewed node degrees [48]. LP is dominated by KG embedding methods, which offer an elegant approach to the incompleteness problem. Different from KG embedding methods, multi-hop LP aims to find a traversal path in the graph such that the path expresses the underlying relation of interest.

We hypothesized that the performance of a multi-hop LP algorithm may be improved if it traverses over a more-complete representation of the KG. This means that there should be more links and entities in the graph to allow the path-finder to be more expressive. We take advantage of the existing KG embedding literature, and introduce a novel framework for multi-hop LP where paths and links are defined in the embedded space. Our framework, HOPLoP, is an end-to-end differentiable framework that traverses an entity embedding space to distinguish between existent and non-existent links of the KG. HOPLoP is unaffected by the skewed node degree and the incompleteness issue, since all entities are represented as points in space and the traversal process is unconstrained, allowing HOPLoP to take any edge in the

embedded representation of the KG.

We notice that the training process helps HOPLoP recognize the global structure of the KG and uncover correlations between traversal paths and relations. This significantly boosts LP performance, which can be attributed to the expressive but controlled traversal in embedded space. By allowing an unconstrained traversal process, HOPLoP is able to better express the underlying relation between the source and target entity, which is *understood* by the path-reasoner, which performs the final link prediction. Due to the inherent differentiability of our framework, parameter sharing mechanisms for multi-task learning [30], as demonstrated by Das *et al.* [31], were explored by M-HOPLoP. M-HOPLoP can reason about multiple relations and performance better than HOPLoP in LP over the entire dataset, with only a marginal increase in parameter size (+0.397%).

Upon performing standard LP evaluation practices, we observe that HOPLoP and M-HOPLoP outperforms previous SOTA multi-hop and KG embedding approaches across 4 datasets and 2 variants of the LP task. On the intrinsic relation prediction task, M-HOPLoP advances previous SOTA methods on the NELL-995 and FB15K2-237 dataset by +0.047 and +0.105 MAP, reducing errors by 46.53% and 54.97% respectively. On the extrinsic entity prediction task, HOPLoP(TransE) advances previous SOTA methods on the WN18RR and YAGO3-10 dataset by +0.280 and +0.238 MRR, with an error reduction of 53.85% and 56.67% respectively. We also described a method to interpret HOPLoP(TransE)'s reasoning paths. Experiment codes, scripts and additional materials can be obtained at https://www.github.com/varunranga/HOPLoP. Refer to Appendix B for more details regarding supplementary and reproducibility.

7.1 Applicability of HOPLoP

Similar to all KG embedding methods, HOPLoP provides a score for a given fact. The adoption of HOPLoP would be similar to any KG embedding method, but would

require an existing embedding space. Motivated practitioners may use the scripts available in the supplementary material to generate an embedding space. To replace an existing embedding space in use would reap the highest benefits. HOPLoP can be trained to operate over any embedding space to directly replace that KG embedding method for LP, without any change in the training pipeline. Since HOPLoP uses separate parameters, it would *not* "fine-tune" the embeddings, which might be in use by different ML systems.

7.2 Limitations of this work

A limitation of this work is that the KG benchmark datasets, used to compare various LP algorithms, are artificial. These datasets are extracted from real-world KGs but the test set is artificially created. To create these datasets, authors usually capture a sub-graph of a real-world KG, randomly select links for the test set, and remove the selected links from the sub-graph to create a training set of links. Since KGs only store the observed facts as links, authors generally hypothesize negative examples for links, which is then used for training and evaluation purposes. This is different from the case of creating a KG (e.g. YAGO [4]) where the evaluation methodology on the accuracy and size of KGs can be quantitative compared against other KGs.

Although HOPLoP advances the state of the art in LP, the results of the experiments cannot be directly translated to a rate of improvement for the KG. This is due to the limitations in evaluation methodologies followed in the LP literature. LP evaluation methodologies follows a closed-world assumption where all the facts observed in the KG are deemed "positive" while any fact that is not observed in the KG is deemed "negative". The results shown are indicative of HOPLoP's LP performance on the test set of the KGs used in experiments. Since we do not make changes to real-world KGs, we cannot quantitatively describe the impact of HOPLoP on real-world KGs. Similar to the study of linked data by Radulovic *et al.* [109], we argue that a comprehensive study of the quality of large-scale KGs, along with the impact of all

KBC methods, is essential to understand to true impact of LP algorithms.

How might one use HOPLoP to quantify the improvement of a KG? We believe that quantifying the rate of improvement of a KG will require a manual inspection of predicted facts. To do this, we can use HOPLoP to predict links in the KG, which can be used to add facts to the KG. For a given relation, first train HOPLoP with positive and negative links for that relation. Use the trained HOPLoP model to predict for links by iterating over all entities in the range and domain of the relation. For example, if there are 5 entities in the range of a relation, and 3 entities in the domain of the relation, if the range and domain do not share any entities, we can predict links between 15 pairs of entities. Using HOPLoP's probabilistic prediction for a link, we can use post-processing techniques from section 2.2.1 to determine which facts should be added to the KG and which facts should not be added. For example, one might identify a threshold for prediction, i.e., if HOPLoP's predicted value is greater than this threshold, we can assume that the link should be present or added to the KG. To quantitatively measure the rate of improvement, a manual inspection of the predicted links must be done before adding the links to the real-world KG. This manual inspection will involve examining the predictions for each link, while manual determining whether the link actually expresses a fact in the world. By comparing the predictions for links against some ground truth, we will be able to quantify the improvement to a KG, with respect to the number of new facts added and the accuracy of the information that is being added to the graph.

7.3 Future research directions

We believe this new approach of "generating" traversal paths over an embedded space can shed light onto a new approach for modeling sequence to sequence tasks. Path lengths are not limited by the size of the model, and thus, they may be the answer to expressing lengthy and ∞-length sequences. Paths can be created, on the go, by an hypothetical one-hop model, similar to HOPLoP, that receives its current state

and a new input, and performs composition, to "move" to a new hidden state. This approach can be used to create representations for sequences, which then can be analyzed by a stronger "reasoning" model.

Recent applications of BP, such as KG embeddings, have shown that we can take advantage of GD for structure learning. We show that we can use GD for sequence creation and learning as well. We have also witnessed progress in sequence tokenization [1], enabling us to perform online machine learning and prediction. The next question we would like to investigate: can we create universal representations for data? Similar to how a KG is a standard human-readable data-model, we may create a "Sequence Space" system, which can create neural representations for data. These neural representations can be used to train neural architectures to perform downstream ML tasks. This work opens up opportunities for a "data-centric" framework for machine learning.

[1] https://huggingface.co/docs/tokenizers/python/latest/

Bibliography

[1] P. Liang, "Talking to Computers in Natural Language," *XRDS*, vol. 21, no. 1, pp. 18–21, Oct. 2014, ISSN: 1528-4972. DOI: 10.1145/2659831.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "DBpedia: A Nucleus for a Web of Open Data," in *The Semantic Web*, Berlin, Germany: Springer Berlin Heidelberg, 2007, pp. 722–735, ISBN: 978-3-540-76298-0. [Online]. Available: https://link.springer.com/content/pdf/10.1007\%2F978-3-540-76298-0_52.pdf.

[3] G. A. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995, ISSN: 0001-0782. DOI: 10.1145/219717.219748.

[4] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A Core of Semantic Knowledge," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07, Banff, Canada: Association for Computing Machinery, 2007, pp. 697–706, ISBN: 9781595936547. DOI: 10.1145/1242572.1242667.

[5] A. Moschitti, K. Tymoshenko, P. Alexopoulos, A. Walker, M. Nicosia, G. Vetere, A. Faraotti, M. Monti, J. Z. Pan, H. Wu, and Y. Zhao, "Question Answering and Knowledge Graphs," in *Exploiting Linked Data and Knowledge Graphs in Large Organisations*. Cham, Switzerland: Springer International Publishing, 2017, pp. 181–212, ISBN: 978-3-319-45654-6. DOI: 10.1007/978-3-319-45654-6_7.

[6] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001, ISSN: 00368733, 19467087. [Online]. Available: http://www.jstor.org/stable/26059207.

[7] A. R. W. Tjiptomongsoguno, A. Chen, H. M. Sanyoto, E. Irwansyah, and B. Kanigoro, "Medical Chatbot Techniques: A Review," in *Software Engineering Perspectives in Intelligent Systems*, Cham, Switzerland: Springer International Publishing, 2020, pp. 346–356, ISBN: 978-3-030-63322-6.

[8] Q. Guo, F. Zhuang, C. Qin, H. Zhu, X. Xie, H. Xiong, and Q. He, "A Survey on Knowledge Graph-Based Recommender Systems," *IEEE Transactions on Knowledge & Data Engineering*, no. 01, ISSN: 1558-2191. DOI: 10.1109/TKDE.2020.3028705.

[9] R. Wickramarachchi, C. A. Henson, and A. P. Sheth, "An Evaluation of Knowledge Graph Embeddings for Autonomous Driving Data: Experience and Practice," in *Proceedings of the AAAI 2020 Spring Symposium on Combining Machine Learning and Knowledge Engineering in Practice, AAAI-MAKE 2020, Palo Alto, CA, USA, March 23-25, 2020, Volume I*, ser. CEUR Workshop Proceedings, vol. 2600, CEUR-WS.org, 2020. [Online]. Available: http://ceur-ws.org/Vol-2600/paper3.pdf.

[10] F. Bianchi, G. Rossiello, L. Costabello, M. Palmonari, and P. Minervini, "Knowledge Graph Embeddings and Explainable AI," pp. 49–72, 2020. DOI: 10.3233/SSW200011.

[11] J. Robinson and A. Voronkov, *Handbook of Automated Reasoning (in two volumes)*. Cambridge, USA: MIT Press, 2001, ISBN: 978-0-262-18221-8.

[12] N. Aggarwal, S. Shekarpour, S. Bhatia, and A. Sheth, "Knowledge graphs: In theory and practice," in *Conference on Information and Knowledge Management*, vol. 17, 2017.

[13] E. Hovy, R. Navigli, and S. P. Ponzetto, "Collaboratively built semi-structured content and Artificial Intelligence: The story so far," *Artificial Intelligence*, vol. 194, pp. 2–27, 2013, ISSN: 0004-3702. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370212001245.

[14] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A Review of Relational Machine Learning for Knowledge Graphs," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 11–33, Jan. 2016, ISSN: 1558-2256. DOI: 10.1109/JPROC.2015.2483592.

[15] K. Adnan and R. Akbar, "Limitations of information extraction methods and techniques for heterogeneous unstructured big data," *International Journal of Engineering Business Management*, vol. 11, 2019. DOI: 10.1177/1847979019890771.

[16] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic Web*, vol. 8, no. 3, pp. 489–508, 2017. DOI: 10.3233/SW-160218. [Online]. Available: https://madoc.bib.uni-mannheim.de/41515/.

[17] H. Cai, V. W. Zheng, and K. C. Chang, "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, Sep. 2018, ISSN: 1558-2191. DOI: 10.1109/TKDE.2018.2807452.

[18] C. Meilicke, M. W. Chekol, D. Ruffinelli, and H. Stuckenschmidt, "Anytime bottom-up rule learning for knowledge graph completion," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, Jul. 2019, pp. 3137–3143. DOI: 10.24963/ijcai.2019/435.

[19] A. Rossi, D. Barbosa, D. Firmani, A. Matinata, and P. Merialdo, "Knowledge Graph Embedding for Link Prediction: A Comparative Analysis," *ACM Transactions on Knowledge Discovery from Data*, vol. 15, no. 2, Jan. 2021, ISSN: 1556-4681. DOI: 10.1145/3424672.

[20] K. Guu, J. Miller, and P. Liang, "Traversing knowledge graphs in vector space," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 318–327. DOI: 10.18653/v1/D15-1038.

[21] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, and O. Yakhnenko, "Translating Embeddings for Modeling Multi-Relational Data," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'13, Lake Tahoe, USA: Curran Associates Inc., 2013, pp. 2787–2795. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

[22] N. Lao and W. W. Cohen, "Relational retrieval using a combination of path-constrained random walks," *Machine Learning*, vol. 81, no. 1, 53—67, Oct. 2010, ISSN: 0885-6125. DOI: 10.1007/s10994-010-5205-8.

[23] N. Lao, T. Mitchell, and W. W. Cohen, "Random Walk Inference and Learning in a Large Scale Knowledge Base," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '11, Edinburgh, United Kingdom: Association for Computational Linguistics, 2011, pp. 529–539, ISBN: 9781937284114. [Online]. Available: https://aclanthology.org/D11-1049.

[24] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge Graph Embedding: A Survey of Approaches and Applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, Dec. 2017, ISSN: 1558-2191. DOI: 10.1109/TKDE.2017.2754499.

[25] M. Nickel, V. Tresp, and H.-P. Kriegel, "A Three-Way Model for Collective Learning on Multi-Relational Data," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11, Bellevue, USA: Omnipress, 2011, 809—816, ISBN: 9781450306195.

[26] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge Graph Embedding by Translating on Hyperplanes," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, Jun. 2014. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/8870.

[27] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning Entity and Relation Embeddings for Knowledge Graph Completion," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, Feb. 2015. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/9491.

[28] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge Graph Embedding via Dynamic Mapping Matrix," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 687–696. DOI: 10.3115/v1/P15-1067.

[29] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[30] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[31] R. Das, A. Neelakantan, D. Belanger, and A. McCallum, "Chains of Reasoning over Entities, Relations, and Text using Recurrent Neural Networks," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 132–141. [Online]. Available: https://www.aclweb.org/anthology/E17-1013.

[32] A. Neelakantan, B. Roth, and A. McCallum, "Compositional Vector Space Models for Knowledge Base Completion," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 156–166. DOI: 10.3115/v1/P15-1016.

[33] H. Ji and R. Grishman, "Knowledge Base Population: Successful Approaches and Challenges," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, ser. HLT '11, Portland, USA: Association for Computational Linguistics, 2011, pp. 1148–1158, ISBN: 9781932432879. [Online]. Available: https://aclanthology.org/P11-1115.

[34] R. Kadlec, O. Bajgar, and J. Kleindienst, "Knowledge Base Completion: Baselines Strike Back," in *Proceedings of the 2nd Workshop on Representation Learning for NLP*, Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 69–74. DOI: 10.18653/v1/W17-2609.

[35] C. Grosan and A. Abraham, "Rule-based expert systems," in *Intelligent Systems: A Modern Approach*. Berlin, Germany: Springer Berlin Heidelberg, 2011, pp. 149–185, ISBN: 978-3-642-21004-4. DOI: 10.1007/978-3-642-21004-4_7.

[36] D. A. Ferrucci, "IBM's Watson/DeepQA," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11, San Jose, USA: Association for Computing Machinery, 2011, ISBN: 9781450304726. DOI: 10.1145/2000064.2019525.

[37] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling, "Never-Ending Learning," *Communications of the ACM*, vol. 61, no. 5, pp. 103–115, Apr. 2018, ISSN: 0001-0782. DOI: 10.1145/3191513.

[38] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A Survey on Knowledge Graphs: Representation, Acquisition, and Applications," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2021, ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3070843.

[39] G. Hinton and T. J. Sejnowski, *Unsupervised Learning: Foundations of Neural Computation*. The MIT Press, May 1999, ISBN: 9780262288033. DOI: 10.7551/mitpress/7011.001.0001.

[40] M. Nickel, V. Tresp, and H.-P. Kriegel, "Factorizing YAGO: Scalable Machine Learning for Linked Data," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12, Lyon, France: Association for Computing Machinery, 2012, pp. 271–280, ISBN: 9781450312295. DOI: 10.1145/2187836.2187874.

[41] J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[42] Q. Xie, Z. Dai, E. Hovy, T. Luong, and Q. Le, "Unsupervised Data Augmentation for Consistency Training," in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 6256–6268. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/44feb0096faa8326192570788b38c1d1-Paper.pdf.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[44] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[45] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[46] A. Gatt and E. Krahmer, "Survey of the State of the Art in Natural Language Generation: Core Tasks, Applications and Evaluation," *Journal of Artificial Intelligence Research*, vol. 61, no. 1, pp. 65–170, Jan. 2018, ISSN: 1076-9757.

[47] B. Kotnis and V. Nastase, "Analysis of the impact of negative sampling on link prediction in knowledge graphs," *arXiv preprint arXiv:1708.06816*, 2017.

[48] W. Xiong, T. Hoang, and W. Y. Wang, "DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 564–573. DOI: 10.18653/v1/D17-1060.

[49] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum, "Go for a Walk and Arrive at the Answer: Reasoning Over Paths in Knowledge Bases using Reinforcement Learning," in *ICLR*, 2018.

[50] X. V. Lin, R. Socher, and C. Xiong, "Multi-Hop Knowledge Graph Reasoning with Reward Shaping," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 3243–3253. DOI: 10.18653/v1/D18-1362.

[51] F. Chollet, "On the measure of intelligence," *arXiv preprint arXiv:1911.01547*, 2019.

[52] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html.

[53] P. Radivojac and M. White, "Machine learning handbook," 2019. [Online]. Available: https://marthawhite.github.io/mlcourse/notes.pdf.

[54] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999, ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6.

[55] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the Importance of Initialization and Momentum in Deep Learning," ICML'13, III–1139–III–1147, 2013.

[56] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Artificial Intelligence Research*, vol. 12, pp. 2121–2159, Jul. 2011, ISSN: 1532-4435.

[57] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," [Online]. Available: http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf.

[58] M. D. Zeiler, "AdaDelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[59] "Adam: A method for stochastic optimization," 2014. [Online]. Available: http://arxiv.org/abs/1412.6980.

[60] S. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and Beyond," 2018.

[61] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[62] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: 10.1038/nature14539.

[63] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0.

[64] V. Ranganathan and A. Lewandowski, *ZORB: A derivative-free backpropagation algorithm for neural networks*, 2020.

[65] Y. LeCun, L. Jackel, L. Bottou, A Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *International Conference on Artificial Neural Networks*, Perth, Australia, vol. 60, 1995, pp. 53–60. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-95b.pdf.

[66] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," *arXiv preprint arXiv:1803.01164*, 2018.

[67] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001. DOI: 10.1.1.101.2647.

[68] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Transactions on Nuclear Science*, vol. 44, no. 3, pp. 1464–1468, 1997. DOI: 10.1109/23.589532.

[69] M. I. Jordan, "Chapter 25 - Serial Order: A Parallel Distributed Processing Approach," in *Neural-Network Models of Cognition*, ser. Advances in Psychology, vol. 121, North-Holland, 1997, pp. 471–495. DOI: 10.1016/S0166-4115(97)80111-2.

[70] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, USA: PMLR, Jun. 2013, pp. 1310–1318. [Online]. Available: http://proceedings.mlr.press/v28/pascanu13.html.

[71] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[72] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014. [Online]. Available: https://arxiv.org/abs/1412.3555.

[73] E. Keogh and A. Mueen, "Curse of Dimensionality," in *Encyclopedia of Machine Learning and Data Mining*. Boston, USA: Springer US, 2017, pp. 314–315, ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_192.

[74] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, and G. Bouchard, "Complex Embeddings for Simple Link Prediction," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, USA: Journal of Machine Learning Research, 2016, pp. 2071–2080. [Online]. Available: http://proceedings.mlr.press/v48/trouillon16.pdf.

[75] I. Balazevic, C. Allen, and T. Hospedales, "TuckER: Tensor Factorization for Knowledge Graph Completion," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5185–5194. DOI: 10.18653/v1/D19-1522.

[76] Y. Pinter and J. Eisenstein, "Predicting Semantic Relations using Global Graph Properties," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 1741–1751. DOI: 10.18653/v1/D18-1201.

[77] H. Thanh-Tung and T. Tran, "Catastrophic forgetting and mode collapse in GANs," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–10. DOI: 10.1109/IJCNN48605.2020.9207181.

[78] V. Ranganathan and N. Subramanyam, "SDE-KG: A Stochastic Dynamic Environment for Knowledge Graphs," in *Machine Learning and Knowledge Discovery in Databases*, Cham, Switzerland: Springer International Publishing, 2020, pp. 483–488, ISBN: 978-3-030-43823-4.

[79] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Rev.*, vol. 51, no. 3, 455—500, Aug. 2009, ISSN: 0036-1445. DOI: 10.1137/07070111X.

[80] B. Yang, S. W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding Entities and Relations for Learning and Inference in Knowledge Bases," in *Proceedings of the International Conference on Learning Representations (ICLR) 2015*, May 2015. [Online]. Available: https://www.microsoft.com/en-us/research/publication/embedding-entities-and-relations-for-learning-and-inference-in-knowledge-bases/.

[81] M. Aceves-Fernandez, R. Domínguez-Guevara, J. C. Pedraza Ortega, and J. Vargas-Soto, "Evaluation of Key Parameters Using Deep Convolutional Neural Networks for Airborne Pollution (PM10) Prediction," *Discrete Dynamics in Nature and Society*, vol. 2020, pp. 1–14, Feb. 2020. DOI: 10.1155/2020/2792481.

[82] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966. DOI: 10.1007/BF02289464.

[83] M. Bianchini, M. Gori, and F. Scarselli, "Inside PageRank," *ACM Transactions on Internet Technology*, vol. 5, no. 1, pp. 92–128, Feb. 2005, ISSN: 1533–5399. DOI: 10.1145/1052934.1052938.

[84] W. Chen, W. Xiong, X. Yan, and W. Y. Wang, "Variational Knowledge Graph Reasoning," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, USA: Association for Computational Linguistics, Jun. 2018, pp. 1823–1832. DOI: 10.18653/v1/N18-1165.

[85] M. Gardner, A. Bhowmick, K. Agrawal, and D. Dua, "Experimenting with the path ranking algorithm," 2015. [Online]. Available: https://www.andrew.cmu.edu/user/kdagrawa/documents/10605report.pdf.

[86] M. Gardner, P. Talukdar, J. Krishnamurthy, and T. Mitchell, "Incorporating Vector Space Similarity in Random Walk Inference over Knowledge Bases," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 397–406. DOI: 10.3115/v1/D14-1044.

[87] V. Ranganathan, S. Suresh, Y. Mathur, N. Subramanyam, and D. Barbosa, "GrCluster: A Score Function to Model Hierarchy in Knowledge Graph Embeddings," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20, Brno, Czech Republic: Association for Computing Machinery, 2020, 964–971, ISBN: 9781450368667. DOI: 10.1145/3341105.3373978.

[88] P. Kolyvakis, A. Kalousis, and D. Kiritsis, "Hyperbolic Knowledge Graph Embeddings for Knowledge Base Completion," in *The Semantic Web*, Cham: Springer International Publishing, 2020, pp. 199–214, ISBN: 978-3-030-49461-2. DOI: 10.1007/978-3-030-49461-2_12.

[89] M. Nickel and D. Kiela, "Poincaré Embeddings for Learning Hierarchical Representations," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/59dfa2df42d9e3d41f5b02bfc32229dd-Paper.pdf.

[90] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao, and S. Liu, "Modeling Relation Paths for Representation Learning of Knowledge Bases," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 705–714. DOI: 10.18653/v1/D15-1082.

[91] S. Raghavan and H. Garcia-Molina, "Representing Web graphs," in *Proceedings 19th International Conference on Data Engineering*, Mar. 2003, pp. 405–416. DOI: 10.1109/ICDE.2003.1260809.

[92] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, USA: A Bradford Book, 2018, ISBN: 0262039249.

[93] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992. DOI: 10.1007/BF00992696.

[94] Y. Shen, J. Chen, P.-S. Huang, Y. Guo, and J. Gao, "M-Walk: Learning to Walk over Graphs using Monte Carlo Tree Search," in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018, pp. 6786–6797. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/c6f798b844366ccd65d99bc7f31e0e02-Paper.pdf.

[95] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[96] Y. Gal, "Uncertainty in Deep Learning," PhD thesis, University of Cambridge, 2016.

[97] F. Mahdisoltani, J. Biega, and F. M. Suchanek, "YAGO3: A Knowledge Base from Multilingual Wikipedias," in *7th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015. [Online]. Available: http://cidrdb.org/cidr2015/Papers/CIDR15_Paper1.pdf.

[98] J. Halpern, *Reasoning about Uncertainty*, ser. Reasoning about Uncertainty. MIT Press, 2005, ISBN: 9780262582599. [Online]. Available: https://books.google.com.vc/books?id=qMBHGwAACAAJ.

[99] D. Hosmer and S. Lemeshow, *Applied Logistic Regression*, ser. Applied Logistic Regression. Wiley, 2004, ISBN: 9780471654025. [Online]. Available: https://books.google.co.in/books?id=Po0RLQ7USIMC.

[100] K. Toutanova, V. Lin, W.-t. Yih, H. Poon, and C. Quirk, "Compositional Learning of Embeddings for Relation Paths in Knowledge Base and Text," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1434–1444. DOI: 10.18653/v1/P16-1136.

[101] M. Nayyeri, C. Xu, J. Lehmann, and H. S. Yazdi, "LogicENN: A Neural Based Knowledge Graphs Embedding Model with Logical Rules," *arXiv preprint arXiv:1908.07141*, 2019.

[102] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2D Knowledge Graph Embeddings," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11573.

[103] B. Ding, Q. Wang, B. Wang, and L. Guo, "Improving Knowledge Graph Embedding Using Simple Constraints," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 110–121. DOI: 10.18653/v1/P18-1011.

[104] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16, Savannah, USA: USENIX Association, 2016, pp. 265–283, ISBN: 9781931971331.

[105] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang, "RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HkgEQnRqYQ.

[106] T. Lacroix, N. Usunier, and G. Obozinski, "Canonical Tensor Decomposition for Knowledge Base Completion," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, Stockholmsmässan, Stockholm Sweden: Proceedings of Machine Learning Research, Jul. 2018, pp. 2863–2872. [Online]. Available: http://proceedings.mlr.press/v80/lacroix18a.html.

[107] I. Chami, A. Wolf, D.-C. Juan, F. Sala, S. Ravi, and C. Ré, "Low-Dimensional Hyperbolic Knowledge Graph Embeddings," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 6901–6914. DOI: 10.18653/v1/2020.acl-main.617.

[108] R. Haeb-Umbach and H. Ney, "Improvements in beam search for 10000-word continuous-speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 2, no. 2, pp. 353–356, Apr. 1994, ISSN: 1558-2353. DOI: 10.1109/89.279287.

[109] F. Radulovic, N. Mihindukulasooriya, R. García-Castro, and A. Gómez-Pérez, "A comprehensive quality model for Linked Data," *Semantic Web*, vol. 9, no. 1, pp. 3–24, 2018. DOI: 10.3233/SW-170267.

[110] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," *Lecture notes of EE392o, Stanford University, Autumn Quarter*, vol. 2004, pp. 2004–2005, 2003. [Online]. Available: https://web.stanford.edu/class/ee392o/subgrad_method.pdf.

[111] J. Larson, M. Menickelly, and S. M. Wild, "Derivative-free optimization methods," *Acta Numerica*, vol. 28, pp. 287–404, 2019. DOI: 10.1017/S0962492919000060.

[112] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$," in *Dokl. akad. nauk Sssr*, vol. 269, 1983, pp. 543–547. [Online]. Available: http://mi.mathnet.ru/eng/dan46009.

[113] P. Guo and M. R. Lyu, "A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data," *Neurocomputing*, vol. 56, pp. 101–121, 2004, ISSN: 0925-2312. DOI: https://doi.org/10.1016/S0925-2312(03)00385-0.

[114] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012. [Online]. Available: http://jmlr.org/papers/v13/bergstra12a.html.

Appendix A: Notes on Gradient Descent and Backpropagation

A.1 Gradient Descent

Gradient Descent (GD) is an iterative optimization algorithm that uses the gradient of an error function, specifically its direction, to tune parameter values in the direction of the local minima of the error function. It is based on the intuition that if a multi-variable error/cost function $c(w_0)$ is differentiable [1] in a neighborhood of a point w_0, then $c(w)$ decreases fastest if one goes from w_0 in the direction of the negative gradient of c at w_0, that is, $-\nabla_w c(w_0)$. It involves approximating the function using the Taylor series, given by:

$$c(w) = \Sigma_{n=0}^{\infty} \frac{c^{(n)}(w_0)}{n!}(w - w_0)^n \tag{A.1}$$

where $c^{(n)}(w_0)$ is the n^{th} derivative of the function $c(w)$ evaluated at point w_0. Newton's method expands this Taylor series with $n = 2$ to the whole cost using only the first three terms.

$$\begin{aligned} c(w) &= \Sigma_{n=0}^{\infty} \frac{c^{(n)}(w_0)}{n!}(w - w_0)^n \\ &\approx c(w_0) + (w - w_0)c'(w_0) + (w - w_0)^2 \frac{c''(w_0)}{2} \end{aligned} \tag{A.2}$$

A stationary point of the approximated error function can be found by setting the derivative, with respect to the weights w, of the function to zero and solving for the

[1] This constraint has been mitigated through Sub-gradient methods [110] and derivative-free methods [111] to optimizing a non-convex function

required parameters.

$$\begin{aligned} c'(w) &\approx \frac{\partial}{\partial w} c(w_0) + \frac{\partial}{\partial w}((w - w_0)c'(w_0)) + \frac{\partial}{\partial w}((w - w_0)^2 \frac{c''(w_0)}{2}) \\ &\approx c'(w_0) + (w - w_0)c''(w_0) = 0 \end{aligned} \tag{A.3}$$

Upon solving for the parameter w, we arrive at the following expression.

$$w = w_0 - \frac{c'(w_0)}{c''(w_0)} \tag{A.4}$$

Since this parameter update is based off an approximation of the error function, it is iteratively updated based on the current parameter values. This requires the recomputation of both the first derivative and the second derivative of the function. Compared to computing the first derivative, computing the second derivative is quite expensive. Instead, practitioners approximate the second-order derivative using a constant value called step size or learning rate [53]. The first-order GD update rule is given by substituting a learning rate in place of second derivative information $\mu \approx c''(w_0)$. This simplifies equation A.4 to:

$$w = w_0 - \mu c'(w_0) \tag{A.5}$$

The learning rate μ is a hyperparameter that is selected through experimentation. Learning rate, one of many hyperparameters, including, momentum [54], nestorov [112], batch size, epochs, etc., need to be tuned for successful GD-based training [113]. These hyperparameter values vary widely across datasets, requiring the practitioner to perform hyperparameter tuning [114], which trains multiple models at the expense of compute power.

A.1.1 Modifications made to the GD update rule

Once the gradient of each parameter in the compute graph is calculated, either directly or through backpropagation, the weights are updated using the GD update rule.

There are several variants and advancements of GD that has been applied to optimize a chain of differentiable operations.

Batch Gradient Descent (BGD) involves updating the weights of the network after accumulating the gradients calculated for every sample in the dataset. Although this procedure is guaranteed to reach the local minima of a smooth convex error function, it has a slower convergence rate since the gradients have to be computed for each sample in the dataset before updating the network. Stochastic Gradient Descent (SGD) uses a single sample to approximate the gradient. This results in faster convergence, but is certainly not the best approach to solving the problem. Since SGD approximates the gradient using a single example, a noisier gradient is calculated, which can bump the weights out of a local minima. Mini-batch Gradient Descent [57] combines both the ideas by using a subset of the dataset to calculate the gradient.

Several modifications have been proposed to the update rule, in order to speed up or stabilize training. Momentum [54] accelerates GD to move out of a local optimum for the search of better local optima. This often leads convergence to a saddle point. Nesterov [55] gives momentum a prior knowledge about the curvature of the error function. This decelerates momentum, preventing it from moving the weights out of a local optimum. Adagrad [56] adjusts the learning rate for each parameter based on their rate of change. Parameters that did not change too often had a higher step size that those parameters than changed frequently. This led to increased robustness compared to SGD. RMSProp [57] and Adadelta [58] were developed to overcome Adagrad's radically diminishing learning rates. Adam [59] combines RMSProp and Momentum by storing an exponentially decaying average of past squared gradients, like RMSProp, and an exponentially decaying average of past gradients, like momentum. AMSGrad [60] fixes convergence issues with Adam by incorporating a "long-term memory" of past gradients. Currently, the Adam update rule produces the fastest convergence rates compared to other update rules described

in this subsection. For this reason, we optimize the parameters of HOPLoP using the Adam update rule.

A.2 Derivative of common activation functions

A.2.1 Sigmoid

$$\begin{aligned}
\sigma(z) &= \frac{1}{1+e^{-z}} \\
\frac{\partial}{\partial z}\sigma(z) &= \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2} \\
&= \left(\frac{1}{1+e^{-z}}\right) - \left(\frac{1}{1+e^{-z}}\right)^2 \\
&= \sigma(z)(1-\sigma(z))
\end{aligned} \tag{A.6}$$

A.2.2 Tanh

$$\begin{aligned}
tanh(z) &= \frac{sinh(z)}{cosh(z)} \\
\frac{\partial}{\partial z}tanh(z) &= \frac{\partial}{\partial z}\left(\frac{sinh(z)}{cosh(z)}\right) \\
&= \frac{cosh(z)\frac{\partial(sinh(z))}{\partial z} - sinh(z)\frac{\partial(cosh(z))}{\partial z}}{cosh^2(z)} \\
&= \frac{cosh^2(z) - sinh^2(z)}{cosh^2(z)} \\
&= 1 - tanh^2(z)
\end{aligned} \tag{A.7}$$

A.2.3 ReLU

$$\begin{aligned}
ReLU(z) &= max(0,z) \\
\frac{\partial}{\partial z}ReLU(z) &= \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}
\end{aligned} \tag{A.8}$$

A.3 Derivative of mean squared error function for linear regression

Let $MSE(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ be the mean squared error function. Let $\hat{y} = LR(x)$ be the predicted output value. Then, the derivative of the MSE function, with respect to the parameters of the LR function is:

$$\begin{aligned}
\frac{\partial}{\partial w_j}(MSE(LR(X), Y)) &= \frac{\partial}{\partial w_j}\left(\frac{1}{N}\left(\sum_{i=1}^{N}\sum_{j=0}^{d}(x_{ij}w_j - y_i)^2\right)\right) \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial}{\partial w_j}\sum_{j=0}^{d}(x_{ij}w_j - y_i)^2\right) \\
&= \frac{2}{N}\sum_{i=1}^{N}\left((x_{ij}w_j - y_i)\frac{\partial}{\partial w_j}(x_{ij}w_j)\right) \\
&= \frac{2}{N}\sum_{i=1}^{N}(x_{ij}w_j - y_i)x_{ij} \\
&= \frac{2}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)x_{ij}
\end{aligned} \tag{A.9}$$

A.4 Derivative of binary cross-entropy function for logistic regression

Let $BCE(\hat{y}, y) = -\frac{1}{N}\sum_{i=1}^{N}(y_i log(\hat{y}_i) + (1 - y_i)log(1 - \hat{y}_i))$ be the binary cross-entropy loss function. Let $\hat{y} = LogR(x)$ be the value predicted by the logistic regression function. The derivative of the loss function with respect to the parameters of the logistic regression function:

$$
\begin{aligned}
&\frac{\partial}{\partial w_j} BCE(LogR(X), Y) \\
&= \frac{\partial}{\partial w_j}\left(\frac{-1}{N}\sum_{i=1}^{N}\left(y_i log\left(\sigma\left(\sum_{j=0}^{d} x_{ij}w_i\right)\right) + (1-y_i)\, log\left(1-\sigma\left(\sum_{j=0}^{d} x_{ij}w_i\right)\right)\right)\right) \\
&= \frac{-1}{N}\sum_{i=1}^{N}\left(y_i \frac{\partial}{\partial w_j}\left(log\left(\sigma\left(\sum_{i=0}^{d} x_{ij}w_i\right)\right)\right) + (1-y_i)\frac{\partial}{\partial w_j}\left(log\left(1-\sigma\left(\sum_{i=0}^{d} x_{ij}w_i\right)\right)\right)\right) \\
&= \frac{-1}{N}\sum_{i=1}^{N}\left(\frac{y_i}{\hat{y}_i}\frac{\partial}{\partial w_j}\left(\sigma\left(\sum_{j=0}^{d} x_{ij}w_i\right)\right) + \frac{(1-y_i)}{(1-\hat{y}_i)}\frac{\partial}{\partial w_j}\left(1-\sigma\left(\sum_{j=0}^{d} x_{ij}w_j\right)\right)\right) \\
&= \frac{-1}{N}\sum_{i=1}^{N}\left(\frac{y_i}{\hat{y}_i}\frac{\partial}{\partial w_j}\left(\sigma\left(\sum_{j=0}^{d} x_{ij}w_i\right)\right) - \frac{(1-y_i)}{(1-\hat{y}_i)}\frac{\partial}{\partial w_j}\left(\sigma\left(\sum_{j=0}^{d} x_{ij}w_j\right)-1\right)\right) \\
&= \frac{-1}{N}\sum_{i=1}^{N}\left(\frac{y_i}{\hat{y}_i}\frac{\partial}{\partial w_j}\left(\sigma\left(\sum_{j=0}^{d} x_{ij}w_i\right)\right) - \frac{(1-y_i)}{(1-\hat{y}_i)}\frac{\partial}{\partial w_j}\left(\sigma\left(\sum_{j=0}^{d} x_{ij}w_j\right)\right)\right) \\
&= \frac{-1}{N}\sum_{i=1}^{N}\left(\left(\frac{y_i}{\hat{y}_i} - \frac{(1-y_i)}{(1-\hat{y}_i)}\right)\frac{\partial}{\partial w_j}\left(\sigma\left(\sum_{j=0}^{d} x_{ij}w_i\right)\right)\right) \\
&= \frac{-1}{N}\sum_{i=1}^{N}\left(\left(\frac{y_i - y_i\hat{y}_i - \hat{y}_i + y_i\hat{y}_i}{\hat{y}_i(1-\hat{y}_i)}\right)\frac{\partial}{\partial w_j}\left(\sigma\left(\sum_{j=0}^{d} x_{ij}w_i\right)\right)\right) \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - y_i\right)x_{ij}
\end{aligned}
\tag{A.10}
$$

A.5 Normalization eliminates the need for bias

Let $MSE(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ be the mean squared error function. Let $\hat{y} = LR(x) = \sum_{j=0}^{d} x_{ij}w_j$ be the predicted output value. Then, the derivative for this loss, with respect to the bias w_0 is:

$$\begin{aligned}\frac{\partial}{\partial w_0}(MSE(LR(X), Y)) &= \frac{\partial}{\partial w_0}\left(\frac{1}{N}\left(\sum_{i=1}^{N}\sum_{j=0}^{d}(x_{ij}w_j - y_i)^2\right)\right) \\ &= \frac{2}{N}\sum_{i=1}^{N}\sum_{j=0}^{d}(x_{ij}w_j - y_i)x_{i0} \\ &= \frac{2}{N}\sum_{i=1}^{N}\sum_{j=0}^{d}(x_{ij}w_j - y_i)\end{aligned} \tag{A.11}$$

Equating the derivative of the error function to 0:

$$\begin{aligned}0 &= \sum_{i=1}^{N}\sum_{j=0}^{d} x_{ij}w_j - y_i \\ &= \sum_{i=1}^{N}\left(w_0 + \sum_{j=1}^{d} x_{ij}w_j - y_i\right)\end{aligned} \tag{A.12}$$

Upon simplifying, we get:

$$\sum_{i=1}^{N} w_0 = \sum_{i=1}^{N} y_i - \sum_{j=1}^{d} w_j \sum_{i=1}^{N} x_{ij} \tag{A.13}$$

When all features columns of X are normalized to have zero mean, i.e. when $\sum_{i=1}^{N} x_{ij}$ for any column j:

$$\sum_{i=1}^{N} w_0 = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{A.14}$$

If the target feature is also normalized, then $\sum_{i=1}^{N} y_i = 0$, which would make the eliminate the bias w_0 term.

A.6 Backpropagation

We describe the backpropagation algorithm with respect to the linear regression setting. Similar to logistic regression, the backpropagation algorithm for the classification setting can be derived. Let $x_i \in X$ be an input vector with elements $x_{ij} \in x_i$ representing the input features. Let $y_i \in Y$ be the corresponding output for input x_i. For the linear regression setting, let us continue minimizing the mean squared error loss, as given by:

$$\begin{aligned} E(NN(X), Y) &= \frac{1}{N}(NN(X) - Y)^2 \\ &= \frac{1}{N}(f_L(...(f_1(X \times W_1))...W_L) - Y)^2 \end{aligned} \tag{A.15}$$

Similar to the previous examples, we proceed by calculating the gradients for $E(NN(X), Y)$ with respect to W_L:

$$\begin{aligned} \frac{\partial}{\partial W_L} E(NN(X), Y) &= \frac{\partial}{\partial W_L} \frac{1}{N}(NN(X) - Y)^2 \\ &= \frac{2}{N}(NN(X) - Y)\frac{\partial}{\partial W_L}(NN(X) - Y) \\ &= \frac{2}{N}(NN(X) - Y)\frac{\partial}{\partial W_L} NN(X) \\ &= \frac{2}{N}(NN(X) - Y)\frac{\partial}{\partial W_L} f_L(h_{L-1}W_L) \\ \text{Let} \quad O_L &= h_{L-1}W_L \qquad \text{and use chain rule}^2 \\ \frac{\partial}{\partial W_L} E(NN(X), Y) &= \frac{2}{N}(NN(X) - Y)\frac{\partial}{\partial W_L} f_L(O_L) \\ \frac{\partial}{\partial W_L} E(NN(X), Y) &= \frac{2}{N}(NN(X) - Y)\frac{\partial}{\partial O_L} f_L(O_L)\frac{\partial}{\partial W_L} O_L \\ &= \frac{2}{N}(NN(X) - Y)\frac{\partial}{\partial O_L} f_L(O_L) h_{L-1} \\ \text{Let} \quad \delta_L &= \frac{2}{N}(NN(X) - Y)\frac{\partial}{\partial O_L} f_L(O_L) \\ \frac{\partial}{\partial W_L} E(NN(X), Y) &= \delta_L \times h_{L-1} \end{aligned} \tag{A.16}$$

Then, the GD weight update rule for W_L is:

$$\begin{aligned} W_L^{(t+1)} &= W_L^{(t)} - \mu \frac{\partial}{\partial W_L} E(NN(X), Y) \\ &= W_L^{(t)} - \mu(\delta_L \times h_{L-1}) \end{aligned} \tag{A.17}$$

Similarly, we find the gradients for $E(NN(X), Y)$ with respect to W_{L-1}:

$$\begin{aligned} \frac{\partial}{\partial W_{L-1}} E(NN(X), Y) &= \frac{\partial}{\partial W_{L-1}} \frac{1}{N} (NN(X) - Y)^2 \\ &= \frac{2}{N} (NN(X) - Y) \frac{\partial}{\partial W_L} f_L(f_{L-1}(h_{L-2} W_{L-1}) W_L) \\ \text{Let} \qquad O_L &= f_{L-1}(h_{L-2} W_{L-1}) W_L \qquad \text{and use chain rule} \\ \frac{\partial}{\partial W_{L-1}} E(NN(X), Y) &= \frac{2}{N} (NN(X) - Y) \frac{\partial}{\partial O_L} f_L(O_L) \frac{\partial}{\partial W_{L-1}} O_L \\ \text{Notice} \qquad \delta_L &= \frac{2}{N} (NN(X) - Y) \frac{\partial}{\partial O_L} f_L(O_L) \\ \frac{\partial}{\partial W_{L-1}} E(NN(X), Y) &= \delta_L \frac{\partial}{\partial W_{L-1}} f_{L-1}(h_{L-2} W_{L-1}) W_L \\ &= \frac{2}{N} (NN(X) - Y) \frac{\partial}{\partial O_L} f_L(O_L) \frac{\partial}{\partial W_{L-1}} f_{L-1}(h_{L-2} W_{L-1}) W_L \\ &= \delta_L W_L^T \frac{\partial}{\partial W_{L-1}} f_{L-1}(h_{L-2} W_{L-1}) \\ &= \delta_L W_L^T \frac{\partial}{\partial O_{L-1}} f_{L-1}(O_{L-1}) \frac{\partial}{\partial W_{L-1}} O_{L-1} \\ &= \delta_L W_L^T \frac{\partial}{\partial O_{L-1}} f_{L-1}(O_{L-1}) h_{L-2} \\ \text{Let} \qquad \delta_{L-1} &= \delta_L W_L^T \frac{\partial}{\partial O_{L-1}} f_{L-1}(O_{L-1}) \\ \frac{\partial}{\partial W_{L-1}} E(NN(X), Y) &= \delta_{L-1} \times h_{L-2} \end{aligned} \tag{A.18}$$

Then, the GD weight update rule for W_{L-1} is:

$$\begin{aligned} W_{L-1}^{(t+1)} &= W_{L-1}^{(t)} - \mu \frac{\partial}{\partial W_{L-1}} E(NN(X), Y) \\ &= W_{L-1}^{(t)} - \mu(\delta_{L-1} \times h_{L-2}) \end{aligned} \tag{A.19}$$

Upon noticing the pattern for weight updates, we can arrive at the following weight update rules:

$$W_l^{(t+1)} = W_l^{(t)} - \mu(\delta_l \times h_{l-1}) \tag{A.20}$$

where $\delta_l = \delta_{l+1} W_{l+1}^T \frac{\partial}{\partial O_l} f_l(O_l)$, $\delta_L = \frac{2}{N}(NN(X) - Y)\frac{\partial}{\partial O_L} f_L(O_L)$ and $h_0 = X$.

Appendix B: Supplementary Information

In this chapter, information is provided for the reproducibility of the experiments conducted in the paper. The datasets and scripts that were used to perform the experiments are available in the https://www.github.com/varunranga/HOPLoP

B.1 Dependencies

B.1.1 Hardware dependencies

A single system was used to train all HOPLoP models. The specifications of that system is as follows:

- CPU : Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz
- GPU : NVIDIA GeForce 1080Ti $\times$ 2
- RAM : 64GB
- Hard disk space : 3.5TB

A single system was used to train all M-HOPLoP models. The specifications of that system is as follows:

- CPU : Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz
- GPU : NVIDIA GeForce 1080Ti + NVIDIA Titan X (Pascal)
- RAM : 64GB

- Hard disk space : 3.5TB

B.1.2 Software dependencies

Table B.1 provides a list of software dependencies that are required to run the required codes. To run the codes, please make sure the packages and softwares are installed on the system.

Package / Software	**Version**
Ubuntu	18.04.3
Docker	19.03.5
Docker container	tensorflow/tensorflow:latest-gpu-py3
Python3	3.6.8
Cuda	10.0.130
Tensorflow-gpu	2.0.0
Numpy	1.17.2
Tqdm	4.48.2
Random	(Available with Python3)
Argparse	(Available with Python3)
Pickle	(Available with Python3)
Math	(Available with Python3)

Table B.1: Software dependencies for running provided codes.

B.2 Datasets

Due to the large size of datasets, we do not provide previously introduced datasets. The NELL-995 and FB15K-237 datasets are publicly available at `https://github.com/xwhan/DeepPath`. Upon extracting the downloaded files, copy the contents of the resultant folder to `(M-)HOPLoP/Datasets/NELL995/` and `(M-)HOPLoP/Datasets/FB15K237/` respectively. Since we introduce two new datasets for this task of multi-hop LP, we include those datasets in

(M-)HOPLoP/Datasets/WN18RR and (M-)HOPLoP/Datasets/YAGO310 directories. Alternatively, you can reproduce the dataset by using the script (M-)HOPLoP/Datasets/[WN18RR|YAGO310]/create_dataset.py. Please run the command PYTHONHASHSEED=0 python3 create_dataset.py in the required directory.

B.3 Executing code

The repository contains all the codes required for experiments. (M-)HOPLoP/Logs contains the log files for all experiments. Due to the large volume of experiments run, we do not provide the trained model files. However, we have provided bash scripts (M-)HOPLoP/run.sh that runs all the experiments we ran with the seed for each experiment. We have also provided a spreadsheet (M-)HOPLoP/Results.xlsx that provides the results of all experiments. We include the bootstrapped confidence intervals for all experiments.

HOPLoP requires an embedding space to traverse over. Therefore, the first script that must be executed is HOPLoP/create_embeddings.py. Table B.2 gives a list of arguments and describes their purpose. Additionally, -h or --help arguments can be invoked. Please keep in mind to save the embedding model information before executing the HOPLoP script. To execute the main HOPLoP script (M-)HOPLoP/main.py, use the python3 command to invoke the Python3 interpreter, and send the argument main.py to execute statements from that script while in the (M-)HOPLoP/ directory. Additional command line arguments may be provided to set hyperparameters and embedding spaces used by HOPLoP. Table B.3 gives a list of short and long arguments. Please make sure to install all dependencies and place external datasets in required directories before executing code.

Flag	Long Argument	Description	Type	Default	Values / Comments
-d	–dataset	Dataset to be used	str	'NELL995'	['NELL995', 'FB15K237', 'WN18RR', 'YAGO310']
-x	–embedding-method	Embedding Model to be used	str	'TransE	['TransE', 'ComplEx', 'TuckER']
-e	–embedding-size	Embedding dimensionality of each vector	int	100	
-m	–margin	Margin of error allowed in the loss	float	1.0	For TransE embedding model
-r	–learning-rate	Learning rate for the optimizer	float	0.001	
-b	–batch-size	Batch size while training	int	1024	
-g	–sampling-type	Method used to negatively sample data	str	'bernoulli'	['uniform', 'bernoulli'], for TransE and ComplEx
-p	–patience	Patience while training the embedding model for validation loss to improve	int	100	
-s	–save	Pickle file name for the trained embeddings to save	str	None	
-sd	–seed	Initial seed value	int	None	

Table B.2: Flags and long arguments that can be used to run embedding generation code ('create-embeddings.py').

Flag	Long Argument	Description	Type	Default	Values / Comments
-d	–dataset	Dataset to be used	str	'NELL995'	['NELL995', 'FB15K237', 'WN18RR', 'YAGO310']
-t	–task	Task / Query relation to train and evaluate HOPLoP	str	'concept_athleteplaysforteam'	see file 'Dataset.py' for options
-e	–load-embedding	Pickle file of the embeddings to be used	str	'NELL995_Embeddings.bin'	
-x	–save-result	Pickle file to save the evaluation result	str	None	
-p	–patience	Patience while training the embedding model for validation loss to improve	int	100	
-r	–learning-rate	Learning rate for the optimizer	float	0.001	
-s	–save	Pickle file name for the trained HOPLoP model to save	str	None	
-l	–load	Pickle file name for the trained HOPLoP model to load	str	None	
-o	–hops	Number of hops HOPLoP takes to perform a traversal	int	10	[1, 3, 5, 10, 15, 20]
-c	–batch-size	Batch size of training HOPLoP model	int	8	
-n	–network	Path finder network architecture	list	['1000', 'relu']	
-sd	–seed	Initial seed value	int	None	

Table B.3: Flags and long arguments that can be used to run HOPLoP code ('main.py').

B.4 Hyperparameters

Table B.6 and B.4 presents the list of hyperparameters that can be adjusted for embedding space creation and HOPLoP training. The value of certain hyperparameters remained constant throughout all experiments for a fair comparison of all models. For HOPLoP and M-HOPLoP, we perform major hyperparameter tuning on the number of hop H. Table B.7 present the optimal H values for each task in each dataset in HOPLoP experiments. We also provide (see table B.5) the optimal H for M-HOPLoP over each dataset.

Hyperparameter	**Value(s)**
Learning Rate	0.001
Batch Size	8
Hops	{1, 3, 5, 10, 15, 20}
Patience	100
Runs per experiment	10

Table B.4: Hyperparameters used for HOPLoP

Dataset	**Embedding**	**Hop**	**MAP Score**
NELL-995	TransE	3.0	0.946
FB15K-237	TransE	5.0	0.914
	ComplEx	10.0	0.826
	TuckER	10.0	0.722
WN18RR	TransE	5.0	0.847
	ComplEx	5.0	0.720
	TuckER	10.0	0.549
YAGO3-10	TransE	5.0	0.842
	ComplEx	3.0	0.818

Table B.5: Optimal H values from M-HOPLoP experiments.

Hyperparameter	Value(s)	Comments
Learning Rate	0.001	
Batch Size	1024	
Embedding Dimension	100	For ComplEx - 50 for real and imaginary components each
Patience	100	
Initialization	Glorot Uniform	
Negative Sampling type	Bernoulli	Only TransE and ComplEx
Margin	1.0	Only TransE
Max epochs	100	Only TuckER, due to computational restrictions
Input Dropout Rate	0.3	Only TuckER, from their paper
Hidden Layer 1 Dropout Rate	0.4	Only TuckER, from their paper
Hidden Layer 2 Dropout Rate	0.5	Only TuckER, from their paper
Number of runs per experiment	10	

Table B.6: Hyperparameters used for embedding generation.

Table B.7: Optimal H values from HOPLoP experiments.

Dataset	Task	Embedding	Hop	MAP Score
NELL-995	AthleteHomeStadium	TransE	10	0.930
	TeamPlaysSport		20	0.980
	AthletePlaysForTeam		5	0.953
	PersonBornInLocation		20	0.961
	AthletePlaysInLeague		5	0.997
	AgentBelongsToOrg		10	0.947
	OrgHiredPerson		5	0.930
	TeamPlaysInLeague		1	0.977
	PersonLeadsOrg		10	0.962
	OrgHeadquarteredInCity		10	0.956
	AthletePlaysSport		1	0.930
	WorksFor		5	0.993
FB15K-237	Director/Film	TransE	10	0.654
		ComplEx	15	0.679
		TuckER	15	0.671
	Event/Locations	TransE	1	0.735
		ComplEx	1	0.776
		TuckER	5	0.708
	Person/BirthPlace	TransE	15	0.980
		ComplEx	15	0.960
		TuckER	5	0.958
	OrgHeadquarters/Location	TransE	20	0.968
		ComplEx	20	0.975
		TuckER	15	0.933
	TvProgram/Genre	TransE	10	0.977
		ComplEx	5	0.919
		TuckER	5	0.889

CapitalOf/Location	TransE	10	0.906
	ComplEx	15	0.905
	TuckER	10	0.838
Film/Music	TransE	10	0.993
	ComplEx	10	0.981
	TuckER	15	0.977
PhoneNumber/ServiceLocation	TransE	5	0.768
	ComplEx	15	0.830
	TuckER	5	0.822
TvProgram/CountryOfOrigin	TransE	15	0.965
	ComplEx	5	0.975
	TuckER	5	0.965
Person/Nationality	TransE	15	0.958
	ComplEx	10	0.977
	TuckER	5	0.981
Ethnicity/LanguagesSpoken	TransE	10	0.601
	ComplEx	10	0.648
	TuckER	10	0.688
Profession/SpecializationOf	TransE	5	0.858
	ComplEx	5	0.821
	TuckER	5	0.959
Artist/Origin	TransE	10	0.966
	ComplEx	15	0.903
	TuckER	15	0.866
TvProgram/Languages	TransE	5	0.984
	ComplEx	15	0.987
	TuckER	3	0.979
Film/Language	TransE	20	0.971
	ComplEx	10	0.971
	TuckER	20	0.918

	OrgFounder/OrgsFounded	TransE	5	0.812
		ComplEx	5	0.864
		TuckER	3	0.757
	OrgMember/Org	TransE	15	0.871
		ComplEx	15	0.921
		TuckER	15	0.859
	Film/Country	TransE	15	0.954
		ComplEx	10	0.958
		TuckER	20	0.942
	Film/WrittenBy	TransE	10	0.994
		ComplEx	5	0.978
		TuckER	20	0.972
	SportsTeam/Sport	TransE	1	0.989
		ComplEx	1	0.993
		TuckER	1	0.995
WN18RR	DerivationallyRelatedForm	TransE	5	0.994
		ComplEx	3	0.977
		TuckER	15	0.772
	HasPart	TransE	5	0.768
		ComplEx	15	0.724
		TuckER	15	0.858
	InstanceHypernym	TransE	5	0.966
		ComplEx	20	0.932
		TuckER	20	0.977
	Hypernym	TransE	3	0.968
		ComplEx	10	0.860
		TuckER	10	0.760
	AlsoSee	TransE	10	0.923
		ComplEx	20	0.772

		TuckER	15	0.841
	MemberDomainRegion	TransE	5	0.669
		ComplEx	10	0.652
		TuckER	10	0.696
	SynsetDomainTopicOf	TransE	3	0.978
		ComplEx	20	0.925
		TuckER	10	0.969
	VerbGroup	TransE	5	0.984
		ComplEx	3	0.983
		TuckER	10	0.796
	MemberDomainUsage	TransE	10	0.571
		ComplEx	20	0.514
		TuckER	5	0.482
	MemberMeronym	TransE	5	0.706
		ComplEx	15	0.598
		TuckER	10	0.800
YAGO3-10	actedIn	TransE	5	0.861
		ComplEx	15	0.840
	diedIn	TransE	10	0.842
		ComplEx	10	0.776
	directed	TransE	5	0.942
		ComplEx	5	0.891
	hasWonPrize	TransE	5	0.840
		ComplEx	5	0.878
	edited	TransE	5	0.970
		ComplEx	5	0.914
	participatedIn	TransE	10	0.790
		ComplEx	5	0.770
	isMarriedTo	TransE	10	0.941
		ComplEx	1	0.967

playsFor	TransE	15	0.999
	ComplEx	15	0.999
isAffiliatedTo	TransE	5	0.941
	ComplEx	10	0.951
wasBornIn	TransE	5	0.883
	ComplEx	5	0.766
happenedIn	TransE	10	0.943
	ComplEx	3	0.877
isCitizenOf	TransE	3	0.936
	ComplEx	5	0.958
isLocatedIn	TransE	15	0.995
	ComplEx	20	0.995
isConnectedTo	TransE	5	0.979
	ComplEx	5	0.948
hasGender	TransE	1	0.973
	ComplEx	1	0.971
hasMusicalRole	TransE	5	0.923
	ComplEx	15	0.935
hasChild	TransE	5	0.958
	ComplEx	10	0.841
worksAt	TransE	15	0.947
	ComplEx	20	0.967
livesIn	TransE	5	0.945
	ComplEx	10	0.931
graduatedFrom	TransE	20	0.931
	ComplEx	20	0.892
wroteMusicFor	TransE	5	0.896
	ComplEx	15	0.940
influences	TransE	10	0.869

	ComplEx	10	0.714
created	TransE	20	0.945
	ComplEx	20	0.952