Representation Learning with Explicit and Implicit Graph Structures

by

Bahare Fatemi

B.Sc., Amirkabir University of Technology, 2015
M.Sc., The University of British Columbia, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies
(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)
April 2023
© Bahare Fatemi 2023
The following individuals certify that they have read, and recommend to the
Faculty of Graduate and Postdoctoral Studies for acceptance, the disserta-
tion entitled: Representation learning with explicit and implicit graph structures
submitted by Bahare Fatemi in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in Computer Science.

**Examining Committee:**

David Poole, Professor, Computer Science, UBC
Supervisor

Leonid Sigal, Professor, Computer Science, UBC
Supervisory Committee Member

Jeff Clune, Associate Professor, Computer Science, UBC
University Examiner

Daniel J. McDonald, Associate Professor, Statistics, UBC
University Examiner

**Additional Supervisory Committee Members:**

Frank Wood, Associate Professor, Computer Science, UBC
Supervisory Committee Member

William L. Hamilton, Adjunct Professor, Computer Science, McGill
Supervisory Committee Member
Abstract

The world around us is composed of objects each having relations with other objects. The objects and relations form a (hyper)graph with objects as nodes and relations between objects as (hyper)edges. When learning, the underlying structure representing the relations between the nodes is either given explicitly in the training set or is implicit and needs to be inferred. This dissertation studies graph representation learning with both explicit and implicit structures. For explicit structure, we first tackle the challenge of enforcing taxonomic information while embedding entities and relations. We prove that some fully expressive models cannot respect subclass and sub-property information. With minimal modifications to an existing knowledge graph completion method, we enable the injection of taxonomic information. A second challenge is in representing explicit structures in relational hypergraphs that contain relations defined on an arbitrary number of entities. While techniques, such as reification, exist that convert non-binary relations into binary ones, we show that current embedding-based methods do not work well out of the box for knowledge graphs obtained through these techniques. We introduce embedding-based methods that work directly with relations of arbitrary arity. We also develop public datasets, benchmarks, and baselines and show experimentally that the proposed models are more effective than the baselines. We further bridge the gap between relational algebra and knowledge hypergraphs by proposing an embedding-based model that can represent relational algebra operations. Having introduced novel architectures for explicitly graph-structured data, we further investigate how models with relational inductive biases can be developed and applied to problems with implicit structures. Graph representation learning models work well when the structure is explicit. However, this structure may not always be available in real-world applications. We propose the Simultaneous Learning of Adjacency and graph neural network Parameters with Self-supervision, or SLAPS, a method that provides more supervision for inferring a graph structure through self-supervision. An experimental study demonstrates that SLAPS scales to large graphs with hundreds of thousands of nodes and outperforms several baselines on established benchmarks.
Lay Summary

On a day-to-day basis, we make predictions about the world to help us make informed decisions. For instance, we predict whether we will like a movie before deciding to watch it or whether it will rain before taking an umbrella when going outside. This research develops computational techniques and representations that use available data about the world to make predictions about unknown events. In particular, this research makes predictions about the world when there is a notion of structure in the data: that is data points are related to each other with different types of relationships. For instance, predicting if each member of the family will like the movie given their relationships with each other and some of the movies they have liked before. This research can exploit the given or hidden relationships among the data points for making a better prediction.
Preface

In what follows, the contribution of all authors for each publication resulting from this research is explained in detail.

1. Bahare Fatemi, Siamak Ravanbakhsh, and David Poole. *Improved Knowledge Graph Embedding Using Background Taxonomic Information*, Proceedings of the AAAI Conference on Artificial Intelligence, 2019 ([39]).

The idea of enforcing taxonomic information into entity and relation embeddings started in our lab meetings. David Poole and I spent several weekly meetings discussing if existing models can enforce taxonomic information. We proved that most successful models for link prediction are not able to enforce taxonomic information. We spent more meetings on how to design a model that can provably enforce taxonomies. I designed and implemented the final model in our discussions and also conducted experiments to analyze the model. Siamak Ravanbakhsh helped us with his thorough feedback in the process, suggesting relevant work, and also proofreading and editing the paper before submission. This work is discussed in Chapter 3.

2. Bahare Fatemi, Perouz Taslakian, David Vazquez, and David Poole. *Knowledge Hypergraphs: Prediction Beyond Binary Relations*, Proceedings of the International Joint Conferences on Artificial Intelligence, 2020 ([40]).

After studying previous work on knowledge graph completion, David Poole and I started questioning whether current models can handle beyond binary (n-ary) relations. At the same time, I discussed the idea of designing models for n-ary relations with Perouz Taslakian and David Vazquez and they were both interested in collaborating on this project. The whole team met weekly and discussed the updates presented by me. I was mainly responsible for creating datasets, baselines, and new models for knowledge hypergraph completion. During the whole process, I received constructive feedback on proposed mod-
Preface

els and experimental and theoretical analyses from all the authors. Sections 4.1 and 4.2 are entirely based on this publication.


After finishing our previous work (Knowledge Hypergraphs: Prediction Beyond Binary Relations), the team continued working on designing completion models. The initial idea of proposing an embedding model based on relational algebra came from David Poole. I was mainly responsible for designing a model that can capture relational algebra operations, studying whether the current version of the model can capture such operations, and modifying the model to reflect our theoretical analysis. I also implemented and conducted experiments on the proposed model. The whole team met regularly to discuss the theoretical and experimental analyses. Section 4.3 discusses this publication.


This work was done during my internship at Borealis AI. When I joined Borealis AI for my internship, Mehran had a proposal on extending the applicability of graph neural networks to implicit graph structures. Initially, I did a literature review on this problem. Layla, Mehran, and I discussed related work and potential ideas in our weekly meeting. After careful studying of existing models, we identified the supervision starvation problem and came up with the idea of using self-supervision to improve upon existing models. I was responsible for designing and implementing SLAPS and also conducting thorough theoretical and experimental analyses. Chapter 5 is entirely based on this publication.
Table of Contents

Abstract .......................................................... ii
Lay Summary ...................................................... iii
Preface ............................................................. iv
Table of Contents ................................................ vi
List of Tables ..................................................... x
List of Figures ................................................... xiv
List of Software Packages ..................................... xix
Acknowledgements ............................................. xxi
Dedication ........................................................ xxii

1 Introduction ...................................................... 1
  1.1 Machine Learning and Structure ............................ 1
  1.2 Learning with Explicit Structure ......................... 2
  1.3 Learning with Implicit Structure ......................... 4

2 Background and Notation ..................................... 5
  2.1 Notation ................................................... 5
  2.2 Learning on Graphs ....................................... 6
    2.2.1 Types of Graphs .................................... 6
    2.2.2 Tasks on Graphs ................................... 9
  2.3 Knowledge Graphs and Hypergraphs ..................... 11
    2.3.1 Definition .......................................... 11
    2.3.2 Knowledge Graph Completion Methods ............. 12
  2.4 Graph Neural Networks .................................. 14
    2.4.1 Motivation and Intuition .......................... 14
3 Incorporating Ontological Information

3.1 Introduction

3.2 Related Work

3.3 Background and Notation

3.4 Tensor Factorization and Taxonomies

3.5 Proposed Variation: SimpE+

3.6 Experimental Results

3.6.1 Effect of Non-negativity Constraints on Entity Embeddings

3.6.2 Sparse Relations

3.6.3 Knowledge Graphs with no Redundant Triples

3.7 Limitations

3.8 Conclusion

4 Incorporating n-ary Relations

4.1 Introduction

4.2 Knowledge Hypergraph Completion: HSimpE and HypE

4.2.1 Related Work

4.2.2 Proposed Methods

4.2.3 Objective Function and Training

4.2.4 Datasets

4.2.5 Baselines

4.2.6 Experiments

4.2.7 Conclusions

4.3 Knowledge Hypergraph Completion with Relational Algebra: ReAlE

4.3.1 Related Work

4.3.2 ReAlE: A Basic Embedding Algorithm

4.3.3 Theoretical Analysis

4.3.4 Full expressivity

4.3.5 Representing Relational Algebra with ReAlE

4.3.6 Datasets

4.3.7 Experiments

4.3.8 Conclusion

4.4 Conclusion and Limitations
# Table of Contents

5 Implicit Graphs ........................................... 70
  5.1 Introduction ........................................... 70
  5.2 Related Work ........................................... 72
  5.3 Proposed Method: SLAPS ............................... 74
    5.3.1 Generator ........................................... 74
    5.3.2 Adjacency Processor ............................... 76
    5.3.3 Classifier ........................................... 77
    5.3.4 Using only the first three components leads to supervision starvation ........................................... 78
    5.3.5 Self-supervision ..................................... 79
    5.3.6 SLAPS .............................................. 81
  5.4 Experimental Setup ..................................... 82
    5.4.1 Baselines ........................................... 82
    5.4.2 Datasets ............................................ 82
  5.5 Experiments .............................................. 84
    5.5.1 Comparative Results ................................ 85
    5.5.2 The Effectiveness of Self-supervision ............. 87
    5.5.3 Experiments with Noisy Graphs .................... 90
    5.5.4 Analyses of the Learned Adjacency ............... 90
    5.5.5 Ablation Studies ................................... 92
  5.6 Limitations ............................................. 93
  5.7 Conclusion ............................................. 94

6 Conclusion and Future Work ................................. 95
  6.1 Future Work for Learning with Explicit Structures .... 95
  6.2 Future Work for Learning with Implicit Structures .... 98
  6.3 Combining Explicit and Implicit Graph Structures .... 99

Bibliography .............................................. 100

Appendices

A Proofs for Chapter 3 ...................................... 117

B Proofs and Implementation Details for Chapter 4 .......... 120
  B.1 Proofs for HypE and HSimplE ........................... 120
  B.2 Theoretical Analysis ................................... 123
    B.2.1 Full Expressivity of ReAIe ......................... 123
    B.2.2 Full Expressivity of Some Baselines ............... 126
Table of Contents

B.2.3 Representing Relational Algebra with ReAlE .......................... 127
B.2.4 Representing Relational Algebra with HypE ......................... 133
B.3 Implementation Details ......................................................... 137

C Supplementary Material for Chapter 5 ................................. 139
C.1 Supervision Starvation in Erdős-Rényi and Scale-free networks .................................................. 139
C.2 Implementation Details .......................................................... 141
C.2.1 kNN Implementation ............................................................ 142
C.3 More Experiments and Analysis ............................................ 142
C.3.1 Symmetrization ................................................................. 142
C.3.2 Fixing a prior graph manually instead of using self-supervision ................................................ 144
C.3.3 Why not compare the learned graph structures with the original ones? .............................................. 146

Index ......................................................................................... 148
List of Tables

2.1 Summary of notation with examples. ...................... 5

3.1 We conduct experiments on SPORT and LOCATION datasets. These datasets are subsets of NELL dataset \[100]\] and are introduced by Wang et al. \[146]\]. The subsumptions in SPORT and LOCATION datasets are listed in this table. .................. 25

3.2 Statistics of the datasets used in the experiments. In this table, \(|\mathcal{E}|\) represents the number of entities, \(|\mathcal{R}|\) represents the number of relations in the dataset, and \(#\text{train}, #\text{valid}, \#\text{test}\) denote number of triples in the train, validation, and test sets respectively. ...................... 25

3.3 Results of SimplE$^+$ with different non-linearity functions in producing non-negative embeddings on wn18 dataset without incorporating subsumptions. ReLU outperforms other choices, and therefore moving forward we use ReLU for non-negativity constraints in SimplE$^+$. ...................... 26

3.4 Results of SimplE$^+$ with different non-linearity functions in producing non-negative embeddings on FB15K dataset without incorporating subsumptions. ReLU outperforms other choices, and therefore moving forward we use ReLU for non-negativity constraints in SimplE$^+$. ...................... 26

3.5 Results on wn18 for SimplE and SimplE$^+$ without incorporating subsumptions. As the results indicate, the non-negativity constraint on entity embeddings deteriorates the model’s performance but allows us to be able to guarantee enforcing subsumptions (as shown in Theorem \[3.5]\). ......... 27

3.6 Results on FB15K for SimplE and SimplE$^+$ without incorporating subsumptions. As the results indicate, the non-negativity constraint on entity embeddings deteriorates the model’s performance but allows us to be able to guarantee enforcing subsumptions (as shown in Theorem \[3.5]\). ......... 27
List of Tables

3.7 Results comparing logical inference, SimplE, and SimplE+ on sport dataset. The best results are in bold. MRR and hit@k for $k > 1$ is not applicable for logical inference as it either predict the correct entity or not (SimplE and SimplE+ predict a ranking). The results demonstrate the effectiveness of SimplE+ in the presence of subsumptions. 

3.8 Results comparing logical inference, SimplE, and SimplE+ on location dataset. The best results are in bold. MRR and hit@k for $k > 1$ is not applicable for logical inference as it either predict the correct entity or not (SimplE and SimplE+ predict a ranking). The results demonstrate the effectiveness of SimplE+ in the presence of subsumptions.

4.1 Statistics of the datasets used in the experiments. In this table, $|\mathcal{E}|$ represents the number of entities, $|\mathcal{R}|$ represents the number of relations in the dataset, and #train, #valid, and #test denote number of triples in the train, validation, and test sets respectively.

4.2 Number of tuples with different arities in different datasets.

4.3 Knowledge hypergraph completion results on JF17K, FB-Auto and M-FB15K for baselines and the proposed method. The prefixes ‘r’ and ‘m’ in the model names stand for reification and multi-arity respectively. Both our methods outperform the baselines on all datasets. Results for JF17K dataset is an average of three runs.

4.4 Knowledge graph completion results on WN18 for baselines and HypE. Note that in knowledge graphs (binary relations), HSimplE and SimplE are equivalent, both theoretically and experimentally. The results show that our methods outperform the baselines.

4.5 Knowledge graph completion results on FB15K for baselines and HypE. Note that in knowledge graphs (binary relations), HSimplE and SimplE are equivalent, both theoretically and experimentally. The results show that our methods outperform the baselines.

4.6 Breakdown performance of hit@10 across relations with different arities on JF17K dataset along with their statistics.
List of Tables

4.7 Knowledge hypergraph completion results on JF17K, FB-AUTO and M-FB15K for baselines and the proposed method. Our method ReAIÉ outperforms (or is competitive to) the baselines on all datasets. The higher is better for all columns. To measure the stability of ReAIÉ, we ran it on JF17K 10 times with the best hyperparameters and the standard deviation of MRR is 0.0004. This shows that ReAIÉ is not sensitive to different runs (different random seeds). . . . . . . . . . . . . . 65

4.8 Breakdown performance of MRR across composite relations (based on a sequence of primitive operations) and of varying depths on the REL-ER dataset along with their statistics. The higher is better. . . . . . . . . . . . . . . . . . . . . . . . . . 66

4.9 Knowledge hypergraph completion results on JF17K, FB-AUTO and M-FB15K for baselines and the proposed method for a fixed embedding dimension of 200, negative ratio of 10, and batch size of 128. Our method ReAIÉ (Small) outperforms the baselines on all datasets with these settings. . . . . . . . . . . . . . 69

4.10 Knowledge hypergraph completion results for ReAIÉ on JF17K for different $\sigma$ (nonlinear function). . . . . . . . . . . . . . . 69

5.1 Statistics of the datasets used in the experiments. . . . . . . . 83

5.2 Results of SLAPS and the baselines on established node classification benchmarks presenting accuracy. † indicates results have been taken from Franceschi et al. [47]. ‡ indicates results have been taken from Stretcu et al. [134]. Bold and underlined values indicate the best and the second-best mean performances respectively. OOM indicates out of memory. OOT indicates out of time (we allowed 24h for each run). NA indicates not applicable. Appendix C.2 provides implementation details of SLAPS and baselines. . . . . . . . . . . . . . . . . . . . . . . . . . . 84

5.3 Results on classification datasets presenting accuracy. † indicates results have been taken from Franceschi et al. [47]. Bold and underlined values indicate best and the second-best mean performances respectively. . . . . . . . . . . . . . . . . . . . . . . . . . . 86

5.4 Classification accuracy results on the MNIST dataset with low label rates as proposed by Jiang et al. [68]. Bold values indicate best mean performances. Underlined values indicate second best mean performance. All the results for baseline have been taken from [68]. . . . . . . . . . . . . . . . . . . . . . . . . . . 87
List of Tables

C.1 Best set of hyperparameters for different datasets chosen on the validation set. .................................................. 143
List of Figures

2.1 An example of a (directed) graph with 8 nodes and 9 edges. In this example, the nodes are ordered as \(e_1, e_2, e_3, e_4, e_5, e_6, e_7,\) and \(e_8.\) The adjacency matrix associated to the graph and the node ordering is \(A.\) ................................. [7]

2.2 An example of a weighted graph with 8 nodes and 9 edges. In this example, the nodes are ordered as \(e_1, e_2, e_3, e_4, e_5, e_6, e_7,\) and \(e_8.\) The adjacency matrix associated to the graph and the node ordering is \(A.\) Each edge in the graph is accompanied by a positive number as the weight of the edge. ............... [8]

2.3 An example of a node attributed graph with 8 nodes and 9 edges. In this example, the nodes are ordered as \(e_1, e_2, e_3, e_4, e_5, e_6, e_7,\) and \(e_8.\) The adjacency matrix associated to the graph and the node ordering is \(A.\) Here, each node is accompanied by a vector as features for the nodes. ...................... [8]

2.4 An example of a relational graph with 8 nodes, 9 edges, and 3 relations. In this example, the nodes are ordered as \(e_1, e_2, e_3, e_4, e_5, e_6, e_7,\) and \(e_8.\) The relations are \(\mathcal{R} = \{r_1, r_2, r_3\}, \gamma(r_1) = \{(e_2, e_3, (e_6, e_7, (e_8, e_9))\}, \gamma(r_2) = \{(e_3, e_4), (r_2, e_6), (e_5, e_7)\},\) and \(\gamma(r_3) = \{(e_1, e_2), (r_5, e_3), (e_6, e_5)\}.\) ......................... [9]

2.5 An example of a hypergraph with 8 nodes and 3 hyperedges. In this example, the nodes are ordered as \(e_1, e_2, e_3, e_4, e_5, e_6, e_7,\) and \(e_8\) and the order of the nodes in a hyperedge is denoted as left to right. .................. [10]

2.6 An example of a relational hypergraph with 8 nodes and 4 hyperedges. In this example, the nodes are ordered as \(e_1, e_2, e_3, e_4, e_5, e_6, e_7,\) and \(e_8,\) the order of the nodes in each hyperedge is denoted as left to right, and \textit{arity} for \(r_1, r_2,\) and \(r_3\) is 3, 3, and 2 respectively. Also, \(\gamma(r_1) = \{(e_1, e_2, e_3)\}, \gamma(r_2) = \{(e_3, e_5, e_4), (e_8, e_7, e_6)\},\) and \(\gamma(r_3) = \{(e_6, e_5)\}.\) .................. [10]
3.1 hit@1 of SimplE, SimplE+, and logical inference when they are trained on different fractions (percentages) of the training data on sport dataset. The results indicate that when training data is scarce, logical inference outperforms (or is on-par with) SimplE, as SimplE does not see enough triples to be able to learn meaningful embeddings. As the amount of training data increases, SimplE starts to outperform logical inference as it can better generalize to unseen cases than pure logical inference. The gap between these two methods becomes larger as the amount of training data increases. For all tested fractions, SimplE+ outperforms both SimplE and logical inference as it uses both the generalization power of SimplE and the inference power of logical rules.

3.2 This experiment shows the loss value for SimplE and SimplE+ at each training epoch on sport dataset. Models are tested with the exact number of parameters and initialization. The results show that SimplE+ requires fewer epochs than SimplE to converge.

4.1 (a) Relation FliesBetween with arity 3 defined on three tuples. (b) Reifying non-binary relations by creating three additional entities $e_1$, $e_2$, and $e_3$. (c) Converting non-binary relations into cliques using star-to-clique. In this example, the three facts in the original graph (a) show that Air Canada has a flight from Montreal to Los Angeles and from Vancouver to New York. Figures (b) and (c) show two methods of converting this ternary relation into binary ones.

4.2 Visualization of HSimplE and HypE architectures. (a) $\phi_\theta$ for HSimplE transforms entity embeddings by shifting them based on their position and combining them with the relation embedding. (b) $f(e, i)$ for HypE takes an entity embedding and the position the entity appears in the given tuple, and returns a vector. (c) $\phi_\theta$ takes as input a tuple and outputs the score of HypE for the tuple.
These experiments show that HypE outperforms HSimplE when trained with fewer parameters, and when tested on samples that contain at least one entity in a position never encountered during training on JF17K dataset. (a) MRR of HypE and HSimplE for different embedding dimensions. (b) Results of m-CP, HSimplE, and HypE on the missing positions test set (containing 1,806 test samples).

An example of a knowledge hypergraph. Train set contains sold(drew, alex, book), buyer(alex, book), sold(mike, sam, tv), and bought(sam, mike, tv). Relation bought can be obtained by applying a renaming operation to relation sold. Similarly, relation buyer is a projection of relation sold. Learning these relational algebra operations can help the model generalize to the tuples in the test set. The test responses, from top to bottom, are drew, tv, and mike.

A schematic of the entity and relation embeddings in ReAlE: the embedding dimension $d$ is divided into $n_w$ windows of size $w$.

MRR of ReAlE for different number of windows $n_w$ on JF17K with fixed values for negative ratio, batch size, and embedding dimension of 10, 128, and 200 respectively. The total number of parameters for each value of $n_w$ is fixed.

Overview of SLAPS. At the top, a generator receives the node features and produces a non-symmetric, non-normalized adjacency having (possibly) both positive and negative values (Section 5.3.1). The adjacency processor makes the values positive, symmetrizes, and normalizes the adjacency (Section 5.3.2). The resulting adjacency and the node features go into GNN$_C$ which predicts the node classes (Section 5.3.3). At the bottom, some noise is added to the node features. The resulting noisy features and the generated adjacency go into GNN$_{DAE}$ which then denoises the features (Section 5.3.5).

Using a two-layer GCN, the predictions made for the labeled nodes are not affected by the dashed (starved) edge.
5.3 Comparing SLAPS, SLAPS$^2$, and kNN-GCN on cora with different generators. Although SLAPS$^2$ does not use the node labels in learning an adjacency matrix, it outperforms kNN-GCN. SLAPS outperforming SLAPS$^2$ indicates that learning the adjacency using both self-supervision and the task-specific node labels results in higher predictive accuracy. [88]

5.4 The performance of SLAPS with MLP graph generator as a function of $\lambda$. SLAPS with $\lambda = 0$ corresponds to removing self-supervision. The results indicate that with removing self-supervision, the model performance is somewhat poor. As soon as $\lambda$ becomes positive, both models see a large boost in performance showing that self-supervision is crucial to the high performance of SLAPS. [89]

5.5 Performance comparison when noisy graphs are provided as input ($\rho$ indicates the percentage of perturbations). The results indicate that self-supervision consistently provides a boost in performance, especially for higher values of $\rho$. [91]

5.6 The odds of two nodes in the test set sharing the same label as a function of the edge weights learned by SLAPS. For both cora and citeseer, nodes that are connected with higher edge weights are more likely to share the same label compared to nodes with lower or zero edge weights. This shows that the learned edge weights are highly predictive of the same classness. [92]

5.7 The performance of SLAPS on cora as a function of $k$ in kNN. The results show that the value of $k$ plays a major role in model performance. [93]

B.1 An example of an embedding where $|\tau| = 5$, $\alpha = 4$ and $f_3$ is the third fact in $\tau$. [120]

B.2 An example of an embedding where $|\tau| = 5$, $\alpha = 4$ and $f_3$ is the third fact in $\tau$. [121]

B.3 An example of a ReAIr embedding assignments for true tuples $\tau_0 = r_1(x_a, x_b, x_c)$, $\tau_1 = r_2(x_a, x_c)$, $\tau_2 = r_3(x_c, x_b, x_d)$ and $\tau_3 = r_1(x_d, x_c, x_a)$. Here, the number of true tuples $\lambda = n_w = 4$, maximum arity $\omega = 3$. Cells that are set to zero are left empty for better readability. As an example, given that $x_b$ is in position 1 of $\tau_2$, we set cell $2 \times \omega + 1 = 7$ of $x_b$ to 1. We further set the value of $r_1$ at positions $[1][7]$ to $K$. [124]
B.4 Proof of Theorem B.2: If at least one value of each vector \( f(x_i, 1) \) is different than zero \( (K_i \neq 0) \) and the value of at most one position \( i \) can be non-zero (the \( d \) columns in the figure), then by the pigeon-hole principle, we will run out of indices as we have more entities in the knowledge hypergraph. In the above image, the entity \( x_{d+1} \).

C.1 The performance of SLAPS on cora and citeseer with different adjacency symmetrizations.

C.2 The performance of SLAPS and regularization toward a manually defined prior structure on cora and citeseer when using the MLP generator.

C.3 Two example graph structures. Node colors indicate the class labels. Solid lines indicate homophilous edges and dashed lines indicate non-homophilous edges. The two graphs exhibit the same degree of homophily yet there is no overlap between their edges.
List of Software Packages

- **SimplE**
  - **Description:** An embedding model for link prediction in knowledge graphs with the ability to enforce taxonomies.
  - **Used in:** Chapter 3
  - **Paper:** Bahare Fatemi, Siamak Ravanbakhsh, and David Poole. *Improved Knowledge Graph Embedding Using Background Taxonomic Information*, Proceedings of the AAAI Conference on Artificial Intelligence, 2019.
  - **URL:**
    * [https://github.com/baharefatemi/SimplE](https://github.com/baharefatemi/SimplE)
    * [https://github.com/baharefatemi/SimplE-plus](https://github.com/baharefatemi/SimplE-plus)
  - **Languages and Tools:** Python and PyTorch.

- **HypE**
  - **Description:** An embedding model for link prediction in knowledge hypergraphs.
  - **Used in:** Chapter 4
  - **URL:** [https://github.com/baharefatemi/HypE](https://github.com/baharefatemi/HypE)
  - **Languages and Tools:** Python and PyTorch.

- **ReAIE**
  - **Description:** An embedding model for link prediction in knowledge hypergraphs.
List of Software Packages

- **Used in:** Chapter 4
- **URL:** [https://github.com/baharefatemi/ReAlE](https://github.com/baharefatemi/ReAlE)
- **Languages and Tools:** Python and PyTorch.

- **SLAPS**
  - **Description:** A framework for simultaneous Learning of adjacency and graph neural network parameters with self-supervision for semi-supervised node classification.
  - **Used in:** Chapter 5
  - **URL:** [https://github.com/BorealisAI/SLAPS-GNN](https://github.com/BorealisAI/SLAPS-GNN)
  - **Languages and Tools:** Python and PyTorch.
Acknowledgements

First and foremost, I would like to express my sincerest gratitude to my PhD supervisor and mentor, David Poole, for giving me this opportunity to pursue a PhD under his supervision. I am tremendously grateful for his continuous support, inspiring discussions, and constant feedback, and for giving me the freedom to pursue a variety of research directions during my PhD. His guidance and support both in academia and life have been truly invaluable.

My deepest acknowledgment goes to all my collaborators and internship hosts from Google Research, Facebook AI Research, Borealis AI, McGill, and Element AI: Perouz Taslakian, David Vazquez, Adriana Romero, Michal Drozdzal, Mehran Kazemi, Deepak Ramachandran, Siamak Ravanbakhsh, Simon Prince, Layla El Asri, Quentin Duval, and Rohit Girdhar. I am grateful for the funding they provided for the projects and all the support and encouragement they gave me. Without their guidance, constant feedback, and valuable ideas, this thesis would not have been achievable.

My deep and sincere gratitude to my family for their continuous and unparalleled love, help, and support. I am grateful to my brother and his wife, Hossein and Fatemeh, for always being there for me as a friend. I am forever indebted to my parents, Mahmoud and Esmat, for giving me the opportunities and experiences that have made me who I am. They selflessly encouraged me to explore new directions in life and seek my own destiny.

Finally, and above all, I cannot begin to express my unfailing gratitude and love to my husband, Mehran, who has supported me throughout this process and has constantly encouraged me when the tasks seemed arduous and insurmountable.
Dedication

Women, Life, Freedom
Dedicated to the women of Iran.
Chapter 1

Introduction

1.1 Machine Learning and Structure

On a day-to-day basis, we make predictions about the world. We predict the upcoming weather by checking the weather forecast to plan and dress for the day accordingly. In the financial sector and investment within the stock market and businesses, people make predictions about the share values of different companies and decide on buying and selling shares. Physicians predict the diseases, nutrition deficiencies, and drug side effects for each patient and decide on what actions to take based on these predictions.

Artificial intelligence \[113, 123\] studies methods for understanding the world, making predictions for unobserved events, and using predictions for decision making. A formal description of a world is often called a model of the world. An artificial intelligence system builds a model of the world to be able to make predictions about it and use prediction results in decision making. Machine learning is a sub-field of artificial intelligence. Its goal is to construct models of the world based on past experiences.

The world around us is composed of different objects each having various relations with the other objects. The objects and relations form a (hyper)graph with objects as nodes and relations between objects as (hyper)edges in the graph. For instance, a social network is in the form of a graph with users as nodes and interactions between users as edges. In chemistry, molecules form graphs with atoms as nodes and their bonds with each other as edges. The dependency parse tree of a text and the abstract syntax tree of a program form a graph as well. With modern technology and the rise of the internet, graphs are everywhere: we can see graphs more frequently as the world wide web, street maps, and knowledge bases used in search engines.

The ubiquity of such a graph-structured description of the world calls for the development of effective methods that make use of and learn to understand information represented in this structural form. The machine learning community has long noticed the importance of structure in data. The primary form of data in the subfield of graph representation learning \[57\]
1.2. Learning with Explicit Structure

is in the form of nodes and edges among them.

A common approach for applying machine learning to discrete data, such as text and graphs, is through learning embeddings. Word, sentence, and paragraph embeddings [13, 73, 98, 111], which vectorize words, sentences, and paragraphs using context information, are widely used in a variety of natural language processing tasks from syntactic parsing to sentiment analysis. In graph representation learning, the goal is to combine given properties of nodes and edges as well as the structure into representations. The embeddings are later used for different tasks e.g., for node classification [e.g., 41, 58, 83], link prediction [e.g., 15, 39, 76, 139], and graph classification [e.g., 37, 167].

This thesis is structured in two parts: Chapters 3 and 4 introduces embedding-based models for a variety of learning tasks with explicitly graph-structured data, i.e., datasets that are given to us in the form of entities and their relations. Chapter 5 explores the topic of learning with implicit structure in the form of structural inductive biases, applied to tasks such as node classification. In this part, we are not given an explicitly structured dataset, but we develop models that infer or make use of hidden structures in the data to achieve improved generalization compared to using unstructured deep models.

1.2 Learning with Explicit Structure

The underlying structure representing the relations between the nodes is sometimes given in the task. Such entity-relationships are often represented in the form of triples of relationship (head, tail) with head and tail representing entities and a relationship between the two entities. The triple representation can be expressed in the form of a graph, with nodes as entities and directed edges from the head node to the tail node labeled with the relationship between them.

One instance of an explicit structure is knowledge graphs. Knowledge graphs contain knowledge about the world and provide a structured representation of the knowledge in the form of a graph. Such graph structures have applications in several tasks such as search [130], automatic question answering [45, 160], relation extraction [115], and recommender systems [18]. A large number of knowledge graphs have been created and are publicly available, such as YAGO [135], DBPEDIA [6], NELL [19], FREEBASE [14], and Google Knowledge Vault [34]. Since accessing and storing all the knowledge (true triples) in the world is difficult, knowledge graphs are incomplete; the
1.2. Learning with Explicit Structure

goal of link prediction in knowledge graphs (or knowledge graph completion) is to predict unknown true triples or relationships between entities based on existing ones.

In graph representation learning, embedding-based methods \[104, 106, 147\] use embeddings for entities and relationships to generalize from existing data. The embeddings are learned such that their interaction through mathematical operations best predicts the (probability of the) existence of the observed triples. These approaches learn embeddings for entities and relations. To find out if \(r(h, t)\) is a fact (i.e., is true), such models define a function that embeds relationship \(r\) and entities \(h\) and \(t\), and produces the probability that \(r(h, t)\) is true. Embedding-based methods have been used in several reasoning tasks in knowledge graphs.

The contributions of this thesis for this part are guided by the following research questions:

**Research Question 1** Can we develop and implement efficient embedding-based models that can enforce taxonomic information of ontologies in the learned embeddings space?

Embedding-based models rely on a large number of annotated triples to learn useful representations of entities and relationships. However, in many domains, often there exists other information in ontologies that specify the meaning of the symbols used in a knowledge base. Ontology can include taxonomic (subclass and subproperty) information. The provided taxonomic information can be exploited as extra information to help the embedding-based model learn a useful representation for entities and relationships. Our main contribution to addressing this question is a novel embedding-based model that we call SimplE+ and is introduced in Chapter 3. SimplE+ can enforce taxonomic information into the learned embeddings and achieves advantages over earlier state-of-the-art methods both in terms of efficiency and predictive accuracy for knowledge graph completion.

**Research Question 2** Can embedding-based models for knowledge graph completion incorporate n-ary relations?

Most of the existing embedding-based methods can only handle relations with at most two arguments but in the real world, there exist relations with more than two arguments. The number of arguments a relation takes is called *arity* of the relation. For instance, the relation \(sells(S, B, I, P, T)\), which is true when the entity \(S\), the seller, sells item \(I\) to buyer \(B\) for price
1.3 Learning with Implicit Structure

$P$ at time $T$, has arity five. There are approaches to converting higher-arity relations to binary ones, however, one contribution of this thesis is showing in Section 4.1 that current embedding-based methods do not work well in the presence of such conversion approaches. Another contribution of this thesis is introducing datasets, baselines, and evaluation metrics for the task of knowledge completion with n-ary relations. We show that to handle higher-arity relations, we need to devise new methods that can handle such entities. In Sections 4.2 and 4.3 we introduce embedding-based models that demonstrate significant improvement over the proposed baselines.

1.3 Learning with Implicit Structure

Having introduced novel architectures for explicitly graph-structured data in Chapters 3 and 4 of this thesis, Chapter 5 investigates how models with structural and compositional inductive biases — in particular graph neural networks — can be developed and applied to problems with the implicit or hidden structure.

Research Question 3 Can we extend the applicability of models that work with structural inductive biases to problems where a graph structure is not readily available?

Graph neural networks (GNNs) are a class of neural network models suitable for processing graph-structured data. GNNs learn embeddings for nodes in the graph based on feature information available for the nodes as well as the structure of the graph. GNNs work well when the graph structure is provided. However, this structure may not always be available in real-world applications. One solution to this problem is to infer a task-specific latent structure and then apply a GNN to the inferred graph. We identify a supervision starvation problem in such approaches in which the edges between pairs of nodes that are far from labeled nodes receive insufficient supervision; this results in learning poor structures away from labeled nodes and hence poor generalization. So the task-specific supervision may be insufficient for learning both the structure and the GNN parameters. In (Chapter 5) of this thesis, we propose the simultaneous learning of a graph structure and graph neural network parameters with self-supervision. A comprehensive experimental study demonstrates that the proposed model scales to large graphs with hundreds of thousands of nodes and outperforms several models that have been proposed to learn a task-specific graph structure on established benchmarks.
Chapter 2

Background and Notation

In this chapter, we provide an introduction to the topics and notation that are used throughout this thesis. Additional background will be introduced where necessary in later chapters.

2.1 Notation

We use lowercase letters to denote scalars, bold lowercase letters to denote vectors, bold uppercase letters to denote matrices, and calligraphic letters to denote sets.

Let \( \mathbb{R} \) denote the space of real numbers and \( \mathbb{R}^n \) denote the \( n \)-dimensional real space. The space of matrices of real numbers with \( n \) rows and \( m \) columns is denoted by \( \mathbb{R}^{n \times m} \). Otherwise, we use symbol \( \times \) for the multiplication of two scalars. Following Python notation, for a vector \( \mathbf{v} \), we represent its \( i \)th element as \( \mathbf{v}[i] \). For a matrix \( \mathbf{M} \), we represent the \( i \)th row as \( \mathbf{M}[i] \) and the element at the \( i \)th row and \( j \)th column as \( \mathbf{M}[i][j] \). \( \mathbf{I} \) represents an identity matrix.

We use \( \text{Re}(\mathbf{x}) \) and \( \text{Im}(\mathbf{x}) \) to denote the real and imaginary parts of a complex vector \( \mathbf{x} \). The element-wise conjugate of the complex vector \( \mathbf{a} + i\mathbf{b} \) is defined as \( \mathbf{a} + i\mathbf{b} = \mathbf{a} - i\mathbf{b} \) (where \( i = \sqrt{-1} \)).

The notation is summarized with examples in Table 2.1.

Table 2.1: Summary of notation with examples.

<table>
<thead>
<tr>
<th>Examples</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x, y, ) and ( z )</td>
<td>Lowercase letters denote scalar.</td>
</tr>
<tr>
<td>( \mathbf{x}, \mathbf{y}, ) and ( \mathbf{z} )</td>
<td>Bold lowercase letters denote vectors.</td>
</tr>
<tr>
<td>( \mathbf{X}, \mathbf{Y}, ) and ( \mathbf{Z} )</td>
<td>Bold uppercase letters denote matrices.</td>
</tr>
<tr>
<td>( \mathcal{X}, \mathcal{Y}, \mathcal{Z} )</td>
<td>Calligraphic letters denote sets.</td>
</tr>
<tr>
<td>( \mathbf{v}[i] )</td>
<td>The ( i )th element of a vector ( \mathbf{v} ).</td>
</tr>
<tr>
<td>( \mathbf{M}[i] )</td>
<td>The vector at the ( i )th row of matrix ( \mathbf{M} ).</td>
</tr>
<tr>
<td>( \mathbf{M}[i][j] )</td>
<td>The element at the ( i )th row and ( j )th column of matrix ( \mathbf{M} ).</td>
</tr>
</tbody>
</table>
2.2. Learning on Graphs

The variadic function `concat` takes input a sequence of vectors \( \mathbf{v}_1, \ldots, \mathbf{v}_k \) and outputs the concatenation of its input vectors. We define the variadic function \( \odot \) to be the sum of the element-wise product of its input vectors, namely \( \odot(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k) = \sum_{i=1}^{k} \mathbf{v}_1[i] \times \mathbf{v}_2[i] \times \cdots \times \mathbf{v}_k[i] \) where each vector \( \mathbf{v}_i \) has the same length.

The 1D convolution operator \( \ast \) takes as input a vector \( \mathbf{v} \in \mathbb{R}^n \), a vector \( \omega \in \mathbb{R}^m \) as the convolution weight filter, and a stride \( s \) (\( n \) should be divisible by \( s \)) and outputs a vector of size \( n/s \). The operation \( \ast \) is the standard 1D convolution as defined in `torch.nn.Conv1D` function in PyTorch [109]. By convention, if index \( j \) is outside the range of \( v \), \( \mathbf{v}[j] = 0 \). The \( x \)-th index of the output of the 1D convolution \( (\mathbf{v} \ast \omega) \) is defined as follows, where stride is implicit in the convolution function.

\[
(\mathbf{v} \ast \omega)[x] = \sum_{i=0}^{m-1} \mathbf{v}[x \times s + i] \times \omega[i]
\]

A permutation function on \( \{1, \ldots, n\} \) is a bijection \( \pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\} \). For example, if \( \pi \) has \( 1 \mapsto 2, \ 2 \mapsto 1, \) and \( 3 \mapsto 3 \), then, \( (x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)}) = (x_2, x_1, x_3) \).

2.2 Learning on Graphs

2.2.1 Types of Graphs

A directed graph with \( n \) nodes is denoted as \( G = \{V, A\} \). \( V = \{e_1, \ldots, e_n\} \) is a set of nodes and \( A \) is a set of ordered pairs of edges. One can impose an ordering on \( V \) and represent \( A \) as a matrix \( A \in \{0, 1\}^{n\times n} \) and \( G \) as \( \{V, A\} \). We refer to \( A \) as adjacency matrix. When \( A[i][j] = 1 \), there is a directed edge in the graph from node \( e_i \) to node \( e_j \). When \( A[i][j] = 0 \), there is no edge in the graph from node \( e_i \) to node \( e_j \). If a \( G \) has \( m \) edges, \( A \) has \( m \) non-zero values. Figure 2.1 shows an example of a graph with 8 nodes and 9 edges and the adjacency matrix associated to the graph. An undirected graph has a symmetric adjacency matrix, i.e., \( A[i][j] = A[j][i] \). In this thesis, we mainly work with directed graphs. Hereafter, we use the term graph to refer to a directed graph.

A weighted graph is a graph with a function from edges into positive numbers. When \( V \) is ordered, the edges and their corresponding weights can be represented as an adjacency matrix \( A \in \mathbb{R}^{n\times n} \) with non-negative values. The weight of the edge from node \( e_i \) to node \( e_j \) is denoted by \( A[i][j] \). When \( A[i][j] = 0 \), there is no edge in the graph from node \( e_i \) to node \( e_j \). The
2.2. Learning on Graphs

Figure 2.1: An example of a (directed) graph with 8 nodes and 9 edges. In this example, the nodes are ordered as $e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8$. The adjacency matrix associated to the graph and the node ordering is $A$.

The number of positive values in $A$ is $m$. Figure 2.2 shows an example of a weighted graph with 8 nodes and 9 edges.

A node attributed graph is a graph with a function from nodes $\mathcal{V}$ into $\mathbb{R}^f$. When $\mathcal{V}$ is ordered, the attributes can be represented as matrix $X \in \mathbb{R}^{n \times f}$ whose $i$-th row correspond to a vector of $f$ descriptive attributes as features for node $e_i$ and $G$ can be represented as $\{\mathcal{V}, A, X\}$. We refer to $X$ as node features. In this thesis, an attributed graph refers to a node attributed graph. An example of an attributed graph is CORA [27]. CORA is a citation network with nodes as publications and edges as citations. Each node (each publication) in the dataset is described by a binary-valued word vector. The $i$-th value in the vector is 1 if the $i$-th word in the dictionary appears anywhere in the document and is 0 otherwise. Figure 2.3 shows an example of a node attributed graph.

A relational graph is $G = \{\mathcal{V}, \mathcal{R}, \gamma\}$ where $\mathcal{V}$ is a set of nodes, $\mathcal{R}$ is a set of relations, $\gamma$ is a function from $\mathcal{R}$ into sets of ordered pairs of nodes. There is an edge from $e_i$ to $e_j$ with relation $r$, written as $r(e_i, e_j)$, when $(e_i, e_j) \in \gamma(r)$. An example of a relational graph is the WORDNET dataset with nodes as words and relations for instance hyperonymy and hyponyms and edges indicating what words are hypernym or hyponym of each other. Figure 2.4 shows an example of a relational graph with $\mathcal{R} = \{r_1, r_2, r_3\}$. The $\gamma$ function for each relation is defined as $\gamma(r_1) = \{(e_2, e_3), (r_6, e_7), (e_8, e_6)\}$, $\gamma(r_2) = \{(e_3, e_4), (r_2, e_6), (e_5, e_7)\}$, and $\gamma(r_3) = \{(e_1, e_2), (r_5, e_3), (e_6, e_5)\}$.

A hypergraph is a $\mathcal{G} = \{\mathcal{V}, A\}$ where $\mathcal{V}$ is a set of nodes and $A$ is a subset of the power sets of $\mathcal{V}$. Figure 2.5 shows an example of a hypergraph with
2.2. **Learning on Graphs**

Figure 2.2: An example of a **weighted graph** with 8 nodes and 9 edges. In this example, the nodes are ordered as \(e_1, e_2, e_3, e_4, e_5, e_6, e_7\), and \(e_8\). The adjacency matrix associated to the graph and the node ordering is \(A\). Each edge in the graph is accompanied by a positive number as the weight of the edge.

Figure 2.3: An example of a **node attributed graph** with 8 nodes and 9 edges. In this example, the nodes are ordered as \(e_1, e_2, e_3, e_4, e_5, e_6, e_7\), and \(e_8\). The adjacency matrix associated to the graph and the node ordering is \(A\). Here, each node is accompanied by a vector as features for the nodes.
2.2. Learning on Graphs

A relational hypergraph is a $\mathcal{G} = \{V, R, \text{arity}, \gamma\}$ where $V$ is a set of nodes, $R$ is a set of relations, $\text{arity}$ is a function from $R$ to natural numbers, $\gamma$ is a function from $R$ into sets of tuples of nodes, such that the tuples in $\gamma(r)$ are all of length $\text{arity}(r)$. We write $(e_1, \ldots, e_k) \in \gamma(r)$ as $r(e_1, \ldots, e_k)$. We call $r(e_1, \ldots, e_k)$ a tuple. A tuple with $k = 2$ is called a triple. The head of a triple $r(e_1, e_2)$ is the first argument of the relation, $e_1$, and the tail refers to the second argument of the relation, $e_2$. Figure 2.6 shows an example of a relational hypergraph with 8 nodes and 4 hyperedges. In this example, there are 3 relations $r_1$, $r_2$, and $r_3$ with arities 3, 3 and 2 respectively. There is one tuple in $\gamma(r_1)$ as $(e_1, e_2, e_3)$. The tuples in $\gamma(r_2)$ are $(e_3, e_4, e_5)$ and $(e_8, e_7, e_6)$ and the only tuple (triple) in $\gamma(r_3)$ is $(e_6, e_5)$.

There are also combinations of the different types of graphs like an attributed relational graph or a weighted relational hypergraph.

2.2.2 Tasks on Graphs

Various tasks can be defined on different types of graphs. Node classification, link prediction, graph classification, and subgraph classification are examples of tasks on graphs. In this thesis, we focus on node classification and link prediction.
2.2. Learning on Graphs

Figure 2.5: An example of a hypergraph with 8 nodes and 3 hyperedges. In this example, the nodes are ordered as $e_1$, $e_2$, $e_3$, $e_4$, $e_5$, $e_6$, $e_7$, and $e_8$ and the order of the nodes in a hyperedge is denoted as left to right.

Figure 2.6: An example of a relational hypergraph with 8 nodes and 4 hyperedges. In this example, the nodes are ordered as $e_1$, $e_2$, $e_3$, $e_4$, $e_5$, $e_6$, $e_7$, and $e_8$, the order of the nodes in each hyperedge is denoted as left to right, and arity for $r_1$, $r_2$, and $r_3$ is 3, 3, and 2 respectively. Also, $\gamma(r_1) = \{(e_1, e_2, e_3)\}$, $\gamma(r_2) = \{(e_3, e_5, e_4), (e_8, e_7, e_6)\}$, and $\gamma(r_3) = \{(e_6, e_5)\}$.
2.3. Knowledge Graphs and Hypergraphs

Node classification. In a node classification task, the graph is accompanied by a set of labels (labels are sometimes called classes) and a partial function from nodes to labels. \( \mathcal{C} \) corresponds to the labels and \(|\mathcal{C}|\) corresponds to the number of labels. The goal, here, is to predict the labels for the unlabeled nodes. Semi-supervised learning is an approach to machine learning that combines a small number of labeled data with a large amount of unlabeled data during training. Semi-supervised node classification in graphs is defined as when a small number of nodes in the graph are labeled.

Link prediction. In the link prediction task, the goal is to use existing edges among the nodes and, if applicable the relations to predict new edges in the graph.

2.3 Knowledge Graphs and Hypergraphs

2.3.1 Definition

A world consists of a finite set of entities \( \mathcal{E} \), a finite set of relations \( \mathcal{R} \). The arity for each relation is fixed. The function \( \text{arity} \), given a relation \( r \), outputs the number of arguments that the relation takes. The arity for relation \( r \) is also denoted as \(|r|\). The function \( \text{arity} \) is the same as in a relational hypergraph.

Let \( \tau \) be a set of tuples – the ground truth – specifying all of the tuples that are true. If a tuple is not in \( \tau \), it is false. If \( \tau \) represents the set of true tuples in the real world, then the following tuple is in \( \tau \).

\[
\text{FliesBetween}(\text{AirCanada}, \text{Vancouver}, \text{NewYork})
\]

An example of a triple in \( \tau \) is the following.

\[
\text{CapitalCityOfCountry}(\text{Paris}, \text{France})
\]

An example of a tuple that is not in \( \tau \) is

\[
\text{FliesBetween}(\text{AirCanada}, \text{SanFransisco}, \text{NewYork})
\]

and an example of a triple that is not in \( \tau \) is as follows.

\[
\text{CapitalCityOfCountry}(\text{Paris}, \text{Germany})
\]

We do not observe the ground truth \( \tau \); we only observe a subset, \( \tau' \). A knowledge hypergraph is a relational hypergraph \( \mathcal{G} = \{\mathcal{V}, \mathcal{R}, \text{arity}, \gamma\} \) with
2.3. Knowledge Graphs and Hypergraphs

\[ V = \mathcal{E} \] that stores \( \tau' \). For a knowledge hypergraph, \( \tau' \) is the union of all tuples \( r(e_i, \ldots, e_j) \) such that \( r \in \mathcal{R} \) and \( (e_i, \ldots, e_j) \in \gamma(r) \). A knowledge graph is a relational graph and is a special case of a knowledge hypergraph where all relations have arity 2: \( \forall r \in \mathcal{R} : \text{arity}(r) = 2 \). Link prediction in knowledge hypergraphs or knowledge (hyper)graph completion is the problem of predicting the missing tuples in \( \tau' \), that is, finding the tuples \( \tau \setminus \tau' \).

As benchmarks for knowledge hypergraphs contain only positive tuples and for contrastive learning of knowledge hypergraphs, we also need negative tuples, we follow the literature [e.g., 15, 76, 139] and generate a set of negative tuples for each positive tuple. The function \( T_{\text{neg}} \), given a positive tuple \( x \in \tau' \), generates a set of negative tuples corresponding to \( x \) (The details of \( T_{\text{neg}} \) are discussed later).

An embedding is a function that given an entity or a relation it provides one or more vectors, matrices, or higher-order tensors of numbers over a field (typically the real numbers). We use bold for embeddings, so that, \( \mathbf{x} \) is the embedding of entity \( x \) in a vector and \( \mathbf{X} \) is the embedding of entity \( x \) in a matrix. \( \mathbf{r} \) is the embedding of relation \( r \) in a vector, and \( \mathbf{R} \) is the embedding of relation \( r \) in a matrix. An embedding-based model defines whether to represent entities or relations into vectors or matrices.

For the task of knowledge hypergraph completion, an embedding-based model with parameters \( \theta \) defines a scoring function \( \phi_{\theta} \) that takes a tuple as input and generates a prediction, e.g., a score (or a probability) of the tuple being true. A model is fully expressive if given any set \( \tau \), there exists an assignment of values to \( \theta \) that separates the true tuples from the false ones i.e., there exists a \( \beta \) such that \( x \in \tau \iff \phi_{\theta}(x) > \beta \).

2.3.2 Knowledge Graph Completion Methods

Tensor factorization approaches have shown promising results for the task of knowledge graph completion [15, 105, 106, 140]. Such approaches learn embeddings for entities and relations. Their scoring function \( \phi_{\theta} \) for a triple \( r(h, t) \) takes the embedding of entities \( h \) and \( t \) and relation \( r \) and outputs a prediction. Here, we explain some important tensor factorization approaches that are of central importance to the topics covered in this thesis. All the following scoring functions contain a non-linearity \( \sigma \) - e.g., \( \sigma : \mathbb{R} \rightarrow [0, 1] \) can be the sigmoid or logistic function.

**Canonical Polyadic (CP)**. CP decomposition [59] is one generalization of the matrix singular value decomposition [8] to tensors in multi-linear algebra. CP can be used for link prediction in knowledge graphs. CP embeds
2.3. Knowledge Graphs and Hypergraphs

Each entity $e$ as two vectors $e^+$ and $e^-$, and each relation $r$ as a single vector $r$; it defines its score function as follows:

$$\phi_\theta(r(e_i, e_j)) = \sigma(\odot(r, e_i^+, e_j^-))$$

The two embeddings $e^+$ and $e^-$ for an entity $e$ are learned independently (and are unrelated to) each other. This independence of the two learned entity embeddings causes CP to perform poorly for knowledge graph completion [139].

**DistMult.** DistMult [158] is one of the simplest knowledge graph embedding approaches. For each entity $e$ and relations $r$, it learns only one embedding $e$ and $r$. It defines its scoring function as

$$\phi_\theta(r(e_i, e_j)) = \sigma(\odot(r, e_i^+, e_j^-)). \quad (2.1)$$

Since Equation 2.1 is commutative, it cannot distinguish between the same entity appearing as in the first or the second argument of a triple. Therefore, it can only model symmetric relations.

**ComplEx.** ComplEx [140] extends DistMult by considering complex values instead of real values for embedding vectors of entities and relations. It defines the scoring function $\phi_\theta$, such that the embedding of each entity and each relation is a vector of complex numbers.

ComplEx defines the probability of any triple $r(e_i, e_j)$ as

$$\phi_\theta(r(e_i, e_j)) = \sigma(\text{Re}(\odot(r, e_i^+, \bar{e}_j^-))).$$

Note that, if the tail did not use the conjugate, it would be equivalent to DistMult [157]. Trouillon et al. [141] prove that ComplEx is fully expressive. In particular, they prove that any assignment of ground truth can be modeled by ComplEx embeddings of length $|\mathcal{E}| \times |\mathcal{R}|$.

**SimplE.** SimplE [76] introduces an enhancement of CP to allow the two embeddings of each entity to be learned dependently. SimplE considers two embeddings for each relation: one for the relation $r \in \mathcal{R}$ itself and one for its inverse. We use $r^+$ to denote the “forward” embedding of $r$ and $r^-$ to denote the embedding of its inverse. The embedding $r = \text{concat}(r^+, r^-)$ for a relation is a concatenation of these two parts. Similarly, the embedding for each entity $e \in \mathcal{E}$ has two parts: its embedding as a head $e^+$ and as a tail $e^-$ — that is $e = \text{concat}(e^+, e^-)$. Using this notation, SimplE calculates
2.4. Graph Neural Networks

the probability of a triple \(r(h, t)\) being true in both forward and backward
directions using

\[
\phi_\theta(r(e_i, e_j)) = \sigma \left( \frac{1}{2}(\odot(r^+, e_i^+, e_j^-) + \odot(r^-, e_j^+, e_i^-)) \right)
\]

Kazemi and Poole [76] prove SimplE is fully expressive and provide a
bound on the size of the embedding vectors: For any truth assignment \(\tau' \subseteq \mathcal{R} \times \mathcal{E} \times \mathcal{E}\), there exists a SimplE model with embedding vectors of size
\[
\min(|\mathcal{E}| \times |\mathcal{R}|, |\tau'| + 1)
\]
that represent the assignment.

2.4 Graph Neural Networks

2.4.1 Motivation and Intuition

Graph neural networks (GNNs) are a class of neural network models suitable
for processing graph-structured data. The key idea in GNNs is to learn
embeddings for nodes in the graph based on feature information available
for the nodes as well as the structure of the graph. The main difference
between GNNs and the models studied in Section 2.3.2 is that GNNs learn
embeddings for nodes based on the graph structure – the neighbors and the
neighbors of the neighbors etc. – and node features but the previous models
directly learn node embeddings.

One feature of GNNs is the neural message passing. The message passing
in GNNs is defined such that at each layer, each node sends messages to its
neighboring nodes. After the messages are sent, each node receives the
messages from its neighboring nodes and aggregates them to update its
embedding.

The variants of GNNs differ in how they define a function to combine the
graph structure and node features to send messages and how to aggregate
the messages on one node to get updated embeddings. Next, we describe
graph convolutional networks [83], a specific type of GNNs which has been
used in this thesis.

2.4.2 Graph Convolutional Networks

We first summarize a graph convolutional networks (GCNs) [83] by stating
its input, intermediate outputs, and final output. We then explain the
components of a GCN.

Input. The input to a GCN is a weighted attributed graph \(\mathcal{G} = \{V, A, X\}\).
2.4. Graph Neural Networks

Intermediate outputs. Each layer in a GCN computes extra attributes, called embeddings, for each node. Let \( H^{(l)} \in \mathbb{R}^{n \times d_l} \) denote the node embedding for layer \( l \) with \( n \) as the number of nodes and \( d_l \) as the number of attributes per node in layer \( l \). Here, \( H^{(l)} \) is computed using the parameters of the layer \( W^{(l)} \), and the previous layer node embeddings \( H^{(l-1)} \). The input node embeddings \( H^{(0)} \) is \( X \), the provided node attributes.

Final output. The final output of a GCN is the node embeddings for the last layer, \( H^{(L)} \).

We now explain the components of a GCN.

Self-loops. Each node receives messages from its neighboring nodes. The neighboring information is stored in the adjacency matrix \( A \). GCN connects each node to itself to enable a node to send messages to itself enabling the accumulation of information. Summing the adjacency matrix \( A \) with an identity matrix \( I \) is equal to connecting each node to itself. The adjacency matrix \( A \) plus connecting each node to itself is denoted by \((A + I)\).

Normalization. GCN normalizes the adjacency matrix \( A \) in such a way that the messages sent across the edges incorporate the degree of the nodes in the messages.

Let \( D \) be a diagonal degree matrix for \((A + I)\) defined as

\[
D[i][i] = 1 + \sum_j A[i][j]
\]

with non-diagonal values as 0. For directed graphs where the adjacency is not necessarily symmetric, the following corresponds to a row normalized adjacency matrix with self-loops. The edge weights in each row of the adjacency matrix are divided by the degree of the node corresponding to the row. The value of \((A + I)[i][j]\) is normalized in \( \hat{A} \) using the degree of the node \( e_i \) as

\[
\hat{A}[i][j] = \frac{(A + I)[i][j]}{D[i][i]}.
\]

Row-normalization in matrix notation is defined as

\[
\hat{A} = D^{-1}(A + I).
\]

For undirected graphs where the adjacency is symmetric, the following corresponds to a row-and-column normalized adjacency matrix with self-loops. Using the row-and-column normalization, \((A + I)[i][j]\) is normalized
2.4. Graph Neural Networks

as

\[
\hat{A}[i][j] = \frac{(A + I)[i][j]}{\sqrt{D[i][i] \times D[j][j]}}.
\]

The matrix notation of the row-and-column normalization is defined as

\[
\hat{A} = D^{-\frac{1}{2}}(A + I)D^{-\frac{1}{2}}.
\] (2.2)

**GCN layer.** Each layer of a GCN aggregates the incoming messages to all nodes and computes node embeddings based on the received messages. Each GCN layer has learnable weight parameters. The weight parameters for layer \( l \) in a GNN is denoted by \( W^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l} \). Layer \( l \) computes node embeddings \( H^{(l)} \in \mathbb{R}^{n \times d_l} \) using the previous layer node embeddings \( H^{(l-1)} \) and its weight parameters \( W^{(l)} \). The input node embeddings \( H^{(0)} \) is \( X \), the provided node attributes. The GCN model defines the message passing function as

\[
H^{(l)}[i] = \sigma \left( \sum_{j \in N(i) \cup \{i\}} (\hat{A}[i][j] \ H^{(l-1)}[j]) \ W^{(l)} \right)
\]

where \( \sigma \) is an activation function such as ReLU \[103\], \( H^{(l)}[i] \) is the embedding of the node \( e_i \) at layer \( l \), \( N(i) \) is the set of indices of neighboring nodes for node \( e_i \) according to \( A \), and \( H^{(l-1)}[j] \) is the embedding of the node \( e_i \) at layer \( l-1 \). \( j \in N(i) \cup \{i\} \) corresponds to adding self-loops to the adjacency.

The message passing function of GCN in matrix notation is

\[
H^{(l)} = \sigma(\hat{A} H^{(l-1)} W^{(l)}).
\] (2.3)

A GCN with \( L \) layers has parameters \( W^{(1)}, \ldots, W^{(L)} \) with each \( W^{(l)} \) as the weight matrix for layer \( l \). These \( L \) layers are stacked on top of each other to output \( H^{(L)} \) as the final node embeddings to be used for tasks on nodes.

**Node classification using a GCN.** For node classification using GCNs, we set \( d_L \) (the second dimension in the weight parameter of the last layer in GCN) to the number of labels \( |C| \) and apply a softmax function to \( H^{(L)} \) so that its elements in each row lie in the range of \([0, 1]\) and sum to 1 to reflect probabilities.

\[
\text{softmax}(H^{(L)}[i][j]) = \frac{H^{(L)}[i][j]}{\sum_{c \in C} H^{(L)}[i][c]}
\] (2.4)
2.5. Ranking-based Evaluation Metrics

The value $\text{softmax}(H^{(L)}[i][j])$ denotes the output probability of the node $i$ being classified as label $j$.

2.5 Ranking-based Evaluation Metrics

Given a knowledge hypergraph defined on $\tau'$, let $\tau'_\text{train}$, $\tau'_\text{test}$, and $\tau'_\text{valid}$ denote the train, test, and validation sets, respectively, so that $\tau' = \tau'_\text{train} \sqcup \tau'_\text{test} \sqcup \tau'_\text{valid}$ where $\sqcup$ is disjoint set union. We use two evaluation metrics: Hit@k and Mean Reciprocal Rank (MRR). Both these measures rely on the ranking of a tuple $x \in \tau'_\text{test}$ within a set of corrupted tuples. For each tuple $r(e_1, \ldots, e_k)$ in $\tau'_\text{test}$ and each entity position $i$ in the tuple, we generate $|E| - 1$ corrupted tuples by replacing the entity $e_i$ with each of the entities in $E \setminus \{e_i\}$. For example, by corrupting entity $e_i$, we would obtain a new tuple $r(e_1, \ldots, e'_i, \ldots, e_k)$ where $e'_i \in E \setminus \{e_i\}$. Let the set of corrupted tuples for position $i$, plus $r(e_1, \ldots, e_k)$, be denoted by $\theta_i(r(e_1, \ldots, e_k))$. Let $\text{rank}_i(r(e_1, \ldots, e_k))$ be the ranking of $r(e_1, \ldots, e_k)$ within $\theta_i(r(e_1, \ldots, e_k))$ based on the score $\phi_\theta(x)$ for each $x \in \theta_i(r(e_1, \ldots, e_k))$. In an ideal knowledge hypergraph prediction method, $\text{rank}_i(r(e_1, \ldots, e_k))$ is 1 among all corrupted tuples. For a tuple $x$, let $|x|$ denote the number of entities in the tuple. $z$ is the number of prediction tasks and is computed as follows.

$$z = \sum_{x \in \tau'_\text{test}} |x|$$

We then compute the MRR as follows.

$$\text{MRR} = \frac{1}{z} \sum_{x \in \tau'_\text{test}} \sum_{i=1}^{|x|} \frac{1}{\text{rank}_i(x)}$$

Hit@k measures the proportion of tuples in $\tau'_\text{test}$ that rank among the top $k$ in their corresponding corrupted sets. Following the literature, we remove all corrupted tuples that are in $\tau'$ from our computation of MRR and Hit@k.
Chapter 3

Incorporating Ontological Information

The ideas and results presented in this chapter have been published in Fatemi et al. [39].

3.1 Introduction

In a knowledge graph, embeddings of entities and relationships are used to generalize from existing data. These embeddings are often formulated in terms of tensor factorization models [15, 76, 106, 140]. Here, the embeddings are learned such that their interaction through (tensor-)products best predicts the (probability of the) existence of the observed triples; see [104, 147] for details and discussion. Tensor factorization methods have been successful, yet they rely on a large number of annotated triples to learn useful representations. There is often other information in ontologies that specifies the meaning of the symbols used in a knowledge base. One type of ontological information is represented in a hierarchical structure called a taxonomy. For example, Donald Trump is the Wikidata [2] entry for Q22686. Wikidata contains the information that Donald Trump held the position of president of the United States and that he is a human, but does not contain information that he is a person, a mammal, and an animal, because these are implied by taxonomic knowledge. Being told that mammals are chordates, lets us conclude that Donald Trump is also a chordate, without needing to have triples specifying this about multiple mammals. We could also have information about subproperties, such as that holding the position of president of the United States is a subproperty of being the head of state, which in turn is a subproperty of being a leader.

This chapter is about combining taxonomic information in the form of subclass and subproperty into knowledge graph embedding models. We show that some of the well-known existing embedding-based models that are fully expressive cannot reflect such constraints for all legal entity embeddings.
3.2. Related Work

We propose a model that is provably fully expressive and can represent such taxonomic information, and evaluate its performance on real-world datasets.

3.2 Related Work

Incorporating background knowledge in knowledge graph embedding methods has been the focus of several studies. Here, we categorize these approaches by emphasizing the shortcomings that are addressed in this chapter; see [107] for a review of knowledge graph embedding methods.

**Soft rules.** There is a large family of link prediction models based on soft first-order logic rules [e.g., 30, 77, 117]. These models either receive a set of handcrafted logic rules as input or learn the rules from data. Each rule is attached with a (typically learned) weight showing some notion of confidence of the model for the correctness of the rule. Different models define different semantics over the rules and their weights to predict new information. While these models can be easily integrated with background taxonomic information, they typically cannot generalize to unseen cases beyond their rules. Exceptions include Fatemi et al. [38], Kazemi and Poole [75] which combine (stacked layers of) soft rules with entity embeddings, but these models have only applied to property prediction for entities. Approaches based on path-constrained random walks [e.g., 85] suffer from similar limitations as they have been shown to be a subset of probabilistic logic-based models [74].

**Augmentation by grounding of the rules.** One simple way to incorporate a set of rules in a knowledge graph is to augment the knowledge graph with their groundings [126] before learning the embedding e.g., for every human $x$ in the knowledge base, adding chordate($x$) to the knowledge base. Demeester et al. [31] address the computational inefficiency of this approach through lifted rule injection. However, in addition to being inefficient, the resulting model does not guarantee the subsumption in the completed knowledge graph.

**Augmentation through post-processing.** A simple approach is to augment the knowledge graph after learning the embedding using an existing method [146, 152]. That is, as a post-processing step we can modify the output of knowledge graph completion so as to satisfy the ontological constraints. The drawback of this approach is that the background knowledge does not help learn a better representation.
3.2. Related Work

**Regularized embeddings.** Rocktäschel et al. [120] regularize the learned embeddings using first-order logic rules. In their work, every logic rule is grounded based on observations, and a differentiable term is added to the loss function for every grounding. For example, grounding the rule $\forall x : \text{human}(x) \rightarrow \text{animal}(x)$ would result in a very large number of loss terms to be added to the loss function in a large knowledge graph. This method as well as other approaches in this category (e.g., Rocktäschel et al. [119]; Wang et al. [146]; Wang and Cohen [149]) do not scale beyond a few entities and rules, because of the very large number of regularization terms added to the loss function [31]. Guo et al. [54] proposed a method for incorporating entailment into ComplEx called RUGE which models rules based on t-norm fuzzy logic, which imposes an independence assumption over the atoms. Such an independence assumption is not necessarily true, especially in the case of subsumption, e.g., in $\text{human}(x) \rightarrow \text{animal}(x)$ for which the left and the right part of the subsumption are dependent. In addition to being inefficient, the resulting model of the regularized embedding approaches does not guarantee the subsumption in the completed knowledge graph.

**Constrained matrix factorization.** Rocktäschel et al. [120] and Demeester et al. [31] incorporate background ontologies into the embeddings learned by matrix factorization. While these methods address the problems of the two categories above, they are inadequate due to the use of matrix factorization. Application of matrix factorization for knowledge graph completion [118] learns a distinct embedding for each head-tail combination. In addition to its prohibitive memory requirement, since entities do not have their own embeddings, some regularities in the knowledge graph are ignored; for example, this representation is oblivious to the fact that $r_k(h_i, t_j)$ and $r_m(h_l, t_j)$ share the same tail.

**Constrained translation-based methods.** In translation-based methods, the relation between two entities is represented using an affine transformation, often in the form of translation. KALE [53] is a model in this category that constrains the representation to accommodate logical rules, albeit after costly propositionalization. Wang et al. [147] and Kazemi and Poole [76] show that a variety of existing translation-based methods are not fully expressive, putting a severe limitation on the kinds of knowledge graphs that can be modeled using translation-based approaches.
Region-based representation. Gutiérrez-Basulto and Schockaert [55] propose representing relations as convex regions in a $2k$-dimensional space, where $k$ is the length of the entity embeddings. A relation between two embeddings is deemed true if the corresponding point is in the convex region of the relation. Although this framework allows Gutiérrez-Basulto and Schockaert [55] to incorporate a subset of existential rules by restricting the convex regions of relations, they did not propose a practical method for learning, and their method is restricted to a subset of existential rules.

Proposed method. The contribution of this work is to prove that some of the well-known existing embedding-based models for knowledge graph completion that are fully expressive cannot reflect taxonomic information in the form of subclass and subproperty for all legal entity embeddings. We then propose a knowledge graph embedding model that is provably fully expressive and can represent such taxonomic information in the learned embeddings.

3.3 Background and Notation

It is common to have structure over the symbols used in the triples [see e.g., 128]. The Web Ontology Language (OWL) [60] defines (among many other meta-relations) subproperties and subclasses, where $p_1$ is a subproperty of $p_2$ if $\forall x, y : p_1(x, y) \rightarrow p_2(x, y)$, that is whenever $p_1$ is true, $p_2$ is also true. Classes can be defined either as a set with a class assertion (often called “type”) between an entity and a class, e.g., saying $x$ is in class $C$ using $type(x, C)$ or in terms of the characteristic function of the class, a function that is true of the element of the class. If $c$ is the characteristic function of class $C$, then $x$ is in class $c$ is written $c(x, true)$. For representations that treat entities and properties symmetrically, the two ways to define classes are essentially the same. $C_1$ is a subclass of $C_2$ if every entity in class $C_1$ is in class $C_2$, that is, $\forall x : type(x, C_1) \rightarrow type(x, C_2)$ or $\forall x : c_1(x, true) \rightarrow c_2(x, true)$. If we treat $true$ as an entity, then subclass can be seen as a special case of subproperty. For the rest of the chapter, we will refer to subsumption in terms of subproperty (and so also of subclass).

Trivial subsumption is when $p_1$ and $p_2$ are both subproperty of each other i.e., $p_1$ and $p_2$ are the same. A non-trivial subsumption is one which is not symmetric; $p_1$ is a subproperty of $p_2$ and there exists at least one pair of entities $(x, y)$ that is true of $p_2$ and is not true of $p_1$. Let $E^*$ be the set of entities when an adversary chooses legal embeddings for the entities
3.4 Tensor Factorization and Taxonomies

In this section, we study some of the well-known existing models for knowledge graph completion and their ability to enforce taxonomies. Proofs are in Appendix A starting at Page 117.

ComplEx. ComplEx \cite{140} is a knowledge graph embedding model and is explained in Section 2.3.2. The following theorem proves that we cannot use ComplEx to enforce prior knowledge about taxonomies.

**Theorem 3.1** ComplEx cannot enforce non-trivial subsumptions.

The intuition behind the proof is to start with a non-trivial subsumption and a pair of entities that the subsumption hold for the pair and another entity with a specific embedding that causes a contradiction to the subsumption assumed.

Ding et al. \cite{33} proposed a method which they call ComplEx-NNE+AER to incorporate a weaker notion of subsumption in ComplEx. For a subsumption \( \forall h, t \in \mathcal{E}^* : r(h, t) \rightarrow s(h, t) \), they suggest adding soft constraints to the loss function to encourage \( \text{Re}(r) \leq \text{Re}(s) \) and \( \text{Im}(r) = \text{Im}(s) \) (Equation 4 in \cite{33}). When the constraints are satisfied, ComplEx-NNE+AER ensures \( \forall h, t \in \mathcal{E} : \phi_\theta(r(h, t)) \leq \phi_\theta(s(h, t)) \). This is a weaker notion than the definition in Section 3.3 which requires \( \forall h, t \in \mathcal{E}^* : \phi_\theta(r(h, t)) \leq \phi_\theta(s(h, t)) \) (that is, \( \mathcal{E}^* \) is replaced with \( \mathcal{E} \)).

**Theorem 3.2** ComplEx-NNE+AER cannot satisfy its constraints and be fully expressive if symmetry constraints are allowed.

SimpLE. SimpLE \cite{76} achieves one of the state-of-the-art results in knowledge graph completion and is explained in Section 2.3.2. The following theorem shows the limitation of SimpLE when it comes to enforcing subsumption.

**Theorem 3.3** SimpLE cannot enforce non-trivial subsumptions.

The intuition behind the proof for this theorem is similar to that of Theorem 3.1.
3.5 Proposed Variation: SimplE+

Neural Network Models for Knowledge Graphs. The neural network models [e.g., 34, 124, 132] are very flexible, and so without explicit mechanisms to enforce subsumption, they cannot be guaranteed to obey any subsumption knowledge.

3.5 Proposed Variation: SimplE+

In this section, we propose a slight modification on SimplE so that the resulting method can enforce subsumption. The modification is restricting entity embeddings to be non-negative – that is \( \forall e \in \mathcal{E} : e^+, e^- \geq 0 \), where the inequality is element-wise. Next, we show that the resulting model is fully expressive and is able to enforce subsumption.

**Theorem 3.4 (Expressivity)** For any truth assignment over entities \( \mathcal{E} \) and relations \( \mathcal{R} \) containing \( |\tau'| \) true triples, there exists a SimplE+ model with embeddings vectors of size \( \min(|\mathcal{E}| \times |\mathcal{R}| + 1, |\tau'| + 1) \) that represent the assignment.

**Theorem 3.5 (Subsumption)** SimplE+ guarantees subsumption using inequality constraints.

**Objective function and training** Given the function \( \phi_\theta \), that maps a triple to the probability or score of the triple being true, ideally we would like to minimize the following regularized negative log-likelihood function:

\[
L(\theta, \tau'_\text{train}) = - \sum_{x' \in \tau'_\text{train}} \left( \log(\phi_\theta(x')) - \sum_{x \in T_{neg}(x')} \log(1 - \phi_\theta(x)) \right) + \Omega(\theta)
\]

Where \( \theta \) corresponds to all parameters of a SimplE+ model (entity and relation embeddings) and \( \Omega(\theta) \) is a regularization term. We use L2-regularization in our experiments. Here, \( T_{neg}(x) \) generates a set of negative samples corresponding to \( x \) by randomly “corrupting” the head or tail of the triple – i.e., replacing it with a random entity (introduced in Section 2.3.1).

**Enforcing the subsumptions.** In order to enforce \( \forall h, t \in \mathcal{E}^* : r(h, t) \rightarrow s(h, t) \), we should have \( \phi_\theta(s(h, t)) \geq \phi_\theta(r(h, t)) \). For that, we add an equality constraint \( r = s - \delta_r \), where \( \delta_r \) is a non-negative vector that specifies how \( r \) differs from \( s \). We learn \( \delta_r \) for all relations \( r \) which are in such a
subsumption. This equality constraint guarantees the inequality constraint of $\phi_{\theta}(s(h, t)) \geq \phi_{\theta}(r(h, t))$.

**Enforcing non-negativity.** To enforce the element-wise non-negativity of a vector, such as $\delta_r$ (a non-negative vector for relation $r$) or $h$ (a non-negative vector for entity $h$), we apply a non-linearity function with non-negative output to its elements. Here, we experiment with three different choices of non-linearity $\sigma$ defined on every element of the vector as follows. I) exponential $\sigma(x) = e^x$; II) logistic $\sigma(x) = (1 + e^{-x})^{-1}$; and III) rectified linear unit (ReLU) $\sigma(x) = \max(x, 0)$. The exponential function is always positive, so it ensures that the output of the non-linearity is always non-negative. The logistic function is a sigmoid function, which means that it has an S-shaped curve. This means that the output of the non-linearity will be close to zero for negative inputs, and it will be close to one for positive inputs. The ReLU function is a piecewise linear function, which means that it is linear for positive inputs and zero for negative inputs. We experiment with these three different choices of non-linearity to see which one performs best on our task.

### 3.6 Experimental Results

The objective of our empirical evaluations is two-fold: First, we investigate the practical implication of non-negativity constraints in terms of the effectiveness of training and the quality of final results. Second, we evaluate the practical benefit of incorporating prior knowledge in the form of subsumptions in sparse data regimes.

**Datasets.** We conducted experiments on the standard benchmarks WN18, FB15K, SPORT and LOCATION. WN18 is a subset of WordNet [99] and FB15K is a subset of Freebase [14]. SPORT and LOCATION datasets are introduced by Wang et al. [146], who created them using NELL [100]. The subsumptions in SPORT and LOCATION datasets are listed in Table 3.1. Table 3.2 gives a summary of these datasets. For evaluation on WN18 and FB15K, we split the existing triples into the same train, validation, and test sets using the same split as Bordes et al. [15].
3.6. Experimental Results

Table 3.1: We conduct experiments on SPORT and LOCATION datasets. These datasets are subsets of NELL dataset [100] and are introduced by Wang et al. [146]. The subsumptions in SPORT and LOCATION datasets are listed in this table.

<table>
<thead>
<tr>
<th>Subsumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SPORT</strong></td>
</tr>
<tr>
<td>( \text{AthleteLedSportsTeam}(x, y) \rightarrow \text{AthletePlaysForTeam}(x, y) )</td>
</tr>
<tr>
<td>( \text{AthletePlaysForTeam}(x, y) \rightarrow \text{PersonBelongsToOrganization}(x, y) )</td>
</tr>
<tr>
<td>( \text{CoachesTeam}(x, y) \rightarrow \text{PersonBelongsToOrganization}(x, y) )</td>
</tr>
<tr>
<td>( \text{OrganizationHiredPerson}(x, y) \rightarrow \text{PersonBelongsToOrganization}(x, y) )</td>
</tr>
<tr>
<td>( \text{PersonBelongsToOrganization}(x, y) \rightarrow \text{OrganizationHiredPerson}(y, x) )</td>
</tr>
<tr>
<td><strong>LOCATION</strong></td>
</tr>
<tr>
<td>( \text{CapitalCityOfCountry}(x, y) \rightarrow \text{CityLocatedInCountry}(x, y) )</td>
</tr>
<tr>
<td>( \text{StateHasCapital}(x, y) \rightarrow \text{CityLocatedInState}(y, x) )</td>
</tr>
</tbody>
</table>

Table 3.2: Statistics of the datasets used in the experiments. In this table, \(|\mathcal{E}|\) represents the number of entities, \(|\mathcal{R}|\) represents the number of relations in the dataset, and \(#\text{train}\), \(#\text{valid}\), and \(#\text{test}\) denote number of triples in the train, validation, and test sets respectively.

| Dataset | \(|\mathcal{E}|\) | \(|\mathcal{R}|\) | \(#\text{train}\) | \(#\text{valid}\) | \(#\text{test}\) |
|---------|----------------|----------------|-------------------|-------------------|-----------------|
| WN18    | 40,943         | 18             | 141,442           | 5,000             | 5,000           |
| FB15K   | 14,951         | 1,345          | 483,142           | 50,000            | 59,071          |
| SPORT   | 1039           | 5              | 1312              | -                 | 307             |
| LOCATION| 445            | 5              | 384               | -                 | 100             |
3.6. Experimental Results

Table 3.3: Results of SimplE+ with different non-linearity functions in producing non-negative embeddings on wn18 dataset without incorporating subsumptions. ReLU outperforms other choices, and therefore moving forward we use ReLU for non-negativity constraints in SimplE+.

<table>
<thead>
<tr>
<th>Function σ</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@3</th>
<th>HIT@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimplE+-Exponential</td>
<td>0.866</td>
<td>0.829</td>
<td>0.925</td>
<td>0.897</td>
</tr>
<tr>
<td>SimplE+-Logistic</td>
<td>0.854</td>
<td>0.836</td>
<td>0.863</td>
<td>0.885</td>
</tr>
<tr>
<td>SimplE+-ReLU</td>
<td>0.937</td>
<td>0.936</td>
<td>0.938</td>
<td>0.939</td>
</tr>
</tbody>
</table>

Table 3.4: Results of SimplE+ with different non-linearity functions in producing non-negative embeddings on fb15k dataset without incorporating subsumptions. ReLU outperforms other choices, and therefore moving forward we use ReLU for non-negativity constraints in SimplE+.

<table>
<thead>
<tr>
<th>Function σ</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@3</th>
<th>HIT@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimplE+-Exponential</td>
<td>0.575</td>
<td>0.468</td>
<td>0.640</td>
<td>0.773</td>
</tr>
<tr>
<td>SimplE+-Logistic</td>
<td>0.425</td>
<td>0.294</td>
<td>0.491</td>
<td>0.694</td>
</tr>
<tr>
<td>SimplE+-ReLU</td>
<td>0.725</td>
<td>0.658</td>
<td>0.770</td>
<td>0.841</td>
</tr>
</tbody>
</table>

3.6.1 Effect of Non-negativity Constraints on Entity Embeddings

Non-negativity has been a subject studied in various research fields. In many NLP-related tasks, non-negativity constraints are studied to learn more interpretable representations for words [101]. In matrix factorization, non-negativity constraints are used to produce more coherent and independent factors [87]. To get non-negativity constraint on the embedding of entities, we simply apply an element-wise non-linearity $\sigma : \mathbb{R} \rightarrow \mathbb{R}^{\geq 0}$ before evaluation.

Tables 3.3 and 3.4 shows the result of SimplE+ with for different choices of $\sigma$: exponential, logistic, and ReLU (defined in Section 3.5). ReLU outperforms other choices, and therefore moving forward we use ReLU for non-negativity constraints.
3.6. Experimental Results

Table 3.5: Results on wn18 for SimplE and SimplE+ without incorporating subsumptions. As the results indicate, the non-negativity constraint on entity embeddings deteriorates the model’s performance but allows us to be able to guarantee enforcing subsumptions (as shown in Theorem 3.5).

<table>
<thead>
<tr>
<th>Model</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@3</th>
<th>HIT@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplEx</td>
<td>0.941</td>
<td>0.936</td>
<td><strong>0.945</strong></td>
<td><strong>0.947</strong></td>
</tr>
<tr>
<td>SimplE</td>
<td><strong>0.942</strong></td>
<td><strong>0.939</strong></td>
<td>0.944</td>
<td><strong>0.947</strong></td>
</tr>
<tr>
<td>SimplE+</td>
<td>0.937</td>
<td>0.936</td>
<td>0.938</td>
<td>0.939</td>
</tr>
</tbody>
</table>

Table 3.6: Results on fb15k for SimplE and SimplE+ without incorporating subsumptions. As the results indicate, the non-negativity constraint on entity embeddings deteriorates the model’s performance but allows us to be able to guarantee enforcing subsumptions (as shown in Theorem 3.5).

<table>
<thead>
<tr>
<th>Model</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@3</th>
<th>HIT@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplEx</td>
<td>0.692</td>
<td>0.599</td>
<td>0.759</td>
<td>0.840</td>
</tr>
<tr>
<td>SimplE</td>
<td><strong>0.727</strong></td>
<td><strong>0.660</strong></td>
<td><strong>0.773</strong></td>
<td>0.838</td>
</tr>
<tr>
<td>SimplE+</td>
<td>0.725</td>
<td>0.658</td>
<td>0.770</td>
<td><strong>0.841</strong></td>
</tr>
</tbody>
</table>

Next, we evaluate the effect of the non-negativity constraint on the performance of the algorithm. Tables 3.5 and 3.6 show our result on wn18 and fb15k datasets. Note that this is effectively comparing SimplE+ with SimplE and ComplEx, without accommodating any subsumptions. As the results indicate, this constraint deteriorates the model’s performance in most cases except for HIT@10 on fb15k. As adding this constraint allow us to do more with guaranteeing to enforce subsumption (as shown in Theorem 3.5), we continue to use it in our experiments.

3.6.2 Sparse Relations

In this section, we study the scenario of learning relations that appear in a few triples in the knowledge graph. In particular, we observe the behavior of various methods as the number of training triples varies. We train SimplE,
3.6. Experimental Results

Figure 3.1: HIT@1 of SimplE, SimplE+, and logical inference when they are trained on different fractions (percentages) of the training data on SPORT dataset. The results indicate that when training data is scarce, logical inference outperforms (or is on-par with) SimplE, as SimplE does not see enough triples to be able to learn meaningful embeddings. As the amount of training data increases, SimplE starts to outperform logical inference as it can better generalize to unseen cases than pure logical inference. The gap between these two methods becomes larger as the amount of training data increases. For all tested fractions, SimplE+ outperforms both SimplE and logical inference as it uses both the generalization power of SimplE and the inference power of logical rules.

In order to test the effect of incorporating taxonomical information on SimplE+, and logical inference on fractions of the SPORT training set and test them on the full test set. Logical inference refers to inferring new triples based only on the subsumptions and training triples.

Figure 3.1 shows the HIT@1 of the three methods when they are trained on different fractions (percentages) of the training data. According to Figure 3.1 when training data is scarce, logical inference performs better than (or on-par with) SimplE, as SimplE does not see enough triples to be able to learn meaningful embeddings. As the amount of training data increases, SimplE starts to outperform logical inference as it can better generalize to unseen cases than pure logical inference. The gap between these two methods becomes larger as the amount of training data increases. For all tested fractions, SimplE+ outperforms both SimplE and logical inference as it uses both the generalization power of SimplE and the inference power of logical rules.

In order to test the effect of incorporating taxonomical information on
3.7. Limitations

In order to test the number of epochs required for training to converge, we tested SimplE and SimplE+ on the Sport dataset with the same set of parameters and the same initialization and plotted the loss function for each epoch. The plot in Figure 3.2 shows that SimplE+ requires fewer epochs than SimplE to converge.

3.6.3 Knowledge Graphs with no Redundant Triples

Tensor factorization techniques rely on large amounts of annotated data. When background knowledge is available, we might expect a knowledge graph to not include redundant information. For instance, if we have the following triple in a knowledge graph:

\[
\text{CapitalCityOfCountry}(\text{Paris}, \text{France})
\]

and the following rule is specified:

\[
\forall h, t \in \mathcal{E}^* : \text{CapitalCityOfCountry}(h, t) \rightarrow \text{CityLocatedInCountry}(h, t)
\]

Given this, the triple \(\text{CityLocatedInCountry}(\text{Paris}, \text{France})\) is redundant. Similar to the experiment for incorporating background knowledge in [76], we remove all redundant triples from the training set and compare SimplE with SimplE+ and logical inference. The obtained results in Tables 3.7 and 3.8 demonstrate that SimplE+ outperforms SimplE and logical inference on both Sport and Location datasets with a large margin. As an example, SimplE+ gains \(-90\%\) and \(-230\%\) improvement over SimplE in terms of Hit@1 for Sport and Location datasets respectively. These results represent the clear advantage of SimplE+ over SimplE when background taxonomic information is available.

3.7 Limitations

Our proposed model can incorporate subproperties into embeddings. However, it is not able to incorporate all ontological information. For instance, it is not able to incorporate the following rules:

- Range information: range rules are in the form of \(\forall h, t \in \mathcal{E}^* : r(h, t) \rightarrow c_1(t, true)\) which enforces type \(c_1\) for tail of relation \(r\), e.g., if we know Paris is located in France, we can infer France is a country.
3.8. Conclusion

Figure 3.2: This experiment shows the loss value for SimplE and SimplE+ at each training epoch on sports dataset. Models are tested with the exact number of parameters and initialization. The results show that SimplE+ requires fewer epochs than SimplE to converge.

- Domain information: domain rules are in the form of $\forall h, t \in \mathcal{E}^*: r(h, t) \rightarrow c_2(h, true)$ which enforces type $c_2$ for head of relation $r$, e.g., if we know Paris is located in France, we can infer Paris is a city.

- Transitivity: transitivity rules are in the form of $\forall h, t, z \in \mathcal{E}^*: r(h, z) \land s(z, t) \rightarrow t(h, t)$, e.g., if we know Paris is the capital of France and France is a country in Europe, we can infer that France is located in Europe.

3.8 Conclusion

In this chapter, we proposed SimplE+, a fully expressive tensor factorization model for knowledge graph completion when background taxonomic information (in terms of subclasses and subproperties) is available. We showed that existing fully expressive models cannot provably respect subclass and subproperty information. Then we proved that by adding non-negativity constraints to entity embeddings of SimplE, one of the state-of-the-art tensor factorization approaches, we can build a model that is not only fully expressive but also able to enforce subsumptions. Experimental results on benchmark knowledge graphs demonstrate that SimplE+ is simple yet effective. On our benchmarks, SimplE+ outperforms SimplE and offers a faster convergence rate when background taxonomic information is available.
3.8. Conclusion

Table 3.7: Results comparing logical inference, SimplE, and SimplE+ on SPORT dataset. The best results are in bold. MRR and HIT@k for k > 1 is not applicable for logical inference as it either predict the correct entity or not (SimplE and SimplE+ predict a ranking). The results demonstrate the effectiveness of SimplE+ in the presence of subsumptions.

<table>
<thead>
<tr>
<th>Model</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@3</th>
<th>HIT@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical inference</td>
<td>-</td>
<td>0.288</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SimplE</td>
<td>0.230</td>
<td>0.184</td>
<td>0.234</td>
<td>0.324</td>
</tr>
<tr>
<td>SimplE+</td>
<td><strong>0.404</strong></td>
<td><strong>0.349</strong></td>
<td><strong>0.440</strong></td>
<td><strong>0.508</strong></td>
</tr>
</tbody>
</table>

Table 3.8: Results comparing logical inference, SimplE, and SimplE+ on LOCATION dataset. The best results are in bold. MRR and HIT@k for k > 1 is not applicable for logical inference as it either predict the correct entity or not (SimplE and SimplE+ predict a ranking). The results demonstrate the effectiveness of SimplE+ in the presence of subsumptions.

<table>
<thead>
<tr>
<th>Model</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@3</th>
<th>HIT@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical inference</td>
<td>-</td>
<td>0.270</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SimplE</td>
<td>0.190</td>
<td>0.130</td>
<td>0.210</td>
<td>0.315</td>
</tr>
<tr>
<td>SimplE+</td>
<td><strong>0.440</strong></td>
<td><strong>0.430</strong></td>
<td><strong>0.440</strong></td>
<td><strong>0.450</strong></td>
</tr>
</tbody>
</table>
Chapter 4

Incorporating n-ary Relations

This chapter is structured in three sections. Section 4.1 discusses n-ary relations in knowledge hypergraphs and why converting them to binary relations as in knowledge graphs does not work. Section 4.2 introduces link prediction models for knowledge hypergraphs inspired by link prediction models for knowledge graphs, benchmarks, baselines, and evaluation metrics for this task. Section 4.3 explores the gap between relational algebra foundations and machine learning techniques for knowledge hypergraph completion. The results presented in this chapter have been published in Fatemi et al. [40, 42].

4.1 Introduction

Knowledge hypergraphs are knowledge bases that store information about the world in the form of tuples describing relations among entities. Knowledge graphs are a specific form of knowledge hypergraphs that represent relations between exactly two entities. Knowledge hypergraphs thus generalize knowledge graphs by allowing multi-argument relations. Knowledge hypergraphs are similar to relational databases. In relational databases, each table describes tuples about one specific relation. Each table has the number of columns equal to the arity of the relation.

Knowledge hypergraphs are incomplete because curating and storing all the true information in the world is difficult. The goal of link prediction in knowledge hypergraphs (or knowledge hypergraph completion) is to predict unknown relationships among entities based on existing ones. In this chapter, we are interested in the problem of link prediction in knowledge hypergraphs. Our motivation for studying link prediction in these more sophisticated knowledge structures is based on the fact that much knowledge in the world is in the form of relations among more than two entities.

Link prediction in knowledge graphs is studied extensively in the literature [e.g., 15, 76, 140, 158]. In these studies, knowledge graphs are defined
as directed graphs having nodes as entities and labeled edges as relations; edges are directed from the head entity to the tail entity. The common data structure for representing knowledge graphs is a set of triples of the form relation(head, tail) that represents information as a collection of binary relations.

Embedding-based models (discussed in Section 2.3.2) learn embeddings for entities and relations and to predict whether a triple relation(head, tail) is true, such models define a scoring function $\phi_\theta$ that uses embeddings of relation, head, and tail and outputs the probability of the triple being true. Such embedding-based methods make the strong assumption that all relations are binary. One might hypothesize that for predicting new links in a knowledge hypergraph, we can convert n-ary relations to binary ones using conversion techniques such as reification or star-to-clique [153] and then applying existing models for link prediction on knowledge graphs on them. In what follows, we explain the challenges that arise when coupling conversion techniques with knowledge graph completion models. We motivate our work by outlining that converting non-binary relations into binary ones and then applying known link prediction methods does not yield satisfactory results.

Reification is one common approach to converting higher-arity relations into binary ones. For each relation $r$ with arity $k$, reification forms $k$ new binary relations $r_1 \ldots r_k$. For each tuple, reification constructs an entity corresponding to the tuple. It converts a tuple $r(e_1, \ldots, e_k)$ to a set of triples $r_1(e, e_1), \ldots, r_k(e, e_k)$ with a new entity $e$.

For instance, for the relation FliesBetween with arity 3, reification forms 3 binary relations as follows: FliesBetweenAirline, FliesBetweenDeparture, and FliesBetweenArrival. Assume we want to reify the following tuple with the relation FliesBetween.

\textit{FliesBetween(AirCanada, Vancouver, NewYork)}

For reifying this tuple, reification defines a new entity \textit{e.g.}, AC548 specifically for this tuple. The tuple is then converted to the following set of triples using the binary relations corresponding to FliesBetween, entity AC548, and the existing entities in the tuple.

\textit{FliesBetweenAirline(AC548, AirCanada)}
\textit{FliesBetweenDeparture(AC548, Vancouver)}
\textit{FliesBetweenArrival(AC548, NewYork)}
4.1. Introduction

The example in Figure 4.1a shows three facts that pertain to the relation FliesBetween. When we reify the hypergraph in this example (Figure 4.1b), we add three reified entities $e_1$, $e_2$, and $e_3$ (brown square nodes). In terms of representation, the binary relations created are equivalent to the original representation. Reification is complete and does not lose information during the conversion and because of this, the reification process is reversible.

It is noteworthy to mention that FREEBASE represents the relations among entities using triples with reification. We reverse the reification process (we explain the reverse reification process in Section 4.2.4) in FREEBASE and notice that 61% of the relations are defined on more than two entities (are beyond binary). Wen et al. (2016) observe that in the original FREEBASE more than 1/3rd of the entities participate in non-binary relations (i.e., defined on more than two entities).

Another conversion approach is star-to-clique, which forms $\binom{k}{2}$ binary relations for each relation with arity $k$. It converts a tuple defined on $k$ entities into $\binom{k}{2}$ tuples with distinct relations between all pairwise entities in the tuple. For instance, for the tuple above, it forms the following 3 binary relations: FliesBetweenAirlineDeparture, FliesBetweenAirlineArrival, and FliesBetweenDepartureArrival. It converts the tuple to the following triples.

FliesBetweenAirlineDeparture (AirCanada, Vancouver)
4.2 Knowledge Hypergraph Completion: HSimplE and HypE

FliesBetweenAirlineArrival(AirCanada, NewYork)

FliesBetweenDepartureArrival(Vancouver, NewYork)

The challenge with reification arises when converting higher-arity relations to binary ones and then applying existing embedding-based methods. This is because we introduce new entities that the model never encounters during training and we do not have a learned embedding for these entities, and current embedding-based methods require an embedding for each entity to be able to make a prediction during evaluation. Applying the star-to-clique method to a hypergraph is problematic. This is because it is impossible to construct the original tuples from the triples after start-to-clique conversion. Figure 4.1d shows the result of applying star-to-clique to the original hypergraph in Figure 4.1a, in which the tuple FliesBetween(AirCanada, NewYork, LosAngeles) might be interpreted as being true since the corresponding entities are connected by edges, whereas looking at the original hypergraph, there is not a hyperedge connecting AirCanada, NewYork, and LosAngeles.

4.2 Knowledge Hypergraph Completion: HSimplE and HypE

In this section, we introduce two embedding-based models that perform link prediction directly on knowledge hypergraphs without converting them to graphs. Both proposed models are based on the idea that predicting the existence of a relation between a set of entities depends on the position of the entities in the relation; otherwise, if the positions of the entities are not incorporated when making a prediction, the model has the assumption that all relations are symmetric. Learning entity embeddings for each position independently does not work well, as it does not let the information flow between the embeddings for the different positions of the same entity. The information flow problem between different embeddings for an entity is similar to the reason why CP performs poorly and how SimplE resolves the issue - see Section 2.3.2.

The first model we propose is HSimplE. For a given entity, HSimplE shifts the entity embedding by a value that depends on the position of the entity in the given relation. Our second model is HypE, which in addition to learning entity embeddings, learns positional (convolutional) embeddings; these positional embeddings are disentangled from entity representations and are used to transform the representation of an entity based on its position in
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

a relation. This makes HypE more robust to changes in the position of an
entity within a tuple. We show that both HSimplE and HypE are fully
expressive. To evaluate our models, we introduce two new datasets from
subsets of FREEBASE, and develop baselines by extending existing models
on knowledge graphs to work with hypergraphs.

The contributions of this section are as follows.

• Showing that current techniques to convert a knowledge hypergraph
to a knowledge graph do not yield satisfactory results for the link
prediction task.

• Introducing HypE and HSimplE, two models for knowledge hyper-
graph completion.

• Introducing a set of baselines for knowledge hypergraph completion by
extending some of the knowledge graph completion models to n-ary
relations.

• Collecting two new datasets containing multi-arity relations from sub-
sets of FREEBASE, which can serve as new evaluation benchmarks for
knowledge hypergraph completion methods. We also show that our
proposed methods outperform baselines.

4.2.1 Related Work

Existing methods that relate to our work in this section can be grouped into
the following three main categories.

Knowledge graph completion. Embedding-based models for knowledge
graph completion such as translational [15, 151], bilinear [76, 158], and deep
models [132] have proved to be effective for knowledge graphs where all
relations are binary. In Section 4.2.6, we extend some of the models in this
category to knowledge hypergraphs and compare their performance with the
proposed methods.

Knowledge hypergraph completion. Soft-rule models [30, 77] can han-
dle variable arity relations and have the advantage of being interpretable.
However, they can only learn a subset of patterns [107]. Guan et al. [51]
propose an embedding-based method based on the star-to-clique approach.
The caveats of this approach are discussed earlier. m-TransH [153] extends
TransH [151] to knowledge hypergraph completion. Kazemi and Poole [76]
prove that TransH, and consequently m-TransH, are not fully expressive and have restrictions in the type of relations they can model (See Lemma 4.3.1 for a detailed discussion). In contrast, we prove that our proposed models are fully expressive.

**Learning on hypergraphs.** Hypergraph learning has been employed to model high-order correlations among data in many tasks, such as in video object segmentation [65] and modeling image relationships and image ranking [66]. There is also a line of work extending graph neural networks to hypergraph neural networks [44] and hypergraph convolution networks [156]. These models are designed for undirected hypergraphs, with edges that are not labeled (no relations), while knowledge hypergraphs are directed and labeled graphs. As there is no clear or easy way of extending these models to our knowledge hypergraph setting, we do not consider them as baselines for our experiments.

**Learning with tabular data.** Tabular data is similar to the data stored in knowledge hypergraphs. There are several models proposed to work with tabular data. Some are based on variants of ensemble decision trees. Among ensembling methods, random forests [61] use randomly selected features from random subsets of data to grow many trees. XGBoost [22] and LightGBM [80] are the two ensemble decision tree approaches that dominate most of the recent data science competitions. Some more recent models for tabular data are based on deep learning approaches [5]. These models are designed for classification tasks on tabular data.

### 4.2.2 Proposed Methods

Our proposed methods explore different approaches to how a prediction can be affected by the embeddings of the relation and entities in the tuple as well as the role of the entities in the tuple. For instance, when making predictions for \texttt{Sold}(Drew, Alex, Book) indicating whether Drew sold Alex a Book or making predictions for \texttt{Sold}(Alex, Drew, Book) indicating whether Alex sold Drew a Book, the entity embeddings for Drew and Alex and the relation embedding for Sold should be used as well as the positions of Drew and Alex in the tuples. This is because the first position in the \texttt{Sold} relation corresponds to the seller and the second position corresponds to the buyer (and the third position contains the item). As another example, in Figure 4.1a, Montreal is the departure city; but it may appear in a different position (e.g., arrival city) in another tuple. This means that the way we
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

Figure 4.2: Visualization of HSimplE and HypE architectures. (a) $\phi$ for HSimplE transforms entity embeddings by shifting them based on their position and combining them with the relation embedding. (b) $f(e, i)$ for HypE takes an entity embedding and the position the entity appears in the given tuple, and returns a vector. (c) $\phi$ takes as input a tuple and outputs the score of HypE for the tuple.

use Montreal’s embedding for making predictions may need to vary based on the position it appears in within the tuple.

When using the embedding of an entity to make a prediction, if the prediction does not depend on the positions of the entities in the tuple, then the relation has to be symmetric (which is not the case for most relations e.g., Sold relation). DistMult \[158\] (discussed in Section 2.3.2) does not distinguish among an entity appearing in different positions in its scoring function and can only model symmetric relations.

On the other hand, if for each entity, we learn multiple independent entity embeddings, each corresponding to when the entity appears in that specific position, information about one position will not interact with that of others. For instance, canonical polyadic \[59\] has this information flow problem as it learns multiple independent embeddings for cases when an entity appears in different positions (discussed in Section 2.3.2). It should be noted that in several embedding-based methods for knowledge graph completion, such as ComplEx \[140\] and SimplE \[76\], the prediction depends on the position of each entity in the tuple.

In what follows, we propose two embedding-based methods for link prediction in knowledge hypergraphs. The first model is inspired by SimplE and has its roots in knowledge graph completion; the second model takes a fresh look at knowledge completion as a multi-arity problem, without first setting it up within the frame of binary relation prediction.
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

HSimplE

HSimplE is an embedding-based method for link prediction in knowledge hypergraphs that is inspired by SimplE \cite{76} (see Section 2.3.2 for a detailed explanation of SimplE). SimplE learns two embedding vectors $e^+$ and $e^-$ for an entity $e$ (one for each possible position of the entity), and two embedding vectors $r^+$ and $r^-$ for a relation $r$ (with one relation embedding as the inverse of the other). SimplE computes the score of a triple as follows with the $\odot()$ function computing the sum of the element-wise product of its input vectors.

$$
\phi_{\theta}(r(e_i, e_j)) = \odot(r^+, e_i^+, e_j^-) + \odot(r^-, e_j^+, e_i^-)
$$

Equation 4.1

Here, we provide another view of the same scoring function Equation 4.1. This view helps us extend SimplE to higher arity relations. Function concat returns the concatenation of its input vectors. $\text{len}(e)$ returns the length of vector $e$. cirshift($v$, $x$) circularly shifts the elements in vector $v$ to the left by $x$ steps. Formally, for a vector $v \in \mathbb{R}^n$, indexed $0 \ldots n - 1$, a circular shift of size $x$ is a permutation $\pi$ of the $n$ entries in the tuple such that the following holds for indices in the output of the circular shift.

$$
\forall i \ 0 \leq i \leq n - 1: \pi(i) = (i + x) \mod n
$$

As an example, for $v = [5, 2, 4, 1]$ and $x = 2$, cirshift($v$, $x$) is $[4, 1, 5, 2]$. The following equation shows a different view of the same scoring function of SimplE as in Equation 4.1. Let $e_1$ be concat($e^+_1, e^-_1$), $e_2$ be concat($e^+_2, e^-_2$), and $r$ be concat($r^+, r^-$).

$$
\phi_{\theta}(r(e_i, e_j)) = \odot(r, e_i, \text{cirshift}(e_j, \text{len}(e_2)/2))
$$

Equation 4.2

Equation 4.1 and Equation 4.2 are equal, because in Equation 4.2, we circularly shift $e_2$ to align its indices with that of $e_1$ to get the same output as in Equation 4.1.

In HSimplE, we adopt the idea of having different representations for an entity based on its position in a relation and updating all these representations from a single training tuple. We do this by representing each entity $e$ as a single vector $e$ (instead of multiple vectors as in SimplE), and each relation $r$ as a single vector $r$. Conceptually, each $e$ can be seen as the concatenation of the different representations of $e$ based on every possible position. For example, in a knowledge hypergraph where the relation with maximum arity is $\alpha$, an entity can appear in $\alpha$ different positions; hence
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

**HSimplE**

HSimplE will be the concatenation of $\alpha$ vectors, one for each possible position. HSimplE scores a tuple using the following function.

$$
\phi_\theta(r(e_i, e_j, \ldots, e_k)) = \odot(r, e_i, \text{shift}(e_j, \text{len}(e_j)/\alpha), \ldots, \text{shift}(e_k, \text{len}(e_k) \cdot (\alpha - 1)/\alpha)) \quad (4.3)
$$

Here, $\alpha = \max_r \epsilon \mathbb{R}(|r|)$ is the maximum arity of the relations in the knowledge hypergraph. We observe that for knowledge graphs ($\alpha = 2$), Equation (4.3) reduces to Equation (4.2) showing that SimplE is a special instance of HSimplE. The architecture of HSimplE is summarized in Figure 4.2a.

**HypE**

HypE learns an embedding for each entity and relation. When an entity appears in a specific position in a tuple, HSimplE circularly shifts the embedding to project it to the latent space of the position. HypE, in addition to learning entity embeddings, learns positional (convolutional) embeddings; these positional embeddings are disentangled from entity representations and are used to transform the representation of an entity based on its position in a relation. This makes HypE more robust to changes in the position of an entity within a tuple.

HypE learns a single representation for each entity, a single representation for each relation, and positional convolutional weight filters for each possible position. When an entity appears in a specific position, the appropriate positional filters are first used to transform the embedding of each entity in the given fact; these transformed entity embeddings are then combined with the embedding of the relation to producing a score, i.e., the probability of the input tuple to be true. The architecture of HypE is summarized in Figures 4.2b and 4.2c.

Let $n$ denote the number of filters per position, $l$ the filter length, $d$ the embedding dimension, and $s$ the stride of a convolution. Let $W_i \in \mathbb{R}^{n \times l}$ be the convolutional filters associated with position $i$, and let $W_i[j] \in \mathbb{R}^l$ be the $j$-th row of $W_i$. We denote by $P \in \mathbb{R}^{nq \times d}$ the projection matrix, where $q = \lfloor (d - l)/s \rfloor + 1$ is the feature map size. For a given tuple with an entity $e$ appearing in the $i$-th position, we use the following function as defined below to map the embedding vector $e$ to $\hat{e} \in \mathbb{R}^{nq}$.

$$
\hat{e} = \text{concat}(e \ast W_i[1], \ldots, e \ast W_i[n])
$$
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

Function $f$ is used afterward as multiplication of the vector $\hat{e}$ and matrix $P$. It returns a vector of size $d$.

$$f(e, i) = \hat{e}P$$

Thus, each entity embedding $e$ appearing at position $i$ in a given tuple is convolved with the set of position-specific filters $W_i$ to give $n$ feature maps of size $q$. All $n$ feature maps corresponding to an entity are concatenated into a vector of size $nq$ and projected to the embedding space through multiplication by $P$. The projected vectors of entities and the embedding of the relation are combined by inner-product to define $\phi$:

$$\phi_\theta(r(e_1, \ldots, e_{|r|})) = \odot(r_f(e_1, 1), \ldots, f(e_{|r|}, |r|)). \quad (4.4)$$

The advantage of learning positional filters disentangled from entity embeddings is two folds: On one hand, learning a single vector per entity keeps entity representations simple and disentangled from its position in a given fact. On the other hand, unlike HSimplE, HypE learns positional filters from all entities that appear in the given position; overall, this separation of representations for entities, relations, and positions facilitates the representation of knowledge bases having facts with an arbitrary number of entities. It also gives HypE additional robustness, such as in the case when we test a trained HypE model on a tuple that contains an entity in a position never seen before at train time. We discuss this in Section 4.2.6.

Both HSimplE and HypE are fully expressive — an important property that has been the focus of several studies [39]. A model that is not fully expressive can embed assumptions that may not be reflected in reality.

**Theorem 4.1 (Expressivity)** For any ground truth over entities $E$ and relations $R$ containing $|\tau|$ true tuples and $\alpha = \max_{r \in R}(|r|)$, there exists a HypE and a HSimplE model with embedding vectors of size $\max(\alpha|\tau|, \alpha)$ that represents that ground truth.

The proof is in Appendix B.1.

4.2.3 Objective Function and Training

Both HSimplE and HypE are trained using stochastic gradient descent with mini-batches. As there are only positive instances available and negative instances are also needed for training the model, we generate negative examples. For negative sample generation, we follow the contrastive approach.
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

of [15] for knowledge graphs and extend it to knowledge hypergraphs: for each tuple, we produce a set of negative samples of size $N|\tau|$ by replacing each of the entities with $N$ random entities in the tuple, one at a time. Here, $N$ is the ratio of negative samples in our training set and is a hyperparameter. For training both models, we define the following cross-entropy loss:

$$L(\theta, \tau'_{\text{train}}) = \sum_{x' \in \tau'_{\text{train}}} - \log \left( \frac{e^{\phi_\theta(x')}}{e^{\phi_\theta(x')}} + \sum_{x \in T_{\text{neg}}(x')} e^{\phi_\theta(x)} \right) \quad (4.5)$$

4.2.4 Datasets

We conduct experiments on a total of 5 datasets. For the experiments on datasets with binary relations, we use two standard benchmarks for knowledge graph completion: \textsc{wn18} [16] and \textsc{fb15k} [15] (introduced in Section 3.6). The experiments on knowledge hypergraph completion are conducted on three datasets. The first is \textsc{jf17k} proposed by [Wen et al. 2016] which contain train and test sets; as no validation set is proposed for \textsc{jf17k}, we randomly select 20% of the train set as validation. We also create two datasets \textsc{fb-auto} and \textsc{m-fb15k} from \textsc{freebase}. In what follows, we explain how these two datasets are created.

Dataset Creation. We downloaded \textsc{freebase} [14] from [1]. Note that \textsc{freebase} uses reification to represent relations with triples; Relations in \textsc{freebase} have three parts separated with a ‘.’ e.g., the following is a relation in \textsc{freebase}: \texttt{film.performance.actor}. The first and second parts show the relation name and the third part shows the role in the relation.

Here are three examples of binary relations in \textsc{freebase}.

* \texttt{film.performance.actor(/m/04q4bd8, /m/07yyhx)}

* \texttt{film.performance.character(/m/04q4bd8, /m/0qzpnz5)}

* \texttt{film.performance.film(/m/04q4bd8, /m/04mxv5p)}

From the [Knowledge Graph Search API link], the entities /m/07yyhx, /m/0qzpnz5, and /m/04mxv5p refer to actress “Kathlyn Williams”, character “Queen Isabel of Bourbon”, and movie “The Spanish Dancer” respectively. The entity /m/04q4bd8 is a reified entity.

To obtain a knowledge hypergraph $H$ from \textsc{freebase}, we perform an inverse reification process by following the steps below.
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

1. From Freebase, we start with all triples without numbers or enumerations. This is because our focus is on entities. Handling numbers or enumerations is a different area and is considered a future work.

2. Extract the relation names from the first two parts of the relations and the mapping from roles to positions. For instance, \texttt{film.performance} is a relation name with its first argument as an actor, the second argument as a character, and the third argument as a film.

3. Join the triples in Freebase that have the same relation name and the same entity in the first position (the first entity refers to the reified entity). The role (the third part of the relation) identifies the position in the resulting relation. For example, we convert the three tuples above to a tuple with a relation of arity 3 as:

\begin{align*}
\text{film.performance}(/m/07yyhx, /m/0qzpnz5, /m/04mxv5p)
\end{align*}

with the first, second, and third positions representing the actor, character, and movie of the performance respectively in $H$.

4. Create the \textsc{fb-auto} dataset by selecting the facts from $H$ whose subject is ‘automotive’ (the first part of relation names is ‘automative’. The ‘automotive’ is selected to get a dataset with different statistics compared to the two other datasets).

5. Create the \textsc{m-fb15k} dataset by following a strategy similar to that proposed by [15]: select the facts in $H$ that pertain to entities present in the Wikilinks database [129].

6. Split the facts in each of \textsc{fb-auto} and \textsc{m-fb15k} randomly into train, test, and validation sets.

See Tables 4.1 and 4.2 for statistics of the datasets.

4.2.5 Baselines

To compare our results to that of existing work, we first design simple baselines that extend current models to work with knowledge hypergraphs. We only consider models that admit a simple extension to higher-binary relations for the link prediction task. The baselines for this task are grouped into the following categories: (1) methods that work with binary relations and that are easily extendable to higher-arity, namely r-SimplE, m-DistMult, and m-CP; (2) existing methods that can handle higher-arity relations, namely m-TransH. Below we give some details about methods in category (1).
### 4.2. Knowledge Hypergraph Completion: HSimplE and HypE

Table 4.1: Statistics of the datasets used in the experiments. In this table, $|\mathcal{E}|$ represents the number of entities, $|\mathcal{R}|$ represents the number of relations in the dataset, and $\#\text{train}$, $\#\text{valid}$, and $\#\text{test}$ denote number of triples in the train, validation, and test sets respectively.

| Dataset   | $|\mathcal{E}|$ | $|\mathcal{R}|$ | $\#\text{train}$ | $\#\text{valid}$ | $\#\text{test}$ |
|-----------|----------------|----------------|------------------|------------------|----------------|
| JF17K     | 29,177         | 327            | 77,733           | –                | 24,915         |
| FB-AUTO   | 3,410          | 8              | 6,778            | 2,255            | 2,180          |
| M-FB15K   | 10,314         | 71             | 415,375          | 39,348           | 38,797         |

Table 4.2: Number of tuples with different arities in different datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>arity=2</th>
<th>arity=3</th>
<th>arity=4</th>
<th>arity=5</th>
<th>arity=6</th>
</tr>
</thead>
<tbody>
<tr>
<td>JF17K</td>
<td>56,322</td>
<td>34,550</td>
<td>9,509</td>
<td>2,230</td>
<td>37</td>
</tr>
<tr>
<td>FB-AUTO</td>
<td>3,786</td>
<td>0</td>
<td>215</td>
<td>7,212</td>
<td>0</td>
</tr>
<tr>
<td>M-FB15K</td>
<td>82,247</td>
<td>400,027</td>
<td>26</td>
<td>11,220</td>
<td>0</td>
</tr>
</tbody>
</table>

**r-SimplE.** To test the performance of a model trained on reified data, we convert higher-arity relations in the train set to binary relations through reification. We then use SimplE (that we call r-SimplE) on this reified data. In this setting, at test time higher-arity relations are first reified to a set of binary relations; this process creates new auxiliary entities for which the model has no learned embeddings. To embed the auxiliary entities for the prediction step, we use the observation we have about them at test time. For example, a relation with arity greater than two $r(e_1, e_2, e_3)$ is reified at test time by adding a new entity $e'$ and converting the higher-arity tuple to three binary facts: $r_1(e', e_1)$, $r_2(e', e_2)$, and $r_3(e', e_3)$. When predicting the tail entity of $r_1(e', ?)$, we use the other two reified facts to learn an embedding for entity $e'$. Because $e'$ is added only to help represent the higher-arity relations as a set of binary relations, we only need to do tail prediction for reified relations. We do not reify binary relations.

**m-DistMult.** DistMult [158] defines a score function as follows.

$$\phi_\theta(r(e_i, e_j)) = \odot(r, e_i, e_j)$$
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

Table 4.3: Knowledge hypergraph completion results on JF17K, FB-AUTO and M-FB15K for baselines and the proposed method. The prefixes ‘r’ and ‘m’ in the model names stand for reification and multi-arity respectively. Both our methods outperform the baselines on all datasets. Results for JF17K dataset is an average of three runs.

<table>
<thead>
<tr>
<th>Model</th>
<th>JF17K</th>
<th>FB-AUTO</th>
<th>M-FB15K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR</td>
<td>H@1</td>
<td>H@3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r-SimplE</td>
<td>0.102</td>
<td>0.116</td>
<td>0.147</td>
</tr>
<tr>
<td>m-DistMult</td>
<td>0.463</td>
<td>0.510</td>
<td>0.634</td>
</tr>
<tr>
<td>m-CP</td>
<td>0.391</td>
<td>0.443</td>
<td>0.563</td>
</tr>
<tr>
<td>m-TransH</td>
<td>0.444</td>
<td>0.475</td>
<td>0.581</td>
</tr>
<tr>
<td>HSimplE (Ours)</td>
<td>0.472</td>
<td>0.520</td>
<td>0.644</td>
</tr>
<tr>
<td>HypE (Ours)</td>
<td>0.494</td>
<td>0.538</td>
<td>0.656</td>
</tr>
</tbody>
</table>

To accommodate non-binary relations, we redefine this function as follows.

\[ \phi_\theta(r(e_i, \ldots, e_j)) = \odot(r, e_i, \ldots, e_j) \]

m-CP. Canonical Polyadic decomposition [59] is a tensor decomposition approach. We refer to the version that only handles binary relations as CP. CP embeds each entity \( e \) as two vectors \( e^{(1)} \) and \( e^{(2)} \), and each relation \( r \) as a single vector \( r \); it defines the score function \( \phi_\theta(r(e_i, e_j)) = \odot(r, e_i^{(1)}, e_j^{(2)}) \). We call the version of CP that accommodates relations of any arity \( m \)-CP. \( m \)-CP embeds each entity \( e \) as \( \alpha \) different vectors \( e^{(1)}, \ldots, e^{(\alpha)} \), where \( \alpha = \max_{r \in \mathbb{R}}(|r|) \); it computes the score of a tuple as follows.

\[ \phi_\theta(r(e_i, \ldots, e_j)) = \odot(r, e_i^{(1)}, \ldots, e_j^{(|r|)}) \]

4.2.6 Experiments

This section summarizes our experiments.

Knowledge Hypergraph Completion Results

The results of our experiments, summarized in Table 4.3, show that both HSimplE and HypE outperform the proposed baselines across the three knowledge hypergraph datasets JF17K, FB-AUTO, and M-FB15K. They further demonstrate that reification for the r-SimplE model does not work well; this is because the reification process introduces auxiliary entities for which the model does not learn appropriate embeddings because these auxiliary entities appear in very few facts. Comparing the results of r-SimplE against
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

Figure 4.3: These experiments show that HypE outperforms HSimplE when trained with fewer parameters, and when tested on samples that contain at least one entity in a position never encountered during training on JF17K dataset. (a) MRR of HypE and HSimplE for different embedding dimensions. (b) Results of m-CP, HSimplE, and HypE on the missing positions test set (containing 1,806 test samples).

HSimplE, we can also see that extending a model to work with hypergraphs works better than reification when high-arity relations are present.

The ability of knowledge sharing through the learned position-dependent convolution filters suggests that HypE would need fewer parameters than HSimplE to obtain good results. To test this, we train both models with different embedding dimensions. Figure 4.3a shows the MRR on the test set for each model with different embedding sizes. Based on the MRR result, we can see that HypE outperforms HSimplE by 24% for embedding dimension 50, implying that HypE works better under a constrained budget.

Disentangling the representations of entity embeddings and positional filters enables HypE to better learn the role of position within a relation because the learning process considers the behavior of all entities that appear in a given position at the time of training. This becomes especially important in the case when some entities never appear in certain positions in the train set, but you still want to be able to reason about them no matter what position they appear in at test time. To test the effectiveness of our models in this more challenging scenario, we created a missing positions test set by selecting the tuples from our original test set that contain at least one entity in a position it never appears in within the training dataset. The results on these experiments (Figure 4.3b) show that (1) both HSimplE and HypE outperform m-CP (which learns different embeddings for each entity-position pair), and more importantly, (2) HypE significantly outperforms HSimplE.
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

Table 4.4: Knowledge graph completion results on wn18 for baselines and HypE. Note that in knowledge graphs (binary relations), HSimplE and SimplE are equivalent, both theoretically and experimentally. The results show that our methods outperform the baselines.

\[
\begin{array}{c|cccc}
\text{Model} & \text{MRR} & \text{Hit@1} & \text{Hit@3} & \text{Hit@10} \\
\hline
\text{CP} [59] & 0.074 & 0.049 & 0.080 & 0.125 \\
\text{TransH} [151] & - & - & - & 0.867 \\
\text{m-TransH} [153] & 0.671 & 0.495 & 0.839 & 0.923 \\
\text{DistMult} [158] & 0.822 & 0.728 & 0.914 & 0.936 \\
\text{HSimplE (Ours)} & \text{0.942} & \text{0.939} & \text{0.944} & \text{0.947} \\
\text{HypE (Ours)} & 0.934 & 0.927 & 0.940 & 0.944 \\
\end{array}
\]

Table 4.5: Knowledge graph completion results on fb15k for baselines and HypE. Note that in knowledge graphs (binary relations), HSimplE and SimplE are equivalent, both theoretically and experimentally. The results show that our methods outperform the baselines.

\[
\begin{array}{c|cccc}
\text{Model} & \text{MRR} & \text{Hit@1} & \text{Hit@3} & \text{Hit@10} \\
\hline
\text{CP} [59] & 0.326 & 0.219 & 0.376 & 0.532 \\
\text{TransH} [151] & - & - & - & 0.585 \\
\text{m-TransH} [153] & 0.351 & 0.228 & 0.427 & 0.559 \\
\text{DistMult} [158] & 0.654 & 0.546 & 0.733 & 0.824 \\
\text{HSimplE (Ours)} & \text{0.727} & \text{0.660} & \text{0.773} & \text{0.838} \\
\text{HypE (Ours)} & 0.725 & 0.648 & \text{0.777} & \text{0.856} \\
\end{array}
\]

for this challenging test set, leading us to believe that disentangling entity and position representations may be a better strategy for this scenario.

**Knowledge Graph Completion Results**

To show that HSimplE and HypE work well also on the more common knowledge graphs, we evaluate them on wn18 and fb15k. Tables 4.4 and 4.5 shows link prediction results on wn18 and fb15k. Baseline results are taken from the original papers except that of m-TransH, which we implement ourselves. Instead of tuning the parameters of HypE to get potentially better results, we follow the Kazemi and Poole (2018) setup with the same grid search approach by setting \( n = 2, \ l = 2, \) and \( s = 2. \) This results in all models in Tables 4.4 and 4.5 having the same number of parameters,
4.2. Knowledge Hypergraph Completion: HSimplE and HypE

Table 4.6: Breakdown performance of hit@10 across relations with different arities on JF17K dataset along with their statistics.

<table>
<thead>
<tr>
<th>Arity</th>
<th>Model</th>
<th>2</th>
<th>3</th>
<th>4-5-6</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>r-SimplE</td>
<td>0.478</td>
<td>0.025</td>
<td>0.017</td>
<td>0.168</td>
</tr>
<tr>
<td></td>
<td>m-DistMult</td>
<td>0.495</td>
<td>0.648</td>
<td>0.809</td>
<td>0.634</td>
</tr>
<tr>
<td></td>
<td>m-CP</td>
<td>0.409</td>
<td>0.563</td>
<td>0.765</td>
<td>0.560</td>
</tr>
<tr>
<td></td>
<td>m-TransH</td>
<td>0.411</td>
<td>0.617</td>
<td>0.826</td>
<td>0.596</td>
</tr>
<tr>
<td></td>
<td>HSimplE (Ours)</td>
<td><strong>0.497</strong></td>
<td><strong>0.699</strong></td>
<td><strong>0.745</strong></td>
<td><strong>0.645</strong></td>
</tr>
<tr>
<td></td>
<td>HypE (Ours)</td>
<td>0.466</td>
<td>0.693</td>
<td><strong>0.858</strong></td>
<td><strong>0.656</strong></td>
</tr>
<tr>
<td># train tuples</td>
<td>36,293</td>
<td>18,846</td>
<td>6,772</td>
<td>61,911</td>
<td></td>
</tr>
<tr>
<td># test tuples</td>
<td>10,758</td>
<td>10,736</td>
<td>3,421</td>
<td>24,915</td>
<td></td>
</tr>
</tbody>
</table>

and thus makes them directly comparable to each other. Note that for knowledge graph completion (all binary relations) HSimplE is equivalent to SimplE, both theoretically and experimentally (as shown in Section 4.2.2). The results show that on wn18 and fb15k, HSimplE and HypE outperform all baselines.

Ablation Study on Different Arities

We break down the performance of models across different arities. As the number of test tuples in higher arities (4-5-6) is much less than in smaller arities (2-3), we used equivalent size bins to show the decomposed results for a reasonable number of test tuples. Table 4.6 shows the hit@10 results of the models for bins of arity 2, 3, and 4-5-6 in JF17K. The proposed models outperform the state-of-the-art and baselines in all arities. We highlight that r-SimplE and HSimplE are quite different models for relations having arity > 2.

4.2.7 Conclusions

Knowledge hypergraph completion is an important problem that has received little attention. Having introduced two new knowledge hypergraph datasets, baselines, and two new methods for link prediction in knowledge hypergraphs, we hope to kindle interest in the problem. Unlike graphs, hypergraphs have a more complex structure that opens the door to more challenging questions such as: how do we effectively predict the missing entities in a given (partial) tuple?
4.3 Knowledge Hypergraph Completion with Relational Algebra: ReAlE

Section 4.2 introduces link prediction models for knowledge hypergraphs inspired by link prediction models for knowledge graphs, benchmarks, baselines, and evaluation metrics for this task. This section explores the gap between relational algebra foundations and machine learning techniques for knowledge hypergraph completion.

Relational databases have been a successful model for data storage, and have relied on query languages for information retrieval. Most of these query languages are based on relational algebra, a mathematical formalization at the core of relational models. This work takes a look at knowledge hypergraph completion through the lens of relational algebra and its core operations. We explore the gap between relational algebra foundations and machine learning techniques for knowledge completion.

Recent research [e.g., 10, 138] has highlighted the importance of relational inductive biases in building learning agents that learn entity-independent relational semantics and reason in a compositional manner. In this work, we explore the foundations of knowledge hypergraph completion; we aim to design a model for reasoning in knowledge hypergraphs that is simple, expressive, and can represent high-level abstractions in terms of the operations of relational databases. We hypothesize that models that can reason about relations in terms of relational algebra operations have better generalization power.

**Constraint vs. query languages.** To understand the role of relational algebra, and to understand its relationships to other foundations, it is important to understand the distinction between constraint and query languages. In relational databases, constraints and queries play complementary roles. Constraints specify restrictions to impose on the database, while queries extract information that may not be explicitly encoded. In other words, constraints imply that some combination of tuples must be false, while queries help us determine what else must be true. Both constraints and queries can be written as (a subset of) first-order logic rules. Consider the following example. Suppose the data tells us that Sam lives with Sally (which we write as LivesWith(Sam, Sally)) and Sally lives in Paris (written as LivesIn(Sally, Paris)). A query language would let us state that someone lives in the same city as a person they live with, which would let us infer LivesIn(Sam, Paris). The constraint that someone only lives in one city
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

makes us reject the statement $\text{LivesIn}(Sally, Berlin)$.

Unlike in constraint languages, defining one relation in terms of others in a query language does not have side effects – i.e. the representation of a relation $r$ does not affect the truth of relations not defined in terms of $r$, which is crucial when we want to capture multiple inference patterns jointly. We can think of a query language as being similar to a directed graphical model and a constraint language as an undirected graphical model. In a directed graphical model, we add a new random variable by using existing variables as its parents. Adding the variable does not change the distribution of the parents. Whereas, in an undirected graphical model, adding a new variable can change the distribution of other variables; indeed for relational undirected models (in particular Markov logic networks), it has been proved that it is impossible to add a new variable without changing the distribution over the existing variables except in trivial cases [17]. A learning model that is based on a query language (e.g. one with relational algebra foundations) can easily capture multiple inference patterns jointly, a challenge underlined by Abboud et al. [3] who state that “capturing multiple inference patterns jointly is significantly more challenging than capturing them singly.” This is true of languages that include constraints, but not of query languages.

Relational algebra. Relational algebra is a formalization of queries and defines relations (views) at the core of relational databases. It consists of several primitive operations that can be combined to synthesize all other operations used. The primitive operations are renaming, projection, selection, set union, set difference, and Cartesian product. Each such operation takes relations as input and returns a relation as output. Renaming changes the order of the entities in a relation. Projection takes a relation and some positions as input and returns a new relation with the entities in specific positions removed from each tuple. Selection returns a subset of tuples for a relation that satisfies a given condition. Set union takes as input two relations of the same arity and returns a new relation containing the tuples that appear in at least one of the relations. Set difference also gets two relations of the same arity as input and returns a new relation containing tuples from the first relation that do not appear in the second relation. Cartesian product takes two relations and returns a relation in which the tuples are the concatenation of the tuples of the input relations. One non-primitive operation is join, which is a Cartesian product followed by selection and projection; selecting the elements in the Cartesian product with matching values on corresponding attributes, and projecting onto the different attributes. Another
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

![Figure 4.4: An example of a knowledge hypergraph. Train set contains sold(drew, alex, book), buyer(alex, book), sold(mike, sam, tv), and bought(sam, mike, tv). Relation bought can be obtained by applying a renaming operation to relation sold. Similarly, relation buyer is a projection of relation sold. Learning these relational algebra operations can help the model generalize to the tuples in the test set. The test responses, from top to bottom, are drew, tv, and mike.](image)

Non-primitive operation is set intersection, which can be defined in terms of set union and set difference.

**Relational Algebra in Knowledge Hypergraphs.** In a knowledge hypergraph, the relational algebra operations can describe how relations depend on each other. To illustrate the connection between relational algebra operations and relations in knowledge hypergraphs, consider the example in Figure 4.4 that shows tuples from a train and test splits of a knowledge hypergraph. The train set contains tuples sold(drew, alex, book), buyer(alex, book), sold(mike, sam, tv), and bought(sam, mike, tv). The relations in the example feature the two primitive relational algebra operations renaming and projection. Relation bought is a renaming of sold. Relation buyer is a projection of relation sold. If a model is able to represent these two operations, it can potentially learn at train time that a tuple bought(X, Y, I) (person X bought from person Y item I) is implied by tuple sold(Y, X, I) (person Y sold to person X item I); or that a tuple buyer(X, I) (person X is the buyer of item I) is implied by the tuple sold(Y, X, I). An embedding model that cannot represent the operations renaming and projection would not be able to learn that relation bought in Figure 4.4 is a renam-
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

It would thus be difficult for such a model to reason about the relationship between these two relations. In contrast, a model that can represent renaming and projection operations is potentially able to determine that bought(alex, drew, book) is true because the train set contains sold(drew, alex, book) and bought is a renaming of sold, buyer(sam, tv) is true because the train set contains sold(mike, sam, tv) and buyer is a projection of sold, and sold(mike, sam, tv) is true because the train set contains bought(sam, mike, tv) and sold is a renaming of bought.

Designing reasoning methods that can capture the relational semantics in terms of relational algebra is especially important in the context of knowledge hypergraphs, where relations can be defined on an arbitrary number of entities. Domains with beyond-binary relations provide multiple methods of expressing the same underlying notion, as seen in the above example where relations sold and bought encode the same information. Since all relations in a knowledge graph are binary (have arity 2), many of the relational algebra operations (such as projection, which change the arity of the relation; for instance, buyer with arity 2 is a projection of sold with arity 3), are not applicable to this setting. This also highlights the importance of knowledge hypergraphs as a data model that encodes rich relational structures ripe for further exploration.

The main contributions of this section are summarized as follows.

• We introduce ReAlE, an embedding-based method for knowledge hypergraph completion that can provably represent the relational algebra operations renaming, projection, set union, selection, and set difference,

• A framework for generating synthetic knowledge hypergraphs, which is based on the Erdős-Rényi random graph generation model and can generate relations by repeated application of primitive relational algebra operations.

• Experimental results that show that ReAlE outperforms or is comparable to the state-of-the-art on well-known public datasets, and the synthesized dataset.

4.3.1 Related Work

Existing work for knowledge hypergraph completion can be grouped into the following categories.
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

**Statistical relational learning.** Models under the umbrella of statistical relational learning \[116\] can handle variable arity relations and explicitly model the inter-dependencies of relations. Our work is complementary to these approaches in that ReAlE is an embedding-based model that represents relational algebra operations implicitly, rather than representing them explicitly.

**Translational models.** Translational models for knowledge hypergraphs are extensions of approaches for binary relations and have restrictions on the types of relations they can model (see Section 4.3.3). One of the earliest models in this category is m-TransH \[153\], which extends TransH \[151\] to knowledge hypergraph embedding. RAE \[168\] extends m-TransH by adding the relatedness of values – the likelihood that two values co-participate in a common instance – to the loss function. NaLP \[51\] uses a similar strategy to RAE, but models the relatedness of values based on the roles they play in different tuples. Models in this category have restrictions on the types of relations they can model (see Section 4.3.3). Liu et al. \[97\] discuss these limitations, and we address them more formally in Section 4.3.3. More recently, Abboud et al. \[3\] proposed a fully expressive translational model based on box embedding \[91\].

**Tensor factorization models.** Models in this category extend tensor factorization models for knowledge graph completion to knowledge hypergraph completion. GETD \[97\] extends Tucker \[7\] to n-ary relations. The memory complexity of GETD grows exponentially with the arity of relations. HypE \[40\] which is motivated by SimplE \[76\] disentangles the embeddings of relations from the positions of its arguments and thus the memory complexity grows linearly with the arity of the relations. HypE is fully expressive but cannot represent relational algebra operations (See Section 4.3.3).

**Key-value pair based models.** Galkin et al. \[48\], Guan et al. \[52\], and Rosso et al. \[121\] assume that a tuple is composed of a triple (binary relation), plus a list of attributes in the form of key-value pairs. This problem is slightly different as we consider all information as part of one tuple. They evaluate their models on datasets for which they obtain the tuple attributes from external data sources using heuristics and thus their results are not comparable to ours.
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

Graph neural network models. These approaches extend graph neural networks to hypergraph neural networks [44, 156]. G-MPNN [155] further extends these models to knowledge hypergraphs (directed and labeled hyperedges). G-MPNN utilizes message passing in knowledge hypergraphs but the scoring function assumes relations are symmetric, and thus has restrictions in modeling the non-symmetric relations and is not fully expressive.

Rule capturing models. The closest work to ours is BoxE [3], which is able to represent a subset of first-order logic rules and is also a translational model. ReAlE differs from BoxE in two ways. First, theoretical analysis of BoxE only concerns binary relations, while we capture inference patterns for relations defined on any number of entities; some of the relational algebra relations (e.g., projection) do not make sense with only binary relations. Second, we analyze the theoretical aspects of the proposed model in terms of relational algebra, a query language in which a relation is defined in terms of others without affecting other relations. In contrast, first-order logic rules in BoxE are a mixture of constraints (mutual exclusion and anti-symmetry) and rules that imply what else must be true (e.g., symmetry and intersection). Capturing multiple patterns from first-order logic does not necessarily provide evidence for the rest of the first-order logic. Capturing the primitive operations of relational algebra is important as all other operations are composed of multiple primitive operations. We compare our model empirically with BoxE in Section 4.3.7.

Proposed method. Reasoning in hypergraphs is a relatively underexplored area that has recently gained more attention. Knowledge hypergraphs are isomorphic to relational databases and relational algebra is the calculus of queries in relational models. In this work, we design a model based on relational algebra operations. Besides the theoretical contributions, we show empirically how basing our model on relational algebra operations gives us improvements compared to existing work.

4.3.2 ReAlE: A Basic Embedding Algorithm

ReAlE (Relational Algebra Embedding) is a knowledge hypergraph completion model that has parameters \( \theta \) and a scoring function \( \phi_\theta \). We motivate our model bottom-up by first describing an intuitive model for this task (Equation 4.6); we then discuss why this formulation does not work well and adjust it in ReAlE (Equation 4.7). Given a tuple \( r(x_1, \ldots, x_n) \),
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

Figure 4.5: A schematic of the entity and relation embeddings in ReAlE: the embedding dimension \( d \) is divided into \( n_w \) windows of size \( w \).

determining whether it is true or false depends on the relation and the entities involved; it also depends on the position of each entity in the tuple, as the role of an entity changes with its position and relation. For example, the role of \( \text{alex} \) is different in the tuples \( \text{sold}(\text{drew}, \text{alex}, \text{book}) \) and \( \text{sold}(\text{alex}, \text{drew}, \text{book}) \) as its position is different. The role of \( \text{alex} \) is also different in the tuples \( \text{sold}(\text{drew}, \text{alex}, \text{book}) \) and \( \text{bought}(\text{drew}, \text{alex}, \text{book}) \) as the relation is different.

An intuitive model to decide whether a tuple is true or not is one that embeds each entity \( x_i \in \mathcal{E} \) into a vector \( \mathbf{x}_i \in [0, 1]^d \) of length \( d \), and the relation \( r \) into a matrix \( \mathbf{R} \in \mathbb{R}^{|r| \times d} \), where the \( i^{th} \) row in \( \mathbf{R} \) operates over the entity at position \( i \). Each relation \( r \) has a learnable bias term \( b_r \) as a part of the relation embedding as a relation-dependent constant that does not depend on any entities and allows the model. Such a model defines the following scoring function, where \( \sigma \) is a nonlinear function that is differentiable almost everywhere.

\[
\phi_r^\sigma(r(x_1, \ldots, x_n)) = \sigma(b_r + \sum_{i=1}^{|r|} \sum_{k=0}^{d-1} x_i[k] \times \mathbf{R}[i][k]) \quad (4.6)
\]

Instead of performing a single version of the above scoring function, we found it beneficial to consider multiple versions of the scoring function and use the output of all of the functions to obtain the final predictive performance. Scores from multiple versions of Equation 4.6 are summed to produce the final score for the input tuple. The idea of using an ensemble of models [32] allows each of the scoring functions to focus on a different
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

aspect of the problem. This is similar to ideas for having multiple heads in self-attention layers \cite{142}.

Here, we introduce the concept of windows, a range of indices whereby elements within the same window are used in one scoring function. The number of embedding elements for each entity in a window, the \textit{window size}, is a hyperparameter (see Figure \ref{fig:windows}). Let \( w \) denote the window size, \( n_w = \lfloor \frac{d}{w} \rfloor \) the number of windows, and \( b_{jr} \) the learnable bias term of relation \( r \) for the \( j \)th window, for all \( j = 0, \ldots, n_w - 1 \). Equation \ref{eq:reale_score} defines ReAlE’s score of a tuple \( r(x_1, x_2, \ldots, x_n) \), where \( \sigma \) is a monotonically-increasing nonlinear function that is differentiable almost everywhere.

\[
\phi_\theta(r(x_1, \ldots, x_n)) = \sum_{j=0}^{n_w-1} \sigma(b_{jr} + |r| \sum_{i=1}^{w-1} \sum_{k=0}^{w-1} z_i[jw + k] \times R[i][jw + k]) \tag{4.7}
\]

\textbf{Learning ReAlE model.} To learn a ReAlE, we follow the same procedure as for learning HSimpE and HypE as explained in Section \ref{sec:learning}. For a ReAlE models with parameters \( \theta \) (including relation and entity embeddings) and \( \phi_\theta \) as the function in the Equation \ref{eq:reale_score}, we minimize the cross-entropy loss defined in Section \ref{sec:learning}. Algorithm \ref{alg:learning_reale} shows a high-level description of how we train a ReAlE model.

\begin{algorithm}
\caption{Learning ReAlE}
Input: Tuples \( \tau'_{\text{train}} \), loss function \( L \), scoring function \( \phi_\theta \)
Output: Embeddings \( z \) and \( r \) for all entities and relations in \( \tau'_{\text{train}} \).
Initialize \( z \) and \( r \) (at random)
\For {every batch \( \tau'_{\text{batch}} \) of tuples in \( \tau'_{\text{train}} \)}
\For {tuple \( t \) in \( \tau'_{\text{batch}} \)}
Generate negative tuples \( T_{\text{neg}}(t) \)
\For {\( t' \in \{t\} \cup T_{\text{neg}}(t) \)}
Compute \( \phi_\theta(t') \) \hspace{1cm} (Equation \ref{eq:reale_score})
end for
end for
\end{algorithm}

Compute the loss \( L(\theta, \tau'_{\text{batch}}) \) \hspace{1cm} (Equation \ref{eq:cross_entropy})
Compute the gradient of the loss with respect to \( z \) and \( r \)
Update embeddings \( z \) and \( r \) through back-propagation

end for

56
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

4.3.3 Theoretical Analysis

To better understand the expressive power of ReAlE and the types of reasoning it can perform, we analyze the extent of its expressivity and its capacity to represent relational algebra operations without the operations being known or given to the learner. Relational algebra allows for quantification, in particular, statements that are true for all entities or statements that are true for at least one. To allow for such statements, we introduce the notion of variables, which, following the convention of Datalog, we write in upper case, e.g., $X_1, \ldots, X_n$. We use lower case $x_1, \ldots, x_n$ to denote particular entities. To make a meaningful statement, each variable is quantified using $\forall$ (the statement is true for all assignments of entities to the variable) and $\exists$ (the statement is true if there exists an assignment of an entity to the variable). Here, $\bar{x}$ is a sequence of particular entities, and $\bar{X}$ is a sequence of variables. We use $\neg$ as negation, $\land$ as conjunction (and), and $\lor$ as disjunction (or). In relational algebra, each relation has a unique definition and there are no cyclic or recursive definitions.

For any $\bar{x}$ and any relation $r$, we define the relation complement function $f$ as $f(\phi_\theta(r(\bar{x}))) = \phi_\theta(\neg r(\bar{x}))$. This function depends on the choice of the nonlinearity $\sigma$ of the scoring function in Equation 4.7. For example, if $\sigma$ is the Sigmoid function, then $f(\phi_\theta(r(\bar{x}))) = 1 - \phi_\theta(r(\bar{x}))$; if it is the hyperbolic tangent (tanh), then $f(\phi_\theta(r(\bar{x}))) = -\phi_\theta(r(\bar{x}))$.

In this section, we state the theorems and defer all the proofs to Appendix B.2 starting at Page 123.

4.3.4 Full expressivity

The two results in this section state that ReAlE is fully expressive and that some other models are not.

**Theorem 4.2 (Full Expressivity)** For any ground truth over entities $E$ and relations $R$ containing $\lambda$ true tuples with $\alpha = \max_{r \in R}(|r|)$ as the maximum arity over all relations in $R$, there is a ReAlE model with $n_w = \lambda$, $w = \alpha$, $d = \max(\alpha \lambda, \alpha)$, and $\sigma(x) = \frac{1}{1 + \exp(-x)}$ that accurately separates the true tuples from the false ones.

**Theorem 4.3** m-TransH, RAE, and NaLP are not fully expressive.

4.3.5 Representing Relational Algebra with ReAlE

Here, we describe some primitive operations and prove how closely ReAlE can represent each.
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

Renaming

Renaming changes the order of one or more entities in a relation. A renaming operation can be written as the following logical rule, where \( t \) is defined in terms of \( s \) and \( \pi \) defines a permutation function.

\[
\forall X_1 \ldots \forall X_n \quad t(X_1, \ldots, X_n) \leftrightarrow s(X_{\pi(1)}, \ldots, X_{\pi(n)}) \quad (4.8)
\]

For example, \( \forall X \forall Y \forall I \textit{ bought}(X, Y, I) \leftrightarrow \textit{sold}(Y, X, I) \) represents renaming relation (person \( X \) bought \( I \) from person \( Y \)) into relation (person \( Y \) sold \( I \) to person \( X \)).

**Theorem 4.4 (Renaming)**Given permutation function \( \pi \), and relation \( s \), there exists a parametrization for relation \( t \) in ReAlE such that for entities \( x_1, \ldots, x_n \), with arbitrary embeddings,

\[
\phi_\theta(t(x_1, \ldots, x_n)) = \phi_\theta(s(x_\pi(1), x_\pi(2), \ldots, x_\pi(n)))
\]

Projection

Projection takes a relation as input and removes some entities corresponding to some specific positions in the relation. A projection operation that defines \( t \) as a projection of \( s \) can be written as the following (for \( m < n \)).

\[
\forall X_1 \ldots \forall X_m \quad t(X_1, \ldots, X_m) \leftrightarrow \exists X_{m+1} \ldots \exists X_n \quad s(X_1, \ldots, X_m, \ldots, X_n) \quad (4.9)
\]

Note that projection can be paired with renaming to allow for arbitrary subsets and ordering of arguments. For example, \( \forall X \forall I \textit{ seller}(X, I) \leftrightarrow \exists P \textit{ bought}(P, X, I) \).

**Theorem 4.5 (Projection)** For any relation \( s \) on \( n \) arguments there exists a parametrization for relation \( t \) on \( m < n \) arguments in ReAlE such that for any arbitrary sequence \( x_1, \ldots, x_n \)

\[
\phi_\theta(t(x_1, \ldots, x_m)) \geq \phi_\theta(s(x_1, \ldots, x_n))
\]

Inequality is the best we can hope for because multiple tuples with relation \( s \) might project to the same tuple with relation \( t \). The score of the tuple with relation \( t \) should thus be greater than or equal to the maximum score for \( s \).
Selection

Selection returns the subset of tuples of a relation that satisfies a given condition. Here, we consider equality conditions whereby a selection operation reduces the number of arguments, and has two forms defining \( t \) as a selection of \( s \).

\[
\forall X_1 \ldots \forall X_n \quad t(X_1, \ldots, X_p-1, X_p+1, \ldots, X_q, \ldots, X_n) \\
\leftrightarrow \exists X_p \quad s(X_1, \ldots, X_n) \land (X_p = X_q) \quad (4.10)
\]

For example, \( \forall X \forall Y \quad \text{sold}\_\text{coffee}(X, Y) \leftrightarrow \exists I \quad \text{sold}(X, Y, I) \land (I = \text{coffee}) \).

Observe that selecting tuples with the condition \( X_p = X_q \) for arbitrary \( p \) and \( q \) is equivalent to first renaming the tuple so that \( X_p \) is in position \( n \) and \( X_q \) is in position \( n-1 \); and then performing a selection with the condition \( X_{n-1} = X_n \) or \( X_n = c \). Thus, we show the selection operation for the case when \( X_{n-1} = X_n \) and \( X_n = c \).

**Theorem 4.6 (Selection 1)** For arbitrary relation \( s \), there exists a parametrization for relation \( t \) in ReAlE such that for arbitrary entities \( x_1, \ldots, x_n \)

\[
\phi_\theta(t(x_1, \ldots, x_{n-1})) = \phi_\theta(s(x_1, \ldots, x_{n-1}, x_{n-1})) \quad (4.12)
\]

**Theorem 4.7 (Selection 2)** For arbitrary relation \( s \) and for a fixed constant \( c \), there exists a parametrization for relation \( t \) in ReAlE such that for arbitrary entities \( x_1, \ldots, x_n \)

\[
\phi_\theta(t(x_1, \ldots, x_{n-1})) = \phi_\theta(s(x_1, \ldots, x_{n-1}, c))
\]

Set union

Set union operates on relations of the same arity, and returns a new relation containing the tuples that appear in at least one of the relations. A set union operation can be written as the following logical rule, with relation \( t \) as the union of \( s \) and \( r \).

\[
\forall \bar{X} \quad t(\bar{X}) \leftrightarrow s(\bar{X}) \lor r(\bar{X}) \quad (4.13)
\]
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlIE

For instance, relation \textit{traded} is defined as a set union of relations \textit{sold} and \textit{bought} in the following example.

\[
\forall X_1 \ \forall X_2 \ \forall I \ \ \text{traded}(X_1, X_2, I) \leftrightarrow \text{sold}(X_1, X_2, I) \lor \text{bought}(X_1, X_2, I)
\]

For a ReAlIE model to be able to represent the set union operation, first observe that any score for a tuple \( t \) that represents the union of relations \( r \) and \( s \) depends on how dependent the two relations \( r \) and \( s \) are. For example, if \( s \) is a subset of \( r \), then the score of \( t \) is equal to that of \( r \). But since we do not know about such dependence relations in the data, then a reasonable bound for the score of \( t \) is \textit{at least} as high as the maximum score of either \( r \) or \( s \), as the following lemma states.

\textbf{Theorem 4.8 (Set Union)} For arbitrary relations \( s \) and \( r \) with the same arity, there exists a parametrization for relation \( t \) in ReAlIE such that for arbitrary entity set \( \bar{x} \)

\[
\phi_\theta(t(\bar{x})) \geq \max(\phi_\theta(s(\bar{x})), \phi_\theta(r(\bar{x})))
\]

\textbf{Set difference}

\textbf{Set difference} operates on relations of the same arity, and returns a new relation containing the tuples from the left relation that do not appear in the right one. The set difference operation can be written as the following logical rule, where relation \( t \) is set difference of \( s \) and \( r \).

\[
\forall \bar{X} \ \ t(\bar{X}) \leftarrow s(\bar{X}) \land \neg r(\bar{X}) \quad (4.14)
\]

The following shows an example of a set difference.

\[
\forall X \ \forall Y \ \ \text{needs_filter}(X, Y) \leftarrow \text{bought_coffee}(X, Y) \land \neg \text{bought_filter}(X, Y)
\]

Similar to set union, the score of a set difference operator depends on how dependent the relations \( r \) and \( s \) are. For the same reasons, the best we can hope for in this case is to show that the score of \( t \) is smaller than that of both \( s \) and \( \neg r \) (since \( t(\bar{X}) \) is true only when both \( s(\bar{X}) \) and \( \neg r(\bar{X}) \) are true, then the scores of the latter two must be higher). In the lemma that follows, \( f \) is the relation complement function described in the introduction of Section 4.3.3 Here, we assume that \( f \) exists for the selected \( \sigma \).

\textbf{Theorem 4.9 (Set Difference)} For arbitrary relations \( r \) and \( s \) with the same arity, if \( f \) is a linear relation complement function and \( f(\sigma(x)) =
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

\[ \sigma(c \ast x) \] with \( c \) as a constant, there exists a parametrization for relation \( t \) in ReAlE such that for arbitrary entities \( x_1, \ldots, x_n \)

\[ \phi_\theta(t(\bar{x})) \leq \min(\phi_\theta(s(\bar{x})), f(\phi_\theta(r(\bar{x})))) \]

The intuition behind the proof of Theorems 4.4 to 4.9 is to come up with a parametrization for the resulting relation \( t \) based on the relation(s) in the operation such that the operation hold.

**Joint representation of operations**

The following theorems establish the ability of ReAlE to jointly capture the relational algebra operations discussed above. This is of interest, particularly because capturing multiple inference patterns jointly has been deemed challenging in some existing methods [3]. In our case, the parametrizations do not interfere with each other, as each rule (operation) defines the relation in the head without side effects on the relations in the body.

**Theorem 4.10 (Composition)** For an arbitrary set of relations \( S_r \) and arbitrary non-empty composition of operations \( S_{op} \) from the set renaming, projection, selection, set union, and set difference, there exists a parametrization for relation \( t \) in ReAlE with \( t \) as the resulting relation of applying \( S_{op} \) to \( S_r \).

**Theorem 4.11 (Joint representation)** ReAlE is able to jointly represent a set of relations each being either the result of a relational algebra operation renaming, projection, selection, set difference, set union, or a composition of these operations.

4.3.6 Datasets

**Real-world Datasets.**

We use three the real-world datasets introduced in Section 4.2.4 for our experiments: jf17k [153], and FB-AUTO and m-FB15K [40].

**Synthetic Dataset**

To study and evaluate the generalization power of models in a controlled environment, we generate a synthetic dataset. This practice has become common in recent years, with the creation of several procedurally generated benchmarks to study the generalization power of models in different tasks. Examples of such datasets include CLEVR [72] for images and
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAI

TextWorld [28] for text data, and GraphLog [131] for graph data. To create a benchmark for analyzing the relational algebraic generalization power of ReAI for hypergraph completion, we consider the following criteria. (1) Completeness: The benchmarks must contain the desired relational algebra operations; in our case: renaming, projection, selection, set union, and set difference. (2) Diversity: The benchmark must contain a variety of relations that are the result of repeated application of relational algebra operations having varying depths. (3) Compositional generalization: The benchmark must contain relations that are the result of repeated application of operations of different types. To synthesize a dataset that satisfies the above conditions, we extend the Erdős-Rényi model [36] for generating random graphs to directed edge-labeled hypergraphs. We use this hypergraph generation model to first generate a given number of true tuples; we then apply the five relational algebra operations to these tuples (repeatedly and recursively) to obtain new tuples with varying depths. In what follows, we discuss the details of our dataset generation algorithm.

**Knowledge Hypergraphs.** A knowledge hypergraph is a directed hypergraph \( H = (V, E, R) \) with nodes (entities) \( V \), edges (tuples) \( E \), and edge labels (relations) \( R \) such that:

- every edge in the hypergraph consists of an ordered sequence of nodes,
- every edge has a label \( r_i \in R \), and
- edges with the same label are defined on the same number of nodes.

Observe that in knowledge hypergraphs, edges having the same label form a uniform directed hypergraph (all edges defined on the same number of nodes). We can thus think of \( H \) as the combination of \( |R| \) directed uniform hypergraphs.

**Extending Erdős-Rényi to Knowledge Hypergraphs.** In the Erdős-Rényi model, all graphs with a fixed number of nodes and edges are equally likely. Equivalently, in such a random graph, each edge is present in the graph with a fixed probability \( p \), independent of other edges. In this section, we describe a method of generating a random knowledge hypergraph inspired by the Erdős-Rényi process.

Let \( n \) be the (predefined) number of nodes in the hypergraph and \( n_r \) be the number of relations. We let \( R \) be a list of relations defined in terms of
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

arity and a probability that influences the number of tuples generated for that given relation. More formally,

\[ R = \{(k_i, p_i) | k_i = \text{arity}, 0 \leq p_i \leq 1, \forall i = 0, \ldots, n_r\} \]

The expected number of edges generated for a given relation \( r_i \) is \( m_i = k_i! \binom{n}{k_i} p_i \). As the process of including edges in the graph is a Binomial, we can compute this expected value by sampling from the following probability density.

\[ P(m_i) = \binom{N}{m_i} p_i (1 - p_i)^{N - m_i} \]

where \( N = k_i! \binom{n}{k_i} \) is the number of possible \( k_i \)-uniform (directed) edges in the hypergraph.

The running time of Algorithm 2 depends on the number of nodes \( n \) and the arity \( k_i \) of each relation. Let \( k = \max_{k_i \in R} k_i \). Thus the running time of Algorithm 2 is \( O(|R| n^k) \).

**Algorithm 2: generate_knowledge_hypergraph(\( V, R \))**

```python
edge_list = []
n = len(V)
for r, (k, p) in enumerate(R) do
    N = k_i! \binom{n}{k_i}
    m = random.binomial(N, p) \{result of flipping a coin \( N \) times with probability of success \( p \}\)
    edge_count = 0
    while edge_count <= m do
        edge = random.sample(V, k) \{select \( k \) vertices from \( V \) at random\}
        if edge not in edge_list then
            edge_list.append([r] + edge)
            edge_count = edge_count + 1
        end if
    end while
end for
return edge_list
```

**Dataset Generation.** To evaluate a model on how well it represents relational algebra operations, we generate a set of ground-truth true tuples, each of which is the result of repeated application of a primary relational algebra operation to an existing tuple (hyperedge). The operations we are interested in are renaming, projection, selection, set union and set difference.
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

Algorithm 3: generate_ground_truth($V, R, n_{\text{derived\_tuples}}$)

```markdown
E = generate_knowledge_hypergraph($V, R$

for $i$ in range($n_{\text{derived\_tuples}}$) do
  op = randomly select one primary operation
  tuple = randomly select one hyperedge from $E$
  apply $op$ to $tuple$
  add $tuple$ to the set of edges $E$
end for
```

Finally, the complete algorithm to generate the train, valid, and test sets of the synthetic dataset is described in Algorithm 4 below.

Algorithm 4: synthesize_dataset($V, R, n_{\text{derived\_tuples}}$)

```markdown
ground\_truth = generate\_ground\_truth($V, R,$

relational\_data = sub-sample from ground\_truth
train, valid, test = randomly split relational\_data into train, valid and test
```

4.3.7 Experiments

We organize our experiments into three groups with different objectives. The goal of the first set of experiments is to evaluate the proposed method on real datasets and compare its performance to that of existing work. Our second goal is to test the ability of ReAlE to represent the relational algebra operations. The final set of experiments is an ablation study that examines the effect of window size.

Results on Real Datasets

We evaluate ReAlE on the three public datasets JF17K, FB-AUTO, and M-FB15K and compare its performance to existing models. On FB-AUTO and M-FB15K, ReAlE outperforms all existing models. These results are encouraging considering that M-FB15K is the largest of all three datasets. ReAlE is comparable to BoxE on JF17K. Results are summarized in Table 4.7. Overall, ReAlE is competitive on all benchmarks and is state-of-the-art on FB-AUTO and M-FB15K.
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

Table 4.7: Knowledge hypergraph completion results on JF17K, FB-AUTO and M-FB15K for baselines and the proposed method. Our method ReAlE outperforms (or is competitive to) the baselines on all datasets. The higher is better for all columns. To measure the stability of ReAlE, we ran it on JF17K 10 times with the best hyperparameters and the standard deviation of MRR is 0.0004. This shows that ReAlE is not sensitive to different runs (different random seeds).

<table>
<thead>
<tr>
<th>Model</th>
<th>JF17K</th>
<th>FB-AUTO</th>
<th>M-FB15K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mrr</td>
<td>hit@1</td>
<td>hit@3</td>
</tr>
<tr>
<td>m-DistMult</td>
<td>0.463</td>
<td>0.372</td>
<td>0.510</td>
</tr>
<tr>
<td>m-CP</td>
<td>0.392</td>
<td>0.303</td>
<td>0.441</td>
</tr>
<tr>
<td>m-TransH</td>
<td>0.444</td>
<td>0.370</td>
<td>0.475</td>
</tr>
<tr>
<td>RAE</td>
<td>0.310</td>
<td>0.219</td>
<td>0.334</td>
</tr>
<tr>
<td>NaLP</td>
<td>0.366</td>
<td>0.290</td>
<td>0.391</td>
</tr>
<tr>
<td>GETD</td>
<td>0.151</td>
<td>0.104</td>
<td>0.151</td>
</tr>
<tr>
<td>HsimplE</td>
<td>0.472</td>
<td>0.378</td>
<td>0.520</td>
</tr>
<tr>
<td>HypE</td>
<td>0.494</td>
<td>0.408</td>
<td>0.538</td>
</tr>
<tr>
<td>G-MPNN</td>
<td>0.501</td>
<td>0.425</td>
<td>0.537</td>
</tr>
<tr>
<td>BoxE</td>
<td>0.553</td>
<td>0.467</td>
<td>0.596</td>
</tr>
<tr>
<td>ReAlE (Ours)</td>
<td>0.559</td>
<td>0.482</td>
<td>0.594</td>
</tr>
</tbody>
</table>

Results on Synthetic Datasets

To evaluate our model in a controlled environment, we create a synthetic dataset whose statistics (e.g., number of entities, number of relations, number of tuples per relation) are proportional to that of JF17K. We call this synthetic dataset REL-ER. We break down the performance of our model based on the type of relational algebra operation and its depth. As a first step of the synthetic dataset generation, we create a list of tuples at random. The relations involved in these tuples are called elementary relations, as they are not generated based on any relational algebra operation. We then create a set of tuples that are based on composite relations: relations that are built from previously defined elementary and composite relations. We let the type of a composite relation be the last operation applied. We decompose our results by type in Table 4.8.

65
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

Table 4.8: Breakdown performance of MRR across composite relations (based on a sequence of primitive operations) and of varying depths on the REL-ER dataset along with their statistics. The higher is better.

<table>
<thead>
<tr>
<th>Model</th>
<th>Operation type</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td></td>
</tr>
<tr>
<td></td>
<td>elementary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>renaming</td>
<td></td>
</tr>
<tr>
<td></td>
<td>project</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set union</td>
<td></td>
</tr>
<tr>
<td></td>
<td>set difference</td>
<td></td>
</tr>
<tr>
<td>m-TransH [153]</td>
<td>.387</td>
<td>.372</td>
</tr>
<tr>
<td></td>
<td>.447</td>
<td>.459</td>
</tr>
<tr>
<td></td>
<td>.464</td>
<td>.468</td>
</tr>
<tr>
<td>HypE [40]</td>
<td>.689</td>
<td>.335</td>
</tr>
<tr>
<td></td>
<td>.877</td>
<td>.865</td>
</tr>
<tr>
<td></td>
<td>.888</td>
<td>.894</td>
</tr>
<tr>
<td></td>
<td>.881</td>
<td>.872</td>
</tr>
<tr>
<td></td>
<td>.897</td>
<td>.976</td>
</tr>
<tr>
<td>BoxE [3]</td>
<td>.695</td>
<td>.327</td>
</tr>
<tr>
<td></td>
<td>.916</td>
<td>.852</td>
</tr>
<tr>
<td></td>
<td>.900</td>
<td>.918</td>
</tr>
<tr>
<td></td>
<td>.885</td>
<td>.904</td>
</tr>
<tr>
<td></td>
<td>.908</td>
<td>.976</td>
</tr>
<tr>
<td>ReAlE (Ours)</td>
<td>.709</td>
<td>.336</td>
</tr>
<tr>
<td></td>
<td>.882</td>
<td>.877</td>
</tr>
<tr>
<td></td>
<td>.932</td>
<td>.923</td>
</tr>
<tr>
<td></td>
<td>.887</td>
<td>.950</td>
</tr>
<tr>
<td></td>
<td>.945</td>
<td>.938</td>
</tr>
<tr>
<td>#test tuples</td>
<td>5378</td>
<td>1833</td>
</tr>
<tr>
<td></td>
<td>458</td>
<td>762</td>
</tr>
<tr>
<td></td>
<td>1676</td>
<td>649</td>
</tr>
<tr>
<td></td>
<td>2075</td>
<td>1204</td>
</tr>
<tr>
<td></td>
<td>245</td>
<td>21</td>
</tr>
</tbody>
</table>

We define the depth of a relation recursively. An elementary relation has a depth of 0. A relation that is the result of a unary operation (e.g., projection or renaming) has a depth of one plus the depth of the input relation. For a relation that is the result of a binary operation (e.g., set union or set difference), the depth is one plus the maximum depth of the input relations. The results of our experiments on REL-ER are summarized in Table 4.8 and show that our proposed model outperforms the state-of-the-art in almost all cases. As the decomposed performance shows, the improvement to the general result is due mostly to improvements in the performance of the operations as well as improvements for elementary relations. These results confirm that our theoretical findings are in line with the practical results. We do not have tuples for the selection operation in the test set because, by nature, selection generates very few tuples; and as the split of train/test/validation in REL-ER is done at random, the chance of having selection tuples in the test set is very low and did not occur when we synthesized the dataset (selection tuples are still present in the train data).

Ablation Study on Varying Number of Windows

Here, we compare the performance of ReAlE with different numbers of windows $n_w$. The negative ratio, batch size, and embedding dimension are fixed to 10, 128, and 200 respectively for this experiment. Here, the total number of parameters for each value of $n_w$ is fixed. The outcome of the study is summarized in Figure 4.6. Number of windows $n_w$ of 1 is the same as Equation 4.6. Increasing $n_w$ means having more models in the ensemble learner. As we increase the number of learners, the performance (MRR) on the test set increases and it plateaus when having a large number of windows. This result confirms the importance of having multiple learners as
4.3. Knowledge Hypergraph Completion with Relational Algebra: ReAlE

Figure 4.6: MRR of ReAlE for different number of windows $n_w$ on JF17k with fixed values for negative ratio, batch size, and embedding dimension of 10, 128, and 200 respectively. The total number of parameters for each value of $n_w$ is fixed.

ReAlE with a fixed negative ratio and batch size

The results reported in Table 4.7 were obtained by running experiments using the BoxE setup: the negative ratio and batch size were tuned for each model. Here, we follow the setup used by all other baselines to report their results and run experiments by fixing the negative ratio to 10 and batch size to 128. We summarize the results for the baselines and ReAlE in Table 4.9. ReAlE (Small) refers to a ReAlE model that has a negative ratio of 10 and a batch size of 128 (called Small because it has a smaller negative ratio and batch size compared to the best set of hyperparameters).

Results of ReAlE with different non-linear functions

Following the last experiment with a fixed negative ratio and batch size, we tried different non-linear functions ($\sigma$). The results of the different non-linear functions are in Table 4.10. As sigmoid is the winner for JF17K, we only experiment with sigmoid for the remaining datasets.

Do the learned embeddings follow the theoretical findings?

Our theoretical analysis shows that ReAlE is able to represent a large subset of relational algebra operations. Here, we further investigate whether, in practice, the embeddings learned by ReAlE follow the theoretical results.
4.4. Conclusion and Limitations

We thus first look at the relations in REL-ER that are the result of the renaming operation and observe the embeddings that ReAlE generates for each. If relation $t$ is a renaming of relation $s$, one possible parametrization for the embedding of $t$ is the one proposed in the proof of Theorem 4.4. Analysing multiple examples of relations $t$ representing renaming of $s$ in REL-ER, we observe that in all cases, the embedding of relations $t$ is the closest to the parametrization proposed in the proof compared to all other relation embeddings. We did the same analysis with the rest of the operations (and their respective theorems), and the same holds for all. The results in Table 4.8 show the breakdown performance for each operation: further evidence for learnability of the operations in ReAlE.

4.3.8 Conclusion

In this section, we introduce ReAlE for reasoning in knowledge hypergraphs. To design a powerful method, we build on the primitives of relational algebra, which is the calculus of relational models. We prove that ReAlE can represent the primitive relational algebra operations renaming, projection, selection, set union, and set difference, and is fully expressive. The results of our experiments on real and synthetic datasets are consistent with the theoretical findings.

4.4 Conclusion and Limitations

In this chapter, we study the problem of knowledge hypergraph completion. We show that converting knowledge hypergraphs to knowledge graphs and using knowledge graph embedding-based models is not effective. We propose real and synthetic datasets, baselines, evaluation metrics, and models (HSimplE, HypE, and ReAlE) for knowledge hypergraph completion. Despite being effective, the proposed models can only take one single entity in each position in a tuple. However, in some applications, a position should contain a set of entities. Another limitation of the proposed models is that they can only predict one entity at a time in a tuple. We need to extend these models to be able to predict multiple missing entities in a tuple.
Table 4.9: Knowledge hypergraph completion results on JF17K, FB-AUTO and M-FB15K for baselines and the proposed method for a fixed embedding dimension of 200, negative ratio of 10, and batch size of 128. Our method ReAIE (Small) outperforms the baselines on all datasets with these settings.

<table>
<thead>
<tr>
<th>Model</th>
<th>JF17K</th>
<th>FB-AUTO</th>
<th>M-FB15K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR</td>
<td>HIT@1</td>
<td>HIT@3</td>
</tr>
<tr>
<td>m-DistMult [40]</td>
<td>.463</td>
<td>.372</td>
<td>.510</td>
</tr>
<tr>
<td>m-CP [40]</td>
<td>.392</td>
<td>.303</td>
<td>.441</td>
</tr>
<tr>
<td>m-TransH [153]</td>
<td>.444</td>
<td>.370</td>
<td>.475</td>
</tr>
<tr>
<td>GETD [87]</td>
<td>.151</td>
<td>.104</td>
<td>.151</td>
</tr>
<tr>
<td>HSimplE [40]</td>
<td>.472</td>
<td>.378</td>
<td>.520</td>
</tr>
<tr>
<td>HypeE [20]</td>
<td>.494</td>
<td>.408</td>
<td>.538</td>
</tr>
<tr>
<td>G-MPNN [155]</td>
<td>.501</td>
<td>.425</td>
<td>.537</td>
</tr>
<tr>
<td>ReAIE (Small)</td>
<td>.530</td>
<td>.454</td>
<td>.563</td>
</tr>
</tbody>
</table>

Table 4.10: Knowledge hypergraph completion results for ReAIE on JF17K for different $\sigma$ (nonlinear function).

<table>
<thead>
<tr>
<th>Model</th>
<th>JF17K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR</td>
</tr>
<tr>
<td>ReAIE ($\sigma$ as exponent) (Ours)</td>
<td>.394</td>
</tr>
<tr>
<td>ReAIE ($\sigma$ as tanh) (Ours)</td>
<td>.512</td>
</tr>
<tr>
<td>ReAIE ($\sigma$ as sigmoid) (Ours)</td>
<td><strong>0.530</strong></td>
</tr>
</tbody>
</table>
Chapter 5

Implicit Graphs

The results presented in this chapter have been published in Fatemi et al. [41].

Having introduced novel architectures for explicitly graph-structured data in Chapters 3 and 4 of this thesis, Chapter 5 investigates how models with relational inductive biases - in particular graph neural networks - can be developed and applied to problems with implicit or hidden graph structures. Graph neural networks require a graph structure in the input but in some applications, there is no graph structure of nodes readily available. In such applications, there are node features available in the input, but there is no graph structure of nodes readily available. However, the assumption is that there is a latent structure connecting the nodes. If we can infer the latent structure, we will be able to use models such as graph neural networks and that can improve the accuracy of prediction tasks.

5.1 Introduction

The problem of learning with implicit graph structures arises in many real-world applications. For instance, when predicting whether the stock of different companies will rise or fall, we might not have access to the graph structure that tells us how different companies are related. However, experience has shown that when the stock of one company changes, the stocks of the other related companies also change [43]. We assume that this change is because of a latent structure connecting different companies. The aim here is to learn the latent structure to help us improve the accuracy of the task.

Graph representation learning has grown rapidly and found applications in domains where a natural graph of the data points (i.e., explicit graph structure) is available [20, 78]. Graph neural networks (GNNs) [125] (see Section 2.4 for a detailed description) have been a key component in the research in this area.

Node classification in a graph is defined as when we have an explicit graph structure with a set of nodes and edges and each node is accompanied by a set of features (or we can use feature extraction techniques
5.1. Introduction

to collect features for nodes). The goal of node classification is to predict labels (classes) for the nodes. In the semi-supervised node classification setup, a small number of nodes are labeled and a large number of nodes are unlabeled during training. GNNs have shown promising results for semi-supervised node classification when the available graph structure exhibits a high degree of homophily (i.e., connected nodes often belong to the same class) [172].

We study the applicability of GNNs to (semi-supervised) classification problems where a graph structure is not readily available. The existing approaches for (semi-supervised) classification (see Section 5.2 for a detailed discussion of the related work) with a hidden structure often take one of the following approaches. 1. **Similarity graph**: these approaches select a similarity metric and set the edge weight between two nodes to be their similarity. 2. **Latent graph learning**: these approaches learn the GNN parameters and a graph structure simultaneously. In both approaches, one main goal is to construct or learn a graph structure with a high degree of homophily with respect to the labels to aid the GNN classification. The latent graph learning approach often results in higher predictive performance compared to the former approach (see, e.g., [23, 47]).

We identify a supervision starvation problem in the latent graph learning approaches in which the edges between pairs of nodes that are far from labeled nodes receive insufficient supervision; this results in learning poor structures away from labeled nodes and hence poor generalization. We propose a solution for this problem by adopting a multi-task learning framework in which we supplement the classification task with a self-supervised task. The self-supervised task is based on the hypothesis that a graph structure that is suitable for predicting the node features is also suitable for predicting the node labels. It works by masking some input features (or adding noise to them) and training a separate GNN aiming at updating the adjacency matrix in such a way that it can recover the masked (or noisy) features. The task is generic and can be combined with several existing latent graph learning approaches.

In this chapter, we propose the **S**imultaneous **L**earning of **A**djacency and **GNN** **P**arameters with **S**elf-supervision, or SLAPS, that adopts the proposed self-supervised task. We provide a comprehensive experimental study on nine datasets (thirteen variations) of various sizes and from various domains and perform thorough analyses to show the merit of SLAPS. Our main contributions include:

- Identifying a supervision starvation problem for latent graph learning.
5.2. Related Work

- Proposing a solution for the identified problem through self-supervision.
- Developing SLAPS, a latent graph learning model that adopts the self-supervised solution.
- Providing comprehensive experimental results showing SLAPS substantially outperforms existing latent graph learning baselines from various categories on various benchmarks.
- Providing an implementation for latent graph learning that scales to graphs with hundreds of thousands of nodes.

5.2 Related Work

Existing methods that relate to this work can be grouped into the following categories (see Zhu et al. [176] for a full survey).

**Fully connected graph.** One approach for inferring a graph structure is to start with a fully connected graph and assign edge weights using the available meta-data or employ the GNN variants that provide weights for each edge via an attention mechanism [143, 165]. This approach has been used in computer vision [e.g., 136], natural language processing [e.g., 171], and few-shot learning [e.g., 49]. The space complexity of a fully connected graph grows rapidly (a graph with \( n \) nodes requires \( n^2 \) space for storing a fully connected graph) making such approaches applicable to small-sized graphs. There are studies trying to make improvements around computational and memory efficiency of creating a fully connected graphs as in [24, 84, 148]. Zhang et al. [166] propose to define local neighborhoods for each node and only assume that these local neighborhoods are fully connected. Their approach relies on an initial graph structure to define the local neighborhoods.

**Similarity graph.** Another approach for inferring a graph structure is to select a similarity metric and set the edge weight between two nodes to be their similarity [11, 122, 137]. To obtain a sparse structure with less space complexity, one may create a k-nearest neighbors (kNN) similarity graph and connect each node to only its \( k \) most similar nodes. Another approach for getting sparsity is to only connect pairs of nodes whose similarity surpasses some predefined threshold or do sampling on the graph. As an example, Gidaris and Komodakis [50] create a (fixed) kNN graph using the cosine similarity of the node features. Wang et al. [150] extend this
5.2. Related Work

idea by creating a fresh graph in each layer of the GNN based on the node embedding similarities in that layer. Instead of choosing a single similarity metric, Halcrow et al. [56] fuse several (potentially weak) measures of similarity. The quality of the predictions of these methods depends heavily on the choice of the similarity metric(s).

**Latent graph learning.** Instead of a similarity graph based on the initial features, one may use a graph generator with learnable parameters. Li et al. [90] create a fully connected graph based on a bilinear similarity function with learnable parameters. Franceschi et al. [47] learn a Bernoulli distribution for each possible edge and graph structures are created through sampling from these distributions. Yang et al. [159] update the input structure to increase homophily based on the labels and model predictions. [23] propose an iterative approach that iterates over projecting the nodes to a latent space and constructing an adjacency matrix from the latent representations multiple times. A common approach in this category is to learn a projection of the nodes to a latent space where node similarities correspond to edge weights or edge probabilities. Wu et al. [154] project the nodes to a latent space by learning weights for each of the input features. Cosmo et al. [27], Jiang et al. [68], Qasim et al. [114] use a multi-layer perceptron for projection. Yu et al. [163], Zhao et al. [170] use a GNN for projection; it uses the node features and an initial graph structure. Kazi et al. [79] create different graph structures in different layers by using separate GNN projectors, where the input to the GNN projector in a layer is the projected values and the generated graph structure from the previous layer. In our experiments, we compare with several approaches from this category.

**Leveraging domain knowledge.** In some applications, one may leverage domain knowledge to guide the model toward learning specific structures. For example, Johnson et al. [71] leverage abstract syntax trees and regular languages in learning graph structures of Python programs that aid reasoning for downstream tasks. Jin et al. [70] guide the structure learning for robustness to adversarial attacks through the domain knowledge that clean adjacency matrices are often sparse and low-rank and exhibit feature smoothness along the connected nodes. Other examples in this category include [67, 114]. In our work, we experiment with general-purpose datasets without access to domain knowledge.
Proposed method. Our model falls within the latent graph learning category. We supplement the training with a self-supervised objective to increase the amount of supervision in learning a structure. Our self-supervised task is inspired by, and similar to, the pre-training strategies for GNNs [63, 64, 69, 162, 173] (specifically, we adopt the multi-task learning framework of You et al. [162]), but it differs from this line of work as we use self-supervision for learning a graph structure whereas the above methods use it to learn better (and, in some cases, transferable) GNN parameters.

5.3 Proposed Method: SLAPS

SLAPS consists of four components: 1) generator, 2) adjacency processor, 3) classifier, and 4) self-supervision. Figure 5.1 illustrates these components. In the next three subsections, we explain the first three components. Then, we point out a supervision starvation problem for a model based only on these components. Then we describe the self-supervision component as a solution to the supervision starvation problem and the full SLAPS model.

5.3.1 Generator

The generator is a function $G : \mathbb{R}^{n \times f} \rightarrow \mathbb{R}^{n \times n}$ with parameters $\theta_G$ which takes the node features $X \in \mathbb{R}^{n \times f}$ as input and produces a matrix $\tilde{A} \in \mathbb{R}^{n \times n}$ as output. We consider the following two generators and leave experimenting with more sophisticated graph generators (e.g., [92, 95, 161]) and models with tractable adjacency computations (e.g., [25]) as future work.

Full parameterization (FP). For this generator, we have its parameters $\theta_G$ to be a matrix of size $n \times n$ i.e., $\theta_G \in \mathbb{R}^{n \times n}$. The generator function is defined as follows.

$$\tilde{A} = G_{FP}(X; \theta_G) = \theta_G$$

That is, the generator ignores the input node features and directly optimizes the adjacency matrix. FP is similar to the generator in [47] except that they treat each element of $\tilde{A}$ as the parameter of a Bernoulli distribution and sample graph structures from these distributions. FP is simple and flexible for learning any adjacency matrix but adds $n^2$ parameters which limits scalability and makes the model susceptible to overfitting.
5.3. Proposed Method: SLAPS

Figure 5.1: Overview of SLAPS. At the top, a generator receives the node features and produces a non-symmetric, non-normalized adjacency having (possibly) both positive and negative values (Section 5.3.1). The adjacency processor makes the values positive, symmetrizes, and normalizes the adjacency (Section 5.3.2). The resulting adjacency and the node features go into GNN\(_C\) which predicts the node classes (Section 5.3.3). At the bottom, some noise is added to the node features. The resulting noisy features and the generated adjacency go into GNN\(_{DAE}\) which then denoises the features (Section 5.3.5).

**MLP-kNN.** This generator has a multi-layer perceptron (MLP) followed by a k-nearest neighbors (kNN). The MLP takes the node features \(X\) as input and produces a matrix with updated node representations \(X'\) as output. The MLP is defined as the function \(\text{MLP} : \mathbb{R}^{n \times f} \rightarrow \mathbb{R}^{n \times f'}\). To obtain a sparse structure, we apply a kNN function to the output of the MLP. We create a kNN similarity graph by connecting each node to only its \(k\) most similar nodes. The kNN function is defined as \(\text{kNN} : \mathbb{R}^{n \times f'} \rightarrow \mathbb{R}^{n \times n}\). Parameters of this generator, \(\theta_G\), correspond to the weights of a multi-layer perceptron (MLP). Using this generator, \(\hat{A}\) is defined as follows.

\[
\hat{A} = G_{\text{MLP}}(X; \theta_G) = \text{kNN}(\text{MLP}(X; \theta_G))
\]

**Initialization and variants of MLP-kNN.** Let \(A_{kNN}\) represent an adjacency matrix created by applying a kNN function on the initial node features \(X\). One smart initialization for \(\theta_G\) is to initialize it in a way that the generator initially generates \(A_{kNN}\) (i.e., \(\hat{A} = A_{kNN}\) before training starts). This can be trivially done for the FP generator by initializing its parameters \(\theta_G\) to \(A_{kNN}\). For MLP-kNN, we consider two variants. For both variants, we have the following properties.
5.3. Proposed Method: SLAPS

- We keep the input dimension the same throughout the layers i.e., $f = f'$.
- We initialize the weight matrices $\theta_G$ with the identity matrix to ensure that the output of the MLP is initially the same as its input and the kNN graph created on the output is equivalent to $A_{kNN}$ (Alternatively, one may use other MLP variants but pre-train the weights to output $A_{kNN}$ before the main training starts).

The only difference between the two variants is that in one variant, hereafter referred to as MLP-D, we consider MLPs with diagonal weight matrices i.e., except the main diagonal, all other parameters in the weight matrices are zero. However, in the other variant, hereafter referred to simply as MLP, the weight matrices can be arbitrary. MLP-D can be thought of as assigning different weights to different features and then computing node similarities.

**Back-propagation through the kNN operation.** To back-propagate through the kNN operation, we followed the straight-through estimator proposed by Bengio et al. [12]. This strategy only flows gradients toward non-zero values in the output of a hard threshold function which is a kNN in our method. As discussed by Bengio et al. [12], this estimator is biased, but it works well because it enforces a prior that for a given input, most of the factors in the model are irrelevant and would be represented by zeros in the representation. Bengio et al. [12] also investigated multiplying the gradient by the derivative of the sigmoid, but they found that better results were obtained without multiplying the derivative of the sigmoid.

5.3.2 Adjacency Processor

The output of the generator $\hat{A}$ may have both positive and negative values and may be non-symmetric and non-normalized. We let $\mu$ be a function with a non-negative range applied element-wise on its input. $\mu(\hat{A})$ maps the output of the generator to non-negative values. The expression $\frac{1}{2}(\mu(\hat{A}) + \mu(\hat{A})^T)$ makes the resulting matrix $\mu(\hat{A})$ symmetric. To understand the reason for taking the mean of $\mu(\hat{A})$ and $\mu(\hat{A})^T$, assume $\hat{A}$ is generated by $G_{MLP}$. If $v_j$ is among the $k$ most similar nodes to $v_i$ and vice versa, then the strength of the connection between $v_i$ and $v_j$ will remain the same. However, if, say, $v_j$ is among the $k$ most similar nodes to $v_i$ but $v_i$ is not among the top
5.3. Proposed Method: SLAPS

k for \( v_j \), then taking the average of the similarities reduces the strength of the connection between \( v_i \) and \( v_j \). Finally, once we have a symmetric adjacency with non-negative values, we normalize \( \frac{1}{2}(\mu(A) + \mu(A)^T) \) by computing its degree matrix \( D \) and multiplying it from left and right to \( D^{-\frac{1}{2}} \). This is the same normalization as introduced in Equation 2.2 for symmetric matrices. We let \( A \) be defined as follows.

\[
A = \frac{1}{2} D^{-\frac{1}{2}} (\mu(A) + \mu(A)^T) D^{-\frac{1}{2}}
\]

In our experiments, when using an MLP generator, we let \( \mu \) be the ReLU [102] function applied element-wise on the elements of \( A \). When using the fully-parameterized (FP) generator, applying ReLU results in a gradient flow problem as any edge whose corresponding value in \( A \) becomes less than or equal to zero stops receiving gradient updates. For this reason, for FP we apply the ELU [26] function to the elements of \( A \) and then add a value of 1.

5.3.3 Classifier

The classifier is a function \( \text{GNN}_C : \mathbb{R}^{n \times f} \times \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times |C|} \) with parameters \( \theta_{\text{GNN}_C} \). It takes the node features \( X \) and the normalized adjacency \( A \) as input and provides for each node the probability for each class. \( n \) corresponds to the number of nodes, \( C \) corresponds to the classes, and \( |C| \) corresponds to the number of classes. We use a two-layer GCN (a GCN layer is defined in Equation 2.3) followed by a softmax function in the classifier. A two-layer GCN has parameters \( \theta_{\text{GNN}_C} = \{ W^{(1)}, W^{(2)} \} \). The classifier predicts labels \( Y \) for nodes as:

\[
\hat{Y} = \text{GNN}_C(A, X; \theta_{\text{GNN}_C}) = \text{softmax}(A \text{ReLU}(AX W^{(1)} W^{(2)})) \quad (5.1)
\]

where the softmax function is applied row-wise and is defined as

\[
\text{softmax}(H[i][j]) = \frac{\exp(H[i][j])}{\sum_{c \in C} \exp(H[i][c])}.
\]

Let \( \mathcal{Y}_L \) be the set of labeled node indices and \( Y \in \mathbb{R}^{n \times |C|} \) be a binary matrix indicating labels of the nodes. If node \( i \) is labeled (\( i.e., i \in \mathcal{Y}_L \)), \( Y[i] \) is a one-hot encoding of the label for the node (\( Y[i] \) is a vector with only one value of 1 indicating the label for the node and the rest of the values are zeros). If node \( i \) is not labeled (\( i.e., i \notin \mathcal{Y}_L \)), then \( Y[i] \) is a vector of zeros (\( Y[i] \) for \( i \notin \mathcal{Y}_L \) does not affect the loss function). The following
5.3. Proposed Method: SLAPS

Figure 5.2: Using a two-layer GCN, the predictions made for the labeled nodes are not affected by the dashed (starved) edge.

The equation shows the classification loss based on cross-entropy error over all labeled examples for the node classification task.

\[
\mathcal{L}_C = -\sum_{i \in Y_C} \sum_{c=1}^{C} Y[i][c] \times \log(\hat{Y}[i][c])
\]

5.3.4 Using only the first three components leads to supervision starvation

One may create a model using only the three components described so far corresponding to the top part of Figure 5.1. As we will explain here, however, this model may suffer severely from supervision starvation. The same problem also applies to many existing approaches for latent graph learning, as they can be formulated as a combination of variants of these three components.

Consider a scenario during training where two unlabeled nodes \(v_i\) and \(v_j\) are not directly connected to any labeled nodes according to the generated structure. Then, since a two-layer GCN makes predictions for the nodes based on their two-hop neighbors, the classification loss (i.e., \(\mathcal{L}_C\)) is not affected by the edge between \(v_i\) and \(v_j\) and this edge receives no supervision.\(^1\) Figure 5.2 provides an example of such a scenario. Let us call the edges that do not affect the loss function \(\mathcal{L}_C\) (and consequently do not receive supervision) as starved edges. These edges are problematic because although they may not affect the training loss, the predictions at the test time depend on these edges and if their values are learned without enough supervision, the model may make poor predictions at test. A natural question concerning

\(^1\)While using more layers may relatively alleviate this problem, deeper GCNs typically produce inferior results, e.g., due to oversmoothing [89, 108] – see Section 5.5.5 (Paragraph “Increasing the number of layers”) for empirical evidence.
5.3. Proposed Method: SLAPS

the extent of the problem caused by such edges is the proportion of starved edges. The following theorem formally establishes the extent of the problem for Erdős-Rényi graphs [36]; in Appendix C.1 we extend this result to the Barabási–Albert model [4] and scale-free networks [9]. An Erdős-Rényi graph with \( n \) nodes and \( m \) edges is a graph chosen uniformly at random from the collection of all graphs which have \( n \) nodes and \( m \) edges.

**Theorem 5.1** Let \( G(n, m) \) be an Erdős-Rényi graph with \( n \) nodes and \( m \) edges. Assume we have labels for \( q \) nodes selected uniformly at random. The probability of an edge being a starved edge with a two-layer GCN is equal to the following:

\[
(1 - \frac{q}{n})(1 - \frac{q}{n-1}) \prod_{i=1}^{2q} (1 - \frac{m - 1}{\binom{n}{2} - i})
\]

We defer the proof to Appendix C.1 To put the numbers from the theorem in perspective, let us consider three established benchmarks for semi-supervised node classification namely CORA, CITeseer, and PUBMED. For an Erdős-Rényi graph with similar statistics as the CORA dataset (\( n = 2708, m = 5429, q = 140 \)), the probability of an edge being a starved edge is 59.4% according to the above theorem. For CITeseer and PUBMED, this number is 75.7% and 96.7% respectively. While Theorem 5.1 is stated for Erdős-Rényi graphs, the identified problem also applies to natural graphs. For the original structures of CORA, CITeseer, and PUBMED, for example, 48.8%, 65.2%, and 91.6% of the edges are starved edges.

5.3.5 Self-supervision

One possible solution to the supervision starvation problem is to define a prior graph structure and regularize the learned structure toward it. This leads the starved edges toward the prior structure as opposed to neglecting them. The choice of the prior is important as it determines the inductive bias incorporated into the model. We define a prior structure based on the following hypothesis:

**Hypothesis 1** A graph structure that is suitable for predicting the node features is also suitable for predicting the node labels.

We first explain why the above hypothesis is reasonable for an extreme case that is easy to understand and then extend the explanation to the general case. Consider an extreme scenario where one of the node features is the same as the node labels. A graph structure that is suitable for predicting
this feature exhibits homophily for it. Because of the equivalence between this feature and the labels, the graph structure also exhibits homophily for the labels, so it is also suitable for predicting the labels. In the general (non-extreme) case, there may not be a single feature that is equivalent to the labels but a subset of the features may be highly predictive of the labels. A graph structure that is suitable for predicting this subset exhibits homophily for the features in the subset. Because this subset is highly predictive of the labels, the structure also exhibits a high degree of homophily for the labels, so it is also suitable for predicting the node labels.

Next, we explain how to design a suitable graph structure for predicting the features and how to regularize toward it. One could design such a structure manually (e.g., by handcrafting a graph that connects nodes based on the collective homophily between their individual features) and then penalize the difference between this prior graph and the learned graph. Alternatively, in this section, we take a learning-based approach based on self-supervision where we not only use the learned graph structure for the classification task, but also for denoising the node features. The self-supervised task encourages the model to learn a structure that is suitable for predicting the node features.

Our self-supervised task is based on denoising autoencoders [144]. Let $\text{GNN}_{DAE} : \mathbb{R}^{n \times f} \times \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times f}$ be a GNN with parameters $\theta_{\text{GNN}_{DAE}}$ that takes node features and a generated adjacency as input and provides updated node features with the same dimension as output. We train $\text{GNN}_{DAE}$ such that it receives a noisy version $\tilde{X}$ of the features $X$ as input and produces the denoised features $\hat{X}$ as output. Let $idx$ represent the indices corresponding to the elements of $X$ to which we have added noise, and $X_{idx}$ represent the values at these indices. During training, we minimize:

$$L_{DAE} = L(X_{idx}, \text{GNN}_{DAE}(\hat{X}, A; \theta_{\text{GNN}_{DAE}})_{idx})$$

where $A$ is the generated adjacency matrix and $L$ is a loss function. Here, $idx$ is defined with one of the following options.

- For datasets where the features consist of binary vectors, $idx$ consists of $r$ percent of the indices of $X$ whose values are 1 and $\eta$ percent of the indices whose values are 0, both selected uniformly at random in each epoch. Both $r$ and $\eta$ (corresponding to the negative ratio) are hyperparameters. In this case, we add noise by setting the 1s in the selected mask to 0s and $L$ is the binary cross-entropy loss.

- For datasets where the input features are continuous numbers, $idx$ consists of $r$ percent of the indices of $X$ selected uniformly at random.
5.3. Proposed Method: SLAPS

in each epoch. We add noise by either replacing the values at \( idx \) with 0 or by adding independent Gaussian noises to each of the features. In this case, \( L \) is the mean-squared error loss.

Note that the self-supervised task in Equation 5.3 is generic and can be added to different GNNs as well as latent graph learning models. It can be also combined with other techniques in the literature that encourage learning more homophilous structures or increase the amount of supervision. In our experiments, we test the combination of our self-supervised task (and kNN-GCN) with two techniques from the literature namely self-training [89] and AdaEdge [21]. These two techniques are explained below.

Self-training. Self-training helps the model “see” more labeled nodes. For self-training, we first train a model using the existing labels in the training set. Then, we use this model to make predictions for the unlabeled nodes that were not in the train, validation, or test sets. We consider the label predictions for the top \( \zeta \) most confident unlabeled nodes as ground truth labels and add them to the training labels. Finally, we train a model from scratch on the expanded set of labels. Here, \( \zeta \) is a hyperparameter tuned using the dev set.

AdaEdge. AdaEdge helps iteratively create graph structure with higher degrees of homophily. For AdaEdge, in the case of kNN-GCN, we first train a kNN-GCN model. Then we change the structure of the graph from the kNN graph to a new graph by following these steps: 1) add edges between nodes with the same class predictions if both prediction confidences surpass a threshold, 2) remove edges between nodes with different class predictions if both prediction confidences surpass a threshold. Then, we train a GCN model on the new structure and repeat the steps above to generate a new structure. We do this iteratively until a new structure does not provide a boost in performance on the validation set. For SLAPS, we follow a similar approach except that the initial model was a SLAPS model instead of a kNN-GCN model.

5.3.6 SLAPS

Our final model is trained to minimize the weighted sum of the classification loss \( L_C \) (defined in Equation 5.2) and the denoising autoencoder loss \( L_{DAE} \) (defined in Equation 5.3) as follows.

\[
\mathcal{L} = L_C + \lambda L_{DAE}
\]
5.4. Experimental Setup

Here, $\lambda$ is a hyperparameter controlling the relative importance of the two losses.

5.4 Experimental Setup

In this section, we report the baselines and the datasets used in our experiments. We defer the implementation details and the best hyperparameter settings for our model on all the datasets to Appendix C.2.

5.4.1 Baselines

We compare our proposal to several baselines with different properties. The first baseline is a multi-layer perceptron (MLP) which does not take the graph structure into account. We also compare against MLP-GAM* [134] which learns a fully connected graph structure and uses this structure to supplement the loss function of the MLP toward predicting similar labels for neighboring nodes. Our third baseline is label propagation (LP) [174], a well-known model for semi-supervised learning. Similar to [47], we also consider a baseline named $kNN$-GCN where we create a kNN graph based on the node feature similarities and feed this graph to a GCN; the graph structure remains fixed in this approach. We also compare with prominent existing latent graph learning models including LDS [47], GRCN [163], DGCNN [150], and IDGL [23]. In [23], another variant named IDGL-ANCH is also proposed that reduces time complexity through anchor-based approximation [96]. We compare against the base IDGL model because it does not sacrifice accuracy for time complexity, and because anchor-based approximation is model-agnostic and could be combined with other models too. We feed a kNN graph to the models requiring an initial graph structure. We also explore how adding self-training and AdaEdge impact the performance of kNN-GCN as well as SLAPS.

5.4.2 Datasets

We use three established benchmarks in the GNN literature namely cora, citeseer, and pubmed [127] as well as the ogbn-arxiv dataset [62] that is orders of magnitude larger than the other three datasets and is more challenging due to the more realistic split of the data into train, validation, and test sets. For these datasets, we only feed the node features to the models and not their original graph structure. Following Franceschi et al. [47] and Chen et al. [23], we also experiment with several classification (non-graph)
5.4. Experimental Setup

Table 5.1: Statistics of the datasets used in the experiments.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Classes</th>
<th>Features</th>
<th>Label rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>cora</td>
<td>2,708</td>
<td>5,429</td>
<td>7</td>
<td>1,433</td>
<td>0.052</td>
</tr>
<tr>
<td>citeseer</td>
<td>3,327</td>
<td>4,732</td>
<td>6</td>
<td>3,703</td>
<td>0.036</td>
</tr>
<tr>
<td>pubmed</td>
<td>19,717</td>
<td>44,338</td>
<td>3</td>
<td>500</td>
<td>0.003</td>
</tr>
<tr>
<td>ogbn-arxiv</td>
<td>169,343</td>
<td>1,166,243</td>
<td>40</td>
<td>128</td>
<td>0.537</td>
</tr>
<tr>
<td>wine</td>
<td>178</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>0.112</td>
</tr>
<tr>
<td>cancer</td>
<td>569</td>
<td>0</td>
<td>2</td>
<td>30</td>
<td>0.035</td>
</tr>
<tr>
<td>digits</td>
<td>1,797</td>
<td>0</td>
<td>10</td>
<td>64</td>
<td>0.056</td>
</tr>
<tr>
<td>20news</td>
<td>9,607</td>
<td>0</td>
<td>10</td>
<td>236</td>
<td>0.021</td>
</tr>
<tr>
<td>mnist</td>
<td>10,000</td>
<td>0</td>
<td>10</td>
<td>784</td>
<td>0.1, 0.2 and 0.3</td>
</tr>
</tbody>
</table>

datasets available in scikit-learn [110] including wine, cancer, digits, and 20news. Furthermore, following Jiang et al. [68], we also provide results on mnist [86].

For cora and citeseer, the LDS model uses the train data for learning the parameters of the classification GCN, half of the validation for learning the parameters of the adjacency matrix (in their bi-level optimization setup, these are considered as hyperparameters), and the other half of the validation set for early stopping and tuning the other hyperparameters. Besides experimenting with the original setups of these two datasets, we also consider a setup that is closer to that of LDS: we use the train set and half of the validation set for training and the other half of validation for early stopping and hyperparameter tuning. We name the modified versions cora390 and citeseer370 respectively where the number proceeding the dataset name shows the number of labels from which gradients are computed. We follow a similar procedure for the scikit-learn datasets.

Jiang et al. [68] show that learning a latent graph structure of the input examples can help with semi-supervised image classification. In particular, they create three versions of the mnist dataset each consisting of a randomly selected subset with 10,000 examples in total. The first version contains 1000, the second contains 2000, and the third version contains 3000 labeled examples for training. All three variants use an extra 1000 labels for validation. The other examples are used as test examples.

The statistics of the datasets used in the experiments can be found in Table 5.1.
5.5. Experiments

Table 5.2: Results of SLAPS and the baselines on established node classification benchmarks presenting accuracy. † indicates results have been taken from Franceschi et al. [47]. ‡ indicates results have been taken from Stretcu et al. [134]. Bold and underlined values indicate the best and the second-best mean performances respectively. OOM indicates out of memory. OOT indicates out of time (we allowed 24h for each run). NA indicates not applicable. Appendix C.2 provides implementation details of SLAPS and baselines.

<table>
<thead>
<tr>
<th>Model</th>
<th>CORA</th>
<th>CITESEER</th>
<th>CORA390</th>
<th>CITESEER370</th>
<th>PUBMED</th>
<th>OGBN-ARXIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>56.1 ± 1.6†</td>
<td>56.7 ± 1.7†</td>
<td>65.8 ± 0.4</td>
<td>67.1 ± 0.5</td>
<td>71.4 ± 0.0</td>
<td>51.7 ± 0.1</td>
</tr>
<tr>
<td>MLP-GAM*</td>
<td>70.7†</td>
<td>70.3†</td>
<td>–</td>
<td>–</td>
<td>71.9†</td>
<td>–</td>
</tr>
<tr>
<td>kNN-GCN</td>
<td>66.5 ± 0.4†</td>
<td>64.3 ± 1.3†</td>
<td>72.5 ± 0.5</td>
<td>71.8 ± 0.8</td>
<td>70.4 ± 0.4</td>
<td>49.1 ± 0.3</td>
</tr>
<tr>
<td>kNN-GCN + AdaEdge</td>
<td>67.4 ± 0.3</td>
<td>67.3 ± 0.8</td>
<td>71.3 ± 0.9</td>
<td>70.9 ± 0.7</td>
<td>67.3 ± 0.3</td>
<td>OOM</td>
</tr>
<tr>
<td>kNN-GCN + self-training</td>
<td>67.7 ± 1.0</td>
<td>68.8 ± 1.0</td>
<td>72.2 ± 0.4</td>
<td>71.8 ± 0.6</td>
<td>OOT</td>
<td>OOT</td>
</tr>
<tr>
<td>SLAPS (FP)</td>
<td>72.4 ± 0.4</td>
<td>70.7 ± 0.4</td>
<td>76.6 ± 0.4</td>
<td>73.1 ± 0.6</td>
<td>OOM</td>
<td>OOM</td>
</tr>
<tr>
<td>SLAPS (MLP)</td>
<td>72.8 ± 0.8</td>
<td>70.5 ± 1.1</td>
<td>75.3 ± 1.0</td>
<td>73.0 ± 0.9</td>
<td>74.4 ± 0.6</td>
<td>56.6 ± 0.1</td>
</tr>
<tr>
<td>SLAPS (MLP-D)</td>
<td>73.4 ± 0.3</td>
<td>72.6 ± 0.6</td>
<td>75.1 ± 0.5</td>
<td>73.9 ± 0.4</td>
<td>73.1 ± 0.7</td>
<td>52.9 ± 0.1</td>
</tr>
<tr>
<td>SLAPS (MLP) + AdaEdge</td>
<td>72.8 ± 0.7</td>
<td>70.6 ± 1.5</td>
<td>75.2 ± 0.6</td>
<td>72.6 ± 1.4</td>
<td>OOT</td>
<td>OOT</td>
</tr>
<tr>
<td>SLAPS (MLP) + self-training</td>
<td><strong>74.2 ± 0.5</strong></td>
<td><strong>73.1 ± 1.0</strong></td>
<td><strong>75.5 ± 0.7</strong></td>
<td><strong>73.3 ± 0.6</strong></td>
<td><strong>74.3 ± 1.4</strong></td>
<td>NA</td>
</tr>
</tbody>
</table>

5.5 Experiments

In this section, we report the key experiments and their results. The experiments in this section investigate the following questions:

- **Q1.** How effective is the proposed method? We answer this by comparing SLAPS to the state-of-the-art and obvious baselines.

- **Q2.** What is the effect of self-supervision in SLAPS? Does self-supervision help the supervision starvation problem?

- **Q3.** What if instead of no graph structure, we have a noisy graph in the input with added and removed edges?

- **Q4.** How is the quality of the learned adjacency by SLAPS? Is the graph homophilous?

- **Q5.** What is the effect of k in k-nearest neighbors (kNN) and the number of layers of GCN in SLAPS?
5.5. Experiments

Section 5.5.1 provides a comprehensive experimental study on 9 datasets with 13 variations of various sizes and from various domains (Q1). Section 5.5.2 investigates the effectiveness of self-supervision in SLAPS (Q2). Section 5.5.3 verifies whether self-supervision is helpful when a noisy structure is provided as input (Q3). Section 5.5.4 analyzes the learned adjacency by SLAPS and investigates the quality of the adjacency (Q4). Finally, Section 5.5.5 provides ablation studies for SLAPS on different values for k in kNN and also experiments with deeper GCNs (Q5). More experimental analyses are presented in Appendix C.3.

5.5.1 Comparative Results

The results of SLAPS and the baselines on our benchmarks are reported in Tables 5.2 and 5.3. We start by analyzing the results in Table 5.2 first. Starting with the baselines, we see that learning a fully connected graph in MLP-GAM* makes it outperform MLP. kNN-GCN significantly outperforms MLP on cora and citeseer but underperforms on pubmed and ogbn-arxiv. Furthermore, both self-training and AdaEdge improve the performance of kNN-GCN. This shows the importance of the similarity metric and the graph structure that is fed into GCN; a low-quality structure can harm model performance. LDS outperforms MLP but the fully parameterized adjacency matrix of LDS results in memory issues for pubmed and ogbn-arxiv. Furthermore, both self-training and AdaEdge improve the performance of kNN-GCN. This shows the importance of the similarity metric and the graph structure that is fed into GCN; a low-quality structure can harm model performance. LDS outperforms MLP but the fully parameterized adjacency matrix of LDS results in memory issues for pubmed and ogbn-arxiv. As for GRCN, it was shown in the original paper that GRCN can revise a good initial adjacency matrix and provide a substantial boost in performance. However, as evidenced by the results, if the initial graph structure is somewhat poor, GRCN’s performance becomes on par with kNN-GCN. IDGL is the best performing baseline.

In addition to the aforementioned baselines, we also experimented with GCN, GAT, and Transformer (encoder only) architectures applied on fully connected graphs. GCN always learned to predict the majority class. This is because, after one fully connected GCN layer, all nodes will have the same embedding and become indistinguishable. GAT also showed similar behavior. We believe this is because the attention weights are (almost) random at the beginning (due to random initialization of the model parameters) resulting in nodes becoming indistinguishable and GAT cannot escape from that state. The skip connections of Transformer helped avoid the problem observed for GCN and GAT and we were able to achieve better results (~ 40% accuracy on cora). However, we observed severe overfitting (even with very small models and with high dropout probabilities).

SLAPS consistently outperforms the baselines in some cases by large
5.5. Experiments

Table 5.3: Results on classification datasets presenting accuracy. † indicates results have been taken from Franceschi et al. [47]. Bold and underlined values indicate best and the second-best mean performances respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>WINE</th>
<th>CANCER</th>
<th>DIGITS</th>
<th>20NEWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>96.1 ± 1.0</td>
<td>95.3 ± 0.9</td>
<td>81.9 ± 1.0</td>
<td>30.4 ± 0.1</td>
</tr>
<tr>
<td>kNN-GCN</td>
<td>93.5 ± 0.7</td>
<td>95.3 ± 0.4</td>
<td><strong>95.4 ± 0.4</strong></td>
<td>46.3 ± 0.3</td>
</tr>
<tr>
<td>LDS</td>
<td><strong>97.3 ± 0.4†</strong></td>
<td>94.4 ± 1.9†</td>
<td>92.5 ± 0.7†</td>
<td>46.4 ± 1.6†</td>
</tr>
<tr>
<td>IDGL</td>
<td>97.0 ± 0.7</td>
<td>94.2 ± 2.3</td>
<td>92.5 ± 1.3</td>
<td>48.5 ± 0.6</td>
</tr>
<tr>
<td>SLAPS (FP)</td>
<td>96.6 ± 0.4</td>
<td>94.6 ± 0.3</td>
<td>94.4 ± 0.7</td>
<td>44.4 ± 0.8</td>
</tr>
<tr>
<td>SLAPS (MLP)</td>
<td>96.3 ± 1.0</td>
<td><strong>96.0 ± 0.8</strong></td>
<td>92.5 ± 0.7</td>
<td><strong>50.4 ± 0.7</strong></td>
</tr>
<tr>
<td>SLAPS (MLP-D)</td>
<td>96.5 ± 0.8</td>
<td><strong>96.6 ± 0.2</strong></td>
<td>94.2 ± 0.1</td>
<td>49.8 ± 0.9</td>
</tr>
</tbody>
</table>

Margins. Among the generators, the winner is dataset-dependent with MLP-D mostly outperforming MLP on datasets with many features and MLP outperforming on datasets with small numbers of features. Using the software that was publicly released by the authors, the baselines that learn a graph structure fail on ogbn-arxiv; our implementation, on the other hand, scales to such large graphs. Adding self-training helps further improve the results of SLAPS. Adding AdaEdge, however, does not seem effective, probably because the graph structure learned by SLAPS already exhibits a high degree of homophily (see Section 5.5.4).

In Table 5.3, we only compared SLAPS with the best performing baselines from Table 5.2 (kNN-GCN, LDS and IDGL). We also included an MLP baseline for comparison. On three out of four datasets, SLAPS outperforms the LDS and IDGL baselines. For the DIGITS dataset, interestingly kNN-GCN outperforms the learning-based models. This could be because the initial kNN structure for this dataset is already a good structure. Among the datasets on which we can train SLAPS with the FP generator, 20NEWS has the largest number of nodes (9,607 nodes). In this dataset, we observed that an FP generator suffers from overfitting and produces weaker results compared to other generators due to its large number of parameters.

Following Jiang et al. [68], we also conduct experiments on the three variants of the MNIST dataset to measure the performance of SLAPS. We compare against GLCN [68] as well as the baselines in the GLCN paper including manifold regularization [11], label propagation, deep walk [112],

\[^2\text{We note that IDGL-ANCH also scales to ogbn-arxiv.}\]
5.5. Experiments

Table 5.4: Classification accuracy results on the MNIST dataset with low label rates as proposed by Jiang et al. [68]. Bold values indicate best mean performances. Underlined values indicate second best mean performance. All the results for baseline have been taken from [68].

<table>
<thead>
<tr>
<th>Model</th>
<th>MNIST 1000</th>
<th>MNIST 2000</th>
<th>MNIST 3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManiReg</td>
<td>92.74 ± 0.3</td>
<td>93.96 ± 0.2</td>
<td>94.62 ± 0.2</td>
</tr>
<tr>
<td>LP</td>
<td>79.28 ± 0.9</td>
<td>81.91 ± 0.8</td>
<td>83.45 ± 0.5</td>
</tr>
<tr>
<td>DeepWalk</td>
<td>94.55 ± 0.3</td>
<td>95.04 ± 0.3</td>
<td>95.34 ± 0.3</td>
</tr>
<tr>
<td>GCN</td>
<td>90.59 ± 0.3</td>
<td>90.91 ± 0.2</td>
<td>91.01 ± 0.2</td>
</tr>
<tr>
<td>GAT</td>
<td>92.11 ± 0.4</td>
<td>92.64 ± 0.3</td>
<td>92.81 ± 0.3</td>
</tr>
<tr>
<td>GLCN</td>
<td>94.28 ± 0.3</td>
<td>95.09 ± 0.2</td>
<td>95.46 ± 0.2</td>
</tr>
<tr>
<td>SLAPS</td>
<td>94.66 ± 0.2</td>
<td>95.35 ± 0.1</td>
<td>95.54 ± 0.0</td>
</tr>
</tbody>
</table>

Graph convolutional networks (GCN), and graph attention networks (GAT). The results are reported in Table 5.4. From the results, it can be viewed that SLAPS outperforms GLCN and all the other baselines on the 3 variants. Compared to GLCN, on the three variants SLAPS reduces the error by 7%, 5%, and 2% respectively, showing that SLAPS can be more effective when the labeled set is small and providing more empirical evidence for Theorem 5.1.

5.5.2 The Effectiveness of Self-supervision

This section investigates the effectiveness of self-supervision in SLAPS.

Learning a structure only using self-supervision. To provide more insight into the value provided by the self-supervision task and the generalizability of the adjacency learned through this task, we conduct experiments with a variant of SLAPS named SLAPS$_{2s}$ that is trained in two stages. We first train the GNN$_{DAE}$ model by minimizing $L_{DAE}$ described in in Equation 5.3. Recall that $L_{DAE}$ depends on the parameters $\theta_G$ of the generator and the parameters $\theta_{GNN_{DAE}}$ of the denoising autoencoder. After every $t$ epochs of training, we fix the adjacency matrix, train a classifier with the fixed adjacency matrix, and measure classification accuracy on the validation set. We select the epoch that produces the adjacency providing the best validation accuracy for the classifier. Note that in SLAPS$_{2s}$, the adjacency matrix only receives gradients from the self-supervised task in Equation 5.3.
5.5. Experiments

Figure 5.3: Comparing SLAPS, SLAPS$_2$, and kNN-GCN on cora with different generators. Although SLAPS$_2$ does not use the node labels in learning an adjacency matrix, it outperforms kNN-GCN. SLAPS outperforming SLAPS$_2$ indicates that learning the adjacency using both self-supervision and the task-specific node labels results in higher predictive accuracy.

Figure 5.3 shows the performance of SLAPS and SLAPS$_2$ on cora and compares them with kNN-GCN. Although SLAPS$_2$ does not use the node labels in learning an adjacency matrix, it outperforms kNN-GCN (8.4% improvement when using an FP generator). With an FP generator, SLAPS$_2$ even achieves competitive performance with SLAPS; this is mainly because FP does not leverage the supervision provided by GCN$_C$ toward learning generalizable patterns that can be used for nodes other than those in the training set. These results corroborate the effectiveness of the self-supervision task for learning an adjacency matrix. Besides, the results show that learning the adjacency using both self-supervision and the task-specific node labels results in higher predictive accuracy.

The value of $\lambda$. Figure 5.4 shows the performance of SLAPS$^3$ on cora and citeseer with different values of $\lambda$ ($\lambda$ is a hyperparameter controlling the importance of the self-supervised task). When $\lambda = 0$, corresponding to removing self-supervision, the model performance is somewhat poor. As soon as $\lambda$ becomes positive, both models see a large boost in performance showing that self-supervision is crucial to the high performance of SLAPS. Increasing $\lambda$ further provides larger boosts until it becomes so large that the self-supervision loss dominates the classification loss and the performance.

---

$^3$The generator used in this experiment is MLP; other generators produced similar results.
5.5. Experiments

Figure 5.4: The performance of SLAPS with MLP graph generator as a function of $\lambda$. SLAPS with $\lambda = 0$ corresponds to removing self-supervision. The results indicate that with removing self-supervision, the model performance is somewhat poor. As soon as $\lambda$ becomes positive, both models see a large boost in performance showing that self-supervision is crucial to the high performance of SLAPS.

deteriorates. Note that with $\lambda = 0$, SLAPS with the MLP generator becomes a variant of the model proposed by Cosmo et al. [27], but with a different similarity function.

Is self-supervision actually solving the supervision starvation problem? In Figure 5.4, we showed that self-supervision is key to the high performance of SLAPS. Here, we examine if this is because self-supervision indeed addresses the supervision starvation problem. For this purpose, we compared SLAPS with and without self-supervision on two groups of test nodes on cora: 1) those that are not connected to any labeled nodes after training, and 2) those that are connected to at least one labeled node after training. The nodes in group one have a high chance of having starved edges. We observed that adding self-supervision provides 38.0% improvement for the first group and only 8.9% improvement for the latter. Since self-supervision mainly helps with nodes in group 1, this provides evidence that self-supervision is an effective solution to the supervision starvation problem.

The effect of the training set size. According to Theorem 5.1, a smaller $q$ (corresponding to the training set size) results in more starved edges in
5.5. Experiments

Each epoch. To explore the effect of self-supervision as a function of \( q \), we compared SLAPS with and without supervision on CORA and CITESEER while reducing the number of labeled nodes per class from 20 to 5. We used the FP generator for this experiment. With 5 labeled nodes per class, adding self-supervision provides 16.7% and 22.0% improvements on CORA and CITESEER respectively, which is substantially higher than the corresponding numbers when using 20 labeled nodes per class (10.0% and 7.0% respectively). This provides empirical evidence for Theorem 5.1. Note that the results on CORA390 and CITESEER370 datasets provide evidence that the self-supervised task is effective even when the label rate is high.

5.5.3 Experiments with Noisy Graphs

The performance of GNNs highly depends on the quality of the input graph structure and deteriorates when the graph structure is noisy [see e.g., 29, 46, 177]. So far, we have investigated the effectiveness of SLAPS when no graph structure is available in the input. Here, we verify whether self-supervision is also helpful when a noisy structure is provided as input. Toward this goal, we experiment with CORA and CITESEER and provide noisy versions of the input graph as input. The provided noisy graph structure is used only for initialization; it is then further optimized by SLAPS. We perturb the graph structure by removing \( \rho \) percent of the edges in the original structure (selected uniformly at random). We calculate the number of removed edges and add the same number of edges to the graph with random weights. Figure 5.5 shows the performance of SLAPS with and without self-supervision (\( \lambda = 0 \) corresponds to no supervision). We also report the results of vanilla GCN on these perturbed graphs for comparison. It can be viewed that self-supervision consistently provides a boost in performance, especially for higher values of \( \rho \).

5.5.4 Analyses of the Learned Adjacency

In this section, we analyze the learned adjacency by SLAPS in two ways. First, we experiment with a noisy graph to investigate whether SLAPS can recover the original graph. Second, we measure the homophily in the graph learned by SLAPS.

Noisy graphs. Following the experiment in Section 5.5.3 we compared the learned and original structures by measuring the number of random
5.5. Experiments

Figure 5.5: Performance comparison when noisy graphs are provided as input ($\rho$ indicates the percentage of perturbations). The results indicate that self-supervision consistently provides a boost in performance, especially for higher values of $\rho$.

Homophily. As discussed, a properly learned graph for semi-supervised classification with GNNs exhibits high homophily. To verify the quality of the learned adjacency with respect to homophily, for every pair of nodes in the test set, we compute the odds of the two nodes sharing the same label as a function of the normalized weight of the edge connecting them. Figure 5.6 represents the odds for different weight intervals (recall that $A$ is row and column normalized). For both CORA and CITESEER, nodes’ connected with higher edge weights are more likely to share the same label compared to nodes with lower or zero edge weights. Specifically, when $A_{ij} \geq 0.1$, $v_i$ and $v_j$ are almost 2.5 and 2.0 times more likely to share the same label on CORA and CITESEER respectively. This shows that the learned edge weights are highly predictive of the same classness.
5.5. Experiments

Figure 5.6: The odds of two nodes in the test set sharing the same label as a function of the edge weights learned by SLAPS. For both cora and citeseer, nodes that are connected with higher edge weights are more likely to share the same label compared to nodes with lower or zero edge weights. This shows that the learned edge weights are highly predictive of the same classness.

5.5.5 Ablation Studies

Importance of $k$ in kNN. Figure 5.7 shows the performance of SLAPS on cora for three graph generators as a function of $k$ in kNN. For all three cases, the value of $k$ plays a major role in model performance. The FP generator is the least sensitive because, in FP, $k$ only affects the initialization of the adjacency matrix but then the model can change the number of neighbors of each node. For MLP and MLP-D, however, the number of neighbors of each node remains close to $k$ (but not necessarily equal as the adjacency processor can add or remove some edges) and the two generators become more sensitive to $k$. For larger values of $k$, the extra flexibility of the MLP generator enables removing some of the unwanted edges through the function $\mu$ or reducing the weights of the unwanted edges resulting in MLP being less sensitive to large values of $k$ compared to MLP-D.

Increasing the number of layers. In Section 5.3.4, we described how some edges may receive no supervision during latent graph learning. We pointed out that while increasing the number of layers of the GCN may alleviate the problem to some extent, deeper GCNs typically provide inferior
results due to issues such as oversmoothing [see, e.g., 89, 108]. We empirically tested deeper GCNs for latent graph learning to see if simply using more layers can obviate the need for the proposed self-supervision. Specifically, we tested SLAPS without self-supervision (i.e., $\lambda = 0$) with 2, 4, and 6 layers on cora. We also added residual connections that have been shown to help train deeper GCNs [88]. The accuracies for 2, 4, and 6-layer models are 66.2%, 67.1%, and 55.8% respectively. It can be viewed that increasing the number of layers from 2 to 4 provides an improvement. This might be because the benefit provided by a 4-layer model in terms of alleviating the starved edge problem outweighs the increase in oversmoothing. However, when the number of layers increases to 6, the oversmoothing problem outweighs and the performance drops significantly. Further increasing the number of layers resulted in even lower accuracies.

5.6 Limitations

In this section, we discuss some of the limitations of the proposed model. Firstly, in cases where nodes do not have input features but an initial noisy structure of the nodes is available, our self-supervised task cannot be readily applied. One possible solution is to first run an unsupervised node embedding model such as DeepWalk [112] to obtain node embeddings, then treat these embeddings as node features and run SLAPS. Secondly, the FP graph
generator is only applicable to the transductive setting (can only be used for a single fixed graph). This is because FP directly optimizes the adjacency matrix. However, our other two graph generators (MLP and MLP-D) can be applied in the inductive setting where the graph is not fixed (e.g., in evolving graphs).

5.7 Conclusion

We proposed SLAPS: a model for learning the parameters of a graph neural network and a graph structure of the nodes’ connectivities simultaneously from data. We identified a supervision starvation problem that emerges for graph structure learning, especially when training data is scarce. We proposed a solution to the supervision starvation problem by supplementing the training objective with a self-supervised task. We showed the effectiveness of our model through a comprehensive set of experiments and analyses.
Chapter 6

Conclusion and Future Work

In the first part of this thesis, we investigated how we can exploit explicit structures of graphs when making a prediction. In such cases, the data is in the form of entities with relations among the entities given. For such structures, in Chapter 3, we propose a novel model for learning embeddings for entities and relations such that they obey taxonomies. We showed how enforcing given subsumptions can lead to better performance for link prediction in knowledge graphs. In Chapter 4, we further investigated the knowledge hypergraph completion problem. We introduced datasets, baselines, novel methods, and metrics for link prediction in knowledge hypergraphs. We study the connections between knowledge hypergraphs and relational algebra and show how introducing a model based on relational algebra operations can give us improvement on benchmarks.

In Chapter 5, we extended the applicability of graph learning to applications where the assumption is that there is a latent structure between the entities but that is not given to us. For such implicit structures, we propose a model to learn the parameters of a prediction model and the latent structure simultaneously from the data. We identified a supervision starvation problem in existing models and proposed a solution to the supervision starvation based on self-supervision.

This work opens up several directions for future research. The following provides an overview of these directions.

6.1 Future Work for Learning with Explicit Structures

Incorporating Ontological Information. In Chapter 3, we proposed a model that can learn embeddings for entities and relationships while enforcing subproperties into embeddings. One possible extension for SimplE+ is to be able to further incorporate ontological background information and rules such as range information, symmetry, transitivity, domain, etc. As an example, range rules are in the form of \(\forall h, t \in \mathcal{E}^* : r(h, t) \rightarrow c_1(t, true)\) which
enforces type \( c_1 \) for tail of relation \( r \), e.g., if we know \( Sam \) has a master’s degree from \( UBC \), we can infer that \( UBC \) is an educational institution.

**Relational hypergraphs for natural language processing (NLP).**
We studied the theory and application of knowledge hypergraphs. One possible direction is to use the learned ideas in knowledge hypergraphs for NLP applications. In natural language, sentences can have an arbitrary number of words and therefore can be modeled using relational hyperedges. Different tasks in NLP can be then mapped to their graph variant. For instance, a corpus of facts can be used as a knowledge hypergraph and tasks that use a corpus e.g., question answering [93] can be mapped to link prediction in the knowledge hypergraph.

**Combining relational learning and set learning.** Current knowledge graph and hypergraph completion methods can handle tuples \( r(e_1, \ldots, e_n) \) with only one entity at each position \( k \) with \( 1 \leq k \leq n \) in the tuple. However, there might be cases with a set of entities at some of the positions in the tuple e.g., “Alex bought potatoes, tomatoes, and bananas from Walmart”. One workaround to handle a tuple \( r(e_1, \ldots, \{e_{k1}, \ldots, e_{ki}\}, \ldots, e_n) \) with a set of entities at position \( k \) is to convert it to a set of tuples \( r(e_1, \ldots, e_{k1}, \ldots, e_n), \ldots, r(e_1, \ldots, e_{ki}, \ldots, e_n) \) but this workaround loses this information that these set of tuples all true together. Another method to handle the tuples is to reify them to a set of tuples by adding a new entity \( e' \). The set of tuples with reification will be \( r(e', e_1, \ldots, e_{k1}, \ldots, e_n), \ldots, r(e', e_1, \ldots, e_{ki}, \ldots, e_n) \). A future direction is to explore current methods that can convert tuples with sets in some positions to tuples with only one entity at a position and evaluate their performance and devise new methods if current conversion methods fail. Ideas in learning with sets [e.g., 164] can be combined with ideas in knowledge hypergraphs to make such models applicable to a wide range of applications.

**Relational representation learning at scale.** Link prediction task in knowledge hypergraphs can be defined as predicting multiple entities at the same time as in \( r(e_0, ?, ?) \). One example of this question can be “where and to whom did Justin Trudeau marry?” as in the following tuple:

\[
MarriedTo(JustinTrudeau, ?, ?)
\]

In which we need to output 2 entities corresponding to whom and location to complete this partial tuple. One way of outputting \( m \) entities at once
that gives the highest probability in a partial tuple is iterating through all possible combinations of entities of which there are $|\mathcal{E}|^m$ with $|\mathcal{E}|$ as the number of entities in the knowledge hypergraph. This is not scalable to partial tuples with relations of high arities and many entities missing. Therefore, one open problem, that mainly arises in knowledge hypergraphs, is how to efficiently predict the missing entities in a given partial tuple. To the best of our knowledge, the only work in this line is proposed in [169], which uses indexing methods to filter entities that are not probable to appear together. Their proposed indexing method is based on heuristics that filter entities that never appeared in the same tuple in the train set. We plan to investigate a method that can learn what tuples of entities are a probable answer to the question instead of filtering the pairs that did not appear in train time.

**Incorporating negative examples.** Current datasets used as benchmarks for knowledge graph completion contain only positive tuples. To turn the link prediction problem into a classification problem with positive and negative examples, for each positive tuple, $m$ negative tuples are created based on a negative sampling strategy. The number of negative examples generated per positive example introduces a bias for models in terms of the prior probability of tuples being true. Since the number $m$ does not typically reflect the true prior probability of tuples, current knowledge graph embedding methods output uncalibrated probabilities. Current embedding-based models for knowledge graph completion use a probabilistic loss such as log-likelihood for training but the evaluation process is ranking-based such as MRR and HIT@K. One future direction is to create a ground truth of facts containing both positive and negative examples. This set is probably difficult to construct for both train and test but it is feasible to create one as a test set. We can use such a test set for evaluating various knowledge graph completion methods based on probabilistic metrics instead of ranking-based methods. After comparing current models on a ground truth containing both positive and negative examples, one possible future work is to look further into training approaches that encourage more calibrated probabilities.

**Beyond ranking.** Currently, knowledge (hyper)graphs are coupled with ranking-based evaluation metrics. That is, for each missing entity in the tuple, they rank the entities from the most probable to the least probable to fill the blank. However, in some applications, we are not interested in the
6.2. Future Work for Learning with Implicit Structures

ranking of the entities but in the probability of a tuple. For instance, we are interested to find out the probability of the tuple

\[ \text{LocatedIn}(\text{Berlin, France}), \]

indicating the probability of city \textit{Berlin} being located in \textit{France}. Being able to extend the evaluation metrics beyond ranking and use probabilistic metrics will enable the use of knowledge (hyper)graphs for more tasks.

6.2 Future Work for Learning with Implicit Structures

Implicit relations for temporal and multi-modal data. In Chapter 5, we proposed learning a static graph structure from data. However, in some applications with temporal data, there are multiple graph structures each associated with one timestamp. For instance, in dynamic simulation [82], each timestamp refers to a different graph structure that is obtained by adding or removing edges from the previous timestamp. Besides temporal data, there might be other forms of data available to be exploited for learning a graph structure. For instance, in some applications, data is available across a wide range of modalities, from visual data in the form of images and video, natural language data in the form of text, audio data such as sounds, as well as other forms of sensory data such as smell forms of data might be available to be used for learning a graph structure. Being able to learn the latent structure of the world in the presence of multi-modal data is a potential extension to the proposed model.

Beyond symmetry. Most of the models developed for learning with an implicit graph use a kNN function. The kNN algorithm uses a similarity metric to find the k-nearest neighbors for each node and most similarity metrics are symmetric \textit{e.g.}, distance metrics. The adjacency matrix generated by kNN is not necessarily symmetric as one node can be among the top \( k \) similar nodes of the other node but not the other way around. However, our experiments in Appendix [C.3.1] shows that symmetrization of the output of the kNN improves the result. Our hypothesis for why symmetrization improves the results is that the similarity metric used in the kNN is symmetric and therefore symmetrization of the output is more compatible with the similarity metric in kNN. Models learning a symmetric graph structure can only learn symmetric edges (edges with no direction). One future direction
6.3. Combining Explicit and Implicit Graph Structures

is to use a non-symmetric similarity metric so that the edges in the graph structure can have direction and therefore be non-symmetric.

**Implicit multi-relations.** The latent structure learned in SLAPS is a graph with only one type of relation. However, some graphs in the world are more complex and contain more than one type of relations. e.g., knowledge graphs often have a various number of relations. Being able to learn a graph structure with multiple types of relations can lead to more realistic and effective results in some applications with complex latent graphs.

**Interpretability.** Latent graph learned with SLAPS boosts the performance compared to various baselines. However, the learned graph is not interpretable. In some applications, interpretability [94] is of high importance, and being able to study the learned graph and investigate what it has learned is essential. Learning interpretable graphs are a future direction.

6.3 Combining Explicit and Implicit Graph Structures

The structure in some real-world applications is not completely explicit or completely implicit: the graph is partially given and partially latent. All the challenges and future directions discussed for explicit and implicit graph structures discussed so far should be combined in one system to be able to reflect the partially given graph.
Bibliography


[10] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti,
Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


[94] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

Bibliography


Bibliography


Bibliography


[160] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with freebase. In *Proceedings of
Bibliography


Appendix A

Proofs for Chapter 3

Theorem 3.1 ComplEx cannot enforce non-trivial subsumptions.

Proof Assume the non-trivial subsumption \( \forall h, t \in E^* : r(h, t) \rightarrow s(h, t) \). Based on the subsumption, we have \( \phi_\theta(s(h, t)) \geq \phi_\theta(r(h, t)) \), and there are entities \( a, b \in E^* \) such that \( \phi_\theta(s(a, b)) > \phi_\theta(r(a, b)) \). Let \( a' \) be an entity such that \( a' = -a \). Then \( \phi_\theta(s(a', b)) = 1 - \phi_\theta(s(a, b)) \) and \( \phi_\theta(r(a', b)) = 1 - \phi_\theta(r(a, b)) \), so \( \phi_\theta(s(a', b)) < \phi_\theta(r(a', b)) \), a contradiction to the subsumption we assumed.

Theorem 3.2 ComplEx-NNE+AER cannot satisfy its constraints and be fully expressive if symmetry constraints are allowed.

Proof In ComplEx a relation \( r \) is symmetric for all possible entities if and only if \( \text{Im}(r) = 0 \) (Trouillon et al. [140], Section 3). In order to satisfy constraints for \( \forall h, t \in E : r(h, t) \rightarrow s(h, t) \), Ding et al. [33] assign \( \text{Im}(r) = \text{Im}(s) \). Therefore, if relation \( r \) is symmetric, it enforces relation \( s \) to be symmetric too which is not generally true. As a counter-example, \( r \) might be the married_to relation, which is symmetric (so the \( \text{Im}(\text{married_to}) = 0 \)), but \( s \) is the knows relation, and \( \forall h, t \in E : \text{married_to}(h, t) \rightarrow \text{knows}(h, t) \) is true in real-world, but setting the \( \text{Im}(\text{knows}) = \text{Im}(\text{married_to}) \) will imply knows is symmetric, which is not true (for instance many people know celebrities but celebrities do not know those many people).

Theorem 3.3 SimplE cannot enforce non-trivial subsumptions.

Proof Consider \( \forall h, t \in E^* : r(h, t) \rightarrow s(h, t) \) as a non-trivial subsumption. So we have \( \phi_\theta(s(h, t)) \geq \phi_\theta(r(h, t)) \), and there are entities \( a, b \in E^* \) such that \( \phi_\theta(s(a, b)) > \phi_\theta(r(a, b)) \). Let \( a' \) be an entity such that \( a' = -a \). Then \( \phi_\theta(s(a', b)) = 1 - \phi_\theta(s(a, b)) \) and \( \phi_\theta(r(a', b)) = 1 - \phi_\theta(r(a, b)) \), so \( \phi_\theta(s(a', b)) < \phi_\theta(r(a', b)) \) a contradiction to the subsumption we assumed.
Theorem 3.4 (Expressivity) For any truth assignment over entities $\mathcal{E}$ and relations $\mathcal{R}$ containing $|\tau'|$ true triples, there exists a SimplE$^+$ model with embeddings vectors of size $\min(|\mathcal{E}| \times |\mathcal{R}| + 1, |\tau'| + 1)$ that represent the assignment.

**Proof** Assume $r_i$ is the $i$-th relation in $\mathcal{R}$ and $e_j$ is the $j$-th entity in $\mathcal{E}$. For a vector $a$ we define $(a)_i$ as the $i$-th element of $a$. We define $(rrr^+_i)_n = 1$ if $n \div |\mathcal{E}| = i$ except the last element $(rrr^+_1)_|\mathcal{E}||\mathcal{R}| = -1$, and for each entity $s_i$ we define $(e^+_j)_n = 1$ if $n \mod |\mathcal{E}| = j$ or $n = |\mathcal{E}| \times |\mathcal{R}|$ and 0 otherwise. In this setting, for each $r_i$ and $e_j$ product of $rrr^+_i$ and $eee^+_i$ is 0 everywhere except for the element at $(i \times |\mathcal{E}| + j)$ and the last element in the embeddings. In order for the triple $r_i(e_j, e_k)$ to hold, we define $(eee^-_i)$ to be a vector where all elements are 0 except the $(i \times |\mathcal{E}| + j)$-th element which is 2. This proves that SimplE$^+$ is fully expressive with the bound of $|\mathcal{E}| \times |\mathcal{R}| + 1$ for size of the embeddings.

We use induction to prove the bound $|\tau'| + 1$. Let $|\tau'| = 0$ (base of induction). We can have embedding vectors of size 1 for each entity and relation, setting the value for entities to 1 and to relations to -1. Then $\odot(r^+_i, h^+, t^+) + \odot(r^-_i, t^-, h^-)$ is negative for every entities $h$ and $t$ and relation $r$. So there exists an assignment of size 1 that represent this ground truth.

Let’s assume for any ground truth where $|\tau'| = n - 1$, there exists an assignment of values to embedding vectors of size $n$ that represent the ground truth (assumption of induction). We must prove for any ground truth where $|\tau'| = n$, there exist an assignment of values to embedding vectors of size $n + 1$ that represent this ground truth.

Let $r(h, t)$ be one of the $n$ true facts. Consider a modified ground truth which is identical to the ground truth with $n$ true facts, except that $r(h, t)$ is assigned false. The modified ground truth has $n - 1$ true facts and based on the assumption of the induction, we can represent it using some embedding vectors of size $n$. Let $q = \odot(r^+, h^+, t^+) + \odot(r^-, t^-, h^-)$. We add an element to the end of all embedding vectors and set it to 0. This increases the vector size to $n + 1$ but does not change any scores. Then we set $h$ to 1, $r$ to 1 and $t$ to $q + 1$. This ensures this triple is true for the new vectors, and no other probability of triple is affected.
Appendix A. Proofs for Chapter 3

Theorem 3.5 (Subsumption) SimplE+ guarantees subsumption using inequality constraints.

Proof Assume \( \forall h, t \in \mathcal{E}^* : r(h, t) \rightarrow s(h, t) \) as a non-trivial subsumption. As legal entity embeddings in SimplE+ have non-negative elements, by adding the element-wise inequality constraint \( s \geq r \), we force \( \phi_{\theta}(s(h, t)) \geq \phi_{\theta}(r(h, t)) \) for all \( h, t \in \mathcal{E}^* \) which is forcing the subsumption. \( \blacksquare \)
Appendix B

Proofs and Implementation
Details for Chapter 4

B.1 Proofs for HypE and HSimplE

**Theorem 4.1 (Expressivity)** For any ground truth over entities $E$ and relations $R$ containing $|\tau|$ true tuples and $\alpha = \max_{r \in R}(|r|)$, there exists a HypE and a HSimplE model with embedding vectors of size $\max(\alpha|\tau|, \alpha)$ that represents that ground truth.

The following proofs the theorem for HypE.

**Proof** To prove the theorem, we show an assignment of embedding values for each entity and relation in $\tau$ such that the scoring function of HypE is as follows:

$$\phi_{\theta}(x) = \begin{cases} 1 & \text{if } x \in \tau \\ 0 & \text{otherwise} \end{cases}$$

We begin the proof by first describing the embeddings of each of the entities and relations in HypE; we then proceed to show that with such an embedding, HypE can represent any world accurately.

Let us first assume that $|\tau| > 0$ and let $f_p$ be the $p$th fact in $\tau$. We let each entity $e \in E$ be represented with a vector of length $\alpha|\tau|$ in which the $p$th block of $\alpha$-bits is the one-hot representation of $e$ in fact $f_p$: if $e$ appears in fact $f_p$ at position $i$, then the $i$th bit of the $p$th block is set to 1, and to 0 otherwise. Each relation $r \in R$ is then represented as a vector of length $\tau$ whose $p$th bit is equal to 1 if fact $f_p$ is defined on relation $r$, and 0 otherwise.

![Figure B.1: An example of an embedding where $|\tau| = 5$, $\alpha = 4$ and $f_3$ is the third fact in $\tau$](image-url)
B.1. Proofs for HypE and HSimplE

HypE defines convolutional weight filters for each entity position within a tuple. As we have at most $\alpha$ possible positions, we define each convolutional filter $\omega_i$ as a vector of length $\alpha$ where the $i$th bit is set to 1 and all others to 0, for each $i = 1, 2, \ldots, \alpha$. When the scoring function $\phi_{\theta}$ is applied to some tuple $x$, for each entity position $i$ in $x$, convolution filter $\omega_i$ is applied to the entity at position $i$ in the tuple as a first step; the $\odot()$ function is then applied to the resulting vector and the relation embedding to obtain a score.

Given any tuple $x$, we want to show that $\phi_{\theta}(x) = 1$ if $x \in \tau$ and 0 otherwise.

First assume that $x = f_p$ is the $p$th fact in $\tau$ that is defined on relation $r$ and entities where $e_i$ is the entity at position $i$. Convolving each $e_i$ with $\omega_i$ results in a vector of length $|\tau|$ where the $p$th bit is equal to 1 (since both $\omega_i$ and the $p$th block of $e_i$ have a 1 at the $i$th position) (See Figure B.1. Then, as a first step, function $\odot()$ computes the element-wise multiplication between the embedding of relation $r$ (that has 1 at position $p$) and all of the convolved entity vectors (each having 1 at position $p$); this results in a vector of length $|\tau|$ where the $p$th bit is set to 1 and all other bits set to 0. Finally, $\odot()$ sums the outcome of the resulting products to give us a score of 1.

To show that $\phi_{\theta}(x) = 0$ when $x \notin \tau$, we prove the contrapositive, namely that if $\phi_{\theta}(x) = 1$, then $x$ must be a fact in $\tau$. We proceed by contradiction. Assume that there exists a tuple $x \notin \tau$ such that $\phi_{\theta}(x) = 1$. This means that at the time of computing the element-wise product in the $\odot()$ function, there was a position $j$ at which all input vectors to $\odot()$ had a value of 1. This can happen only when (1) applying the convolution filter $w_j$ to each of the entities in $x$ produces a vector having 1 at position $j$, and (2) the embedding of relation $r \in x$ has 1 at position $j$.

The first case can happen only if all entities of $x$ appear in the $j$th fact.
B.1. Proofs for HypE and HSimplE

Let \( f_j \in \tau \); the second case happens only if relation \( r \in x \) appears in \( f_j \). But if all entities of \( x \), as well as its relation, appear in fact \( f_j \), then \( x \in \tau \), contradicting our assumption. Therefore, if \( \phi_\theta(x) = 1 \), then \( x \) must be a fact in \( \tau \).

To complete the proof, we consider the case when \( |\tau| = 0 \). In this case, since there are no facts, all entities and relations are represented by zero-vectors of length \( \alpha \). Then, for any tuple \( x \), \( \phi_\theta(x) = 0 \). This completes the proof.

The following proofs the theorem for HSimplE.

**Proof** To prove the theorem, we follow the same strategy as Theorem 1 by showing an assignment of embedding values for each entity and relation in \( \tau \) such that the scoring function of HSimplE is as follows:

\[
\phi_\theta(x) = \begin{cases} 
1 & \text{if } x \in \tau \\
0 & \text{otherwise}
\end{cases}
\]

We begin the proof by first describing the embeddings of each of the entities and relations in HSimplE; we then proceed to show that with such an embedding, HSimplE can represent any world accurately.

Let us first assume that \( |\tau| > 0 \). We let each entity \( e \in \mathcal{E} \) be represented with a vector of length \( \alpha|\tau| \) in which the \( p \)th block of \( |\tau| \)-bits indicates entities that appeared in position \( p \) in the facts. In an entity embedding, the \( i \)th bit in the \( j \)th block is set to 1 if the entity appears in position \( j \) in the \( i \)th fact. Each relation \( r \in \mathcal{R} \) is then represented as a vector of length \( \alpha * |\tau| \) whose \( p \)th bit is equal to 1 if fact \( f_p \) is defined on relation \( r \), and 0 otherwise.

Given any tuple \( x \), we want to show that \( \phi_\theta(x) = 1 \) if \( x \in \tau \) and 0 otherwise.

First assume that \( x = f_i \) is the \( i \)th fact in \( \tau \) that is defined on relation \( r \) and entities \( e_1, e_2, \ldots, e_p \), where \( e_p \) is the entity at position \( p \). Shifting each \( e_p \) results in a vector where the \( i \)th bit is equal to 1 (see Figure [B.2]). Then, as a first step, function \( \odot() \) computes the element-wise multiplication between the embedding of relation \( r \) (that has 1 at position \( i \)) and all of the shifted entity vectors (each having 1 at position \( i \)). Finally, \( \odot() \) sums the outcome of the resulting products to give us a score of 1.

To show that \( \phi_\theta(x) = 0 \) when \( x \notin \tau \), we prove the contrapositive, namely that if \( \phi_\theta(x) = 1 \), then \( x \) must be a fact in \( \tau \). We proceed by contradiction. Assume that there exists a tuple \( x \notin \tau \) such that \( \phi_\theta(x) = 1 \). This means that at the time of computing the element-wise product in the \( \odot() \) function,
there was a position \( i \) at which all input vectors to \( \odot() \) had a value of 1. This can happen only when (1) applying the shift operation to each of the entities in \( x \) produces a vector having 1 at position \( i \), and (2) the embedding of relation \( r \in x \) has 1 at position \( i \).

The first case can happen only if all entities of \( x \) appear in the \( i \)th fact \( f_i \in \tau \); the second case happens only if relation \( r \in x \) appears in \( f_i \). But if all entities and relations of \( x \) appear in fact \( f_i \), then \( x \in \tau \), contradicting our assumption. Therefore, if \( \phi(x) = 1 \), then \( x \) must be a fact in \( \tau \).

To complete the proof, we consider the case when \( |\tau| = 0 \). In this case, since there are no facts, all entities and relations are represented by zero-vectors of length \( \alpha \). Then, for any tuple \( x \), \( \phi(x) = 0 \).

\[ \begin{align*}
B.2 & \quad \text{Theoretical Analysis} \\

\text{The current section groups the theoretical analysis of our work into four parts. In particular, Appendix B.2.1 proves that ReAlE is fully expressive. Appendix B.2.2 proves that m-TransH, RAE, and NaLP are not fully expressive (G-MPNN scoring function and its non-expressiveness were discussed in Section 4.3.1). Appendix B.2.3 shows how closely ReAlE can represent relational algebra operations. Appendix B.2.4 further proves that HypE cannot represent all relational algebra operations (in particular, we show that HypE cannot represent selection). For completeness, we restate the theorems.} \\

B.2.1 & \quad \text{Full Expressivity of ReAlE} \\

\text{The following result proves that there exists a setting of the parameters for which ReAlE can separate true and false tuples for arbitrary input. In particular, we show it for the case where \( \sigma \) is the sigmoid function.} \\

\text{Theorem 4.2 (Full Expressivity) For any ground truth over entities \( \mathcal{E} \) and relations \( \mathcal{R} \) containing \( \lambda \) true tuples with \( \alpha = \max_{r \in \mathcal{R}}(|r|) \) as the maximum arity over all relations in \( \mathcal{R} \), there is a ReAlE model with \( n_w = \lambda \), \( w = \alpha \), \( d = \max(\alpha \lambda, \alpha) \), and \( \sigma(x) = \frac{1}{1+\exp(-x)} \) that accurately separates the true tuples from the false ones.} \\

\text{Proof} \quad \text{Let} \quad \tau = \{\tau_0, \tau_1, \ldots, \tau_{\lambda-1}\} \text{ be all the true tuples defined over} \quad \mathcal{E} \text{ and} \quad \mathcal{R}. \quad \text{To prove the theorem, we show an assignment of embedding values for each of the entities and relations in} \quad \tau \text{ such that the scoring function of ReAlE is} \\
\end{align*} \]
B.2. Theoretical Analysis

Figure B.3: An example of a ReAlE embedding assignments for true tuples \( \tau_0 = r_1(x_a, x_b, x_c) \), \( \tau_1 = r_2(x_a, x_c) \), \( \tau_2 = r_1(x_c, x_b, x_d) \) and \( \tau_3 = r_1(x_d, x_c, x_a) \).

Here, the number of true tuples \( \lambda = n_w = 4 \), maximum arity \( \omega = 3 \). Cells that are set to zero are left empty for better readability. As an example, given that \( x_b \) is in position 1 of \( \tau_2 \), we set cell \( 2 \times \omega + 1 = 7 \) of \( x_b \) to 1. We further set the value of \( r_1 \) at positions \([1][7] \) to \( K \).

as follows.

\[
\phi_\theta(\tau) \begin{cases} 
\geq 1 - \epsilon & \text{if } \tau \in T \\
< \lambda \epsilon & \text{otherwise}
\end{cases}
\]

Here, \( \epsilon \) is an arbitrary small value such that \( \epsilon < \frac{1}{2+\lambda^2} \). Observe that \( \lambda \) (the number of true tuples) is positive. So, for any value of \( \lambda \), \( \lambda \epsilon \) and \( 1 - \epsilon \) never meet (As \( \epsilon < \frac{1}{2+\lambda^2} \) and also \( \epsilon < \frac{1}{1+\lambda} \), therefore \( \lambda \epsilon < 1 - \epsilon \)).

We first consider the case where \( \lambda > 0 \). Let \( K = 2 \times \sigma^{-1}(1 - \epsilon) \). Then, \( \sigma(K) = 1 - \epsilon \) and \( \sigma(-K) = \epsilon \).

We begin the proof by first describing an assignment of the embeddings of each of the entities and relations in ReAlE; we then proceed to show that with such an embedding, ReAlE accurately separates the true tuples from the false ones.

We consider the embeddings of entities to be \( n_w = \lambda \) blocks of size \( w \) each, such that each block \( i \) is conceptually associated with true tuple \( \tau_i \) for all \( 0 \leq i < \lambda \). Then, for a given entity \( x_m \) at position \( m \) of tuple \( \tau_i \), we set the value \( x_m[w + m] \) to 1, for all \( 0 \leq i < \lambda \). All other values in \( x_m \) are set to zero. For the relation embeddings, first, recall that these embeddings are matrices of dimension \( |r| \times w \lambda \), and we consider each of the \(|r|\) rows of this matrix to be \( \lambda \) blocks of size \( w \), where \(|r| > 1 \) is the arity of the given relation. If a given relation \( r \) appears in true tuple \( \tau_i \), we set the \( |r| \) values at position \([iw+k][iw+k] \) to \( K \), for all \( 0 \leq k < |r| \) and \( 0 \leq i < \lambda \); all other values in the relation embedding are set to zero. Finally, we set all the bias terms in our model to \( b^j_r = \frac{K}{2} - |r| \times K \), for all \( 0 \leq j < \lambda \). As an example,
consider Figure B.3, in which the first, third and fourth tuples \(\tau_0, \tau_1\) and \(\tau_3\) are defined on relation \(r_1\); thus the embedding of \(r_1\) has \(K\) in the diagonal of the first, third and fourth \(3 \times 3\) blocks. The tuple \(\tau_0\) has entity \(x_b\) at its second position; hence the embedding of \(x_b\) has a 1 in its second position. We claim that with such an assignment, the score of tuples that are true is \(\geq 1 - \epsilon\) and \(< \lambda \epsilon\) otherwise.

To see why this assignment works, first, observe that our scoring function \(\phi_{\theta}\) is a summation of \(\lambda\) sigmoids; each sigmoid is defined on an embedding block where we sum the bias term with the sum of pairwise product between the entity and relation embeddings of the given block.

Let \(\tau_p = r(x_1, \ldots, x_m)\) be a true tuple and observe the embeddings of its entities and relations. The blocks at position \(p\) of the embeddings of each of \(x_1, \ldots, x_m\) contain exactly one value 1 each (with the rest being zero); and the block at position \(p\) of the relation embedding for \(r\) contains \(K\)s in the diagonal and zeros elsewhere. With such an assignment, the block at position \(p\) will contribute the following sigmoid to the scoring function.

\[
\sigma (b_p^r + |r|K) = \sigma \left( \frac{K}{2} - (|r|K) + (|r|K) \right) = \sigma \left( \frac{K}{2} \right) = 1 - \epsilon \quad (B.1)
\]

All other blocks \(q \neq p\) in \(\tau_p\) will contribute to the scoring function in one of two ways, depending on whether or not \(r\) is a relation in \(\tau_q\).

If the relation \(r\) of \(\tau_p\) is also the relation in \(\tau_q\) (e.g., \(r_1\) is a relation in \(\tau_0, \tau_2\) and \(\tau_3\) in Figure B.3), then when we multiply the \(|r|\) \(w\)-sized blocks at position \(q\) of the relation embedding to their corresponding entity blocks at position \(q\) (the inner double-summation in the scoring function Equation 4.7), there will be at most \(c < |r|\) instances where \(K\) (in the relation embedding) is multiplied by 1 (in the entity embedding). This is because otherwise, the fact at position \(q\) is the same as the fact at position \(p\) in \(\tau\), indicating there is a duplicate in our set of facts. More precisely, for all blocks \(q \neq p\) in \(\tau_p\) such that \(r\) is the relation of \(\tau_q\), the block at position \(p\) will contribute the following sigmoid to the scoring function.

\[
\sigma (b_q^r + cK) = \sigma \left( \frac{K}{2} - |r|K + cK \right) = \sigma \left( \frac{K}{2} - (r - c)K \right) < \epsilon \quad (B.2)
\]

If \(r\) is not a relation in \(\tau_q\), then the block at position \(q\) of the relation embedding is all zeros. In this case,

\[
\sigma (b_q^r + 0) = \sigma \left( \frac{K}{2} - (|r|K) \right) < \epsilon \quad (B.3)
\]
B.2. Theoretical Analysis

In the end, the score of any true tuple $\phi_\theta(\tau_p)$ will be the sum of $\lambda$ sigmoids such that exactly one of these sigmoids (block at position $p$) has a value $1 - \epsilon$, while the other $\lambda - 1$ sigmoids are $< \epsilon$. The score of a true tuple $\tau_p$ can thus be bounded as follows.

$$\phi_\theta(\tau_p) \geq 1 - \epsilon$$ (B.4)

In the case of a false tuple, all the sigmoids will yield values $< \epsilon$ (as they will be of the forms in either of Equation [B.2] or Equation [B.3]). The score of a false tuple $\tau_f$ can thus be bounded as follows.

$$\phi_\theta(\tau_f) < \lambda \epsilon$$ (B.5)

To complete the proof, we consider the case when $\lambda = 0$. In this case, we let the entity and relation embeddings be blocks of arbitrary size and set all values to zero. We set the bias terms as before. Then, the score of any tuple will be $\phi_\theta(\tau_f) < \epsilon$ (Equation [B.5]), which is what we want. This completes the proof.

B.2.2 Full Expressivity of Some Baselines

The following results prove that some of the baselines are not fully expressive and have severe restrictions on the types of relations they can model. Liu et al. [97] discuss some of these limitations, and here we address them more formally.

Theorem 4.3 m-TransH, RAE, and NaLP are not fully expressive and have restrictions on the relations these approaches can represent.

The proof of the above theorem follows from Lemma 4.3.1 and 4.3.2 below.

Lemma 4.3.1 m-TransH and RAE are not fully expressive and have restrictions on what relations these approaches can represent.

Proof Kazemi and Poole (2018) prove that TransH [151] is not fully expressive and explore its restrictions. As m-TransH reduces to TransH for binary relations, it inherits all its restrictions and is not fully expressive. RAE also follows the same strategy as m-TransH in modeling the relations. Therefore, both m-TransH and RAE are not fully expressive.
B.2. Theoretical Analysis

**Lemma 4.3.2** NaLP is not fully expressive.

**Proof** NaLP first concatenates the embeddings of entities and the embedding of their corresponding roles in the tuples, then applies to them the following functions: 1D convolution, projection layer, minimum, and another projection layer. Looking carefully at the output of the model, the NaLP scoring function for a tuple $r(x_1, \ldots, x_n)$ is in the form of $|P_1 x_1 + \cdots + P_n x_n + r|_1$ with $P_i$ as learnable diagonal matrices with some shared parameters. In the binary setup ($n = 2$), the score function of NaLP is $|P_1 x_1 + P_2 x_2 + r|_1$. Kazemi and Poole (2018), however, proved that translational methods having a score function of $|P_1 x_1 - \alpha P_2 x_2 + r|_i$ are not fully expressive and have severe restrictions on what relations these approaches can represent. NaLP has the same score function with $\alpha = -1$ and $i = 1$ and therefore is not fully expressive.

B.2.3 Representing Relational Algebra with ReAlE

**Theorem 4.4** (Renaming) Given permutation function $\pi$, and relation $s$, there exists a parametrization for relation $t$ in ReAlE such that for entities $x_1, \ldots, x_n$, with arbitrary embeddings

$$\phi_\theta(t(x_1, \ldots, x_n)) = \phi_\theta(s(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}))$$

**Proof** To prove the above statement, we show that given entity embeddings $x_1, \ldots, x_n$ and an embedding for $s$, there exists a parametrization for $t$ that satisfies the above equality.

We claim that the following settings for $t$ are enough to show the theorem.

1. $T[\pi(i)][k] = S[i][k] \ \forall \ 1 \leq i \leq n, \ \forall \ 0 \leq k < d$, and
2. $b^t_j = b^s_j \ \forall \ 0 \leq j < n_w$

To see why, we simply expand the score function of $s$ and replace the values for $s$ by that of $t$ as described, to obtain the score for $t$. 

127
B.2. Theoretical Analysis

\[ \phi_\theta(s(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})) \]
\[ = \sum_{j=0}^{n_w-1} \sigma(b^t_j + \sum_{i=1}^{|s|} \sum_{k=0}^{w-1} x_{\pi(i)}[j \times w + k] \times S[i][j \times w + k]) \]
\[ = \sum_{j=0}^{n_w-1} \sigma(b^s_j + \sum_{i=1}^{|s|} \sum_{k=0}^{w-1} x_{\pi(i)}[j \times w + k] \times T[\pi(i)][j \times w + k]) \]
\[ = \sum_{j=0}^{n_w-1} \sigma(b^t_j + \sum_{i=1}^{|s|} \sum_{k=0}^{w-1} x_i[j \times w + k] \times S[i][j \times w + k]) = \phi_\theta(t(x_1, x_2, \ldots, x_n)) \]

Theorem 4.5 (Projection) For any relation s on n arguments there exists a parametrization for relation t on m < n arguments in ReAlE such that for any arbitrary sequence x_1, x_2, \ldots, x_n

\[ \phi_\theta(t(x_1, x_2, \ldots, x_m)) \geq \phi_\theta(s(x_1, \ldots, x_n)) \]

Proof To prove the above statement, we first expand the score function of each side of the inequality.

\[ \phi_\theta(t(x_1, \ldots, x_m)) = \sum_{j=0}^{n_w-1} \sigma(b^t_j + \sum_{i=1}^m \sum_{k=0}^{w-1} x_i[j \times w + k] \times T[i][j \times w + k]) \]

(B.6)

\[ \phi_\theta(s(x_1, \ldots, x_n)) \]
\[ = \sum_{j=0}^{n_w-1} \sigma(b^s_j + \sum_{i=1}^{|s|} \sum_{k=0}^{w-1} x_i[j \times w + k] \times S[i][j \times w + k]) + \sum_{i=m+1}^{n} \sum_{k=0}^{w-1} x_i[j \times w + k] \times S[i][j \times w + k]) \]

(B.7)

The theorem holds with the following assignments.
B.2. Theoretical Analysis

1. $T[i][k] = S[i][k] \quad \forall \ 1 \leq i \leq m, \ \forall \ 0 \leq k < d$, and

2. $b'_j = b'_s + \sum_{i=m+1}^{n} \sum_{k=0}^{w-1} \max(0, S[i][k]) \quad \forall \ 0 \leq j < n_w$

Theorem 4.6 (Selection 1) For arbitrary relation $s$, there exists a parametrization for relation $t$ in ReAlE such that for arbitrary entities $x_1, \ldots, x_{n-1}$

$$
\phi_\theta(t(x_1, \ldots, x_{n-1})) = \phi_\theta(s(x_1, \ldots, x_{n-1}, x_{n-1}))
$$

Proof To prove the above statement, we first expand the score function of $\phi_\theta$ for the right side of the equality.

$$
\phi_\theta(s(x_1, \ldots, x_{n-1}, x_{n-1})) = \sum_{j=0}^{n_w-1} \sigma(b_s^j + \sum_{i=1}^{n-2} \sum_{k=0}^{w-1} x_i[j \times w + k] \times S[i][j \times w + k])
\quad + \sum_{k=0}^{w-1} x_{n-1}[j \times w + k] \times S[n-1][j \times w + k]
\quad + \sum_{k=0}^{w-1} x_{n-1}[j \times w + k] \times S[n][j \times w + k]) \quad \text{(B.8)}
$$

= (grouping terms)

$$
\sum_{j=0}^{n_w-1} \sigma(b_s^j + \sum_{i=1}^{n-2} \sum_{k=0}^{w-1} x_i[j \times w + k] \times S[i][j \times w + k])
\quad + \sum_{k=0}^{w-1} x_{n-1}[j \times w + k] \times (S[n-1][j \times w + k] + S[n][j \times w + k])
$$

For the theorem to hold, the above score has to be equal to that of $t$, which is described as follows.

$$
\phi_\theta(t(x_1, \ldots, x_{n-1}))
\quad = \sum_{j=0}^{n_w-1} \sigma(b_t^j + \sum_{i=1}^{n-2} \sum_{k=0}^{w-1} x_i[j \times w + k] \times T[i][j \times w + k])
\quad + \sum_{k=0}^{w-1} x_{n-1}[j \times w + k] \times T[n-1][j \times w + k]) \quad \text{(B.9)}
$$
B.2. Theoretical Analysis

The scores in Equation B.8 and Equation B.9 are equal when the embedding and bias of $t$ are set as follows.

1. $T[i][k] = S[i][k]$ \quad $\forall 1 \leq i \leq n - 2$, $\forall 0 \leq k < d$, and
2. $T[n - 1][k] = S[n - 1][k] + S[n][k]$ \quad $\forall 0 \leq k < d$, and
3. $b^t_i = b^t_k$ \quad $\forall 0 \leq j < n_w$

Theorem 4.7 (Selection 2) For arbitrary relation $s$ and for a fixed constant $c$, there exists a parametrization for relation $t$ in ReAlE such that for arbitrary entities $x_1, \ldots, x_{n-1}$

$$\phi_\theta(t(x_1, \ldots, x_{n-1})) = \phi_\theta(s(x_1, \ldots, x_{n-1}, c))$$

Proof Using similar score expansions as in the proof of Theorem 4.6, we can rewrite the scores of the two relations as follows.

$$\phi_\theta(s(x_1, \ldots, x_{n-1}, c))$$

$$= \sum_{j=0}^{n_w-1} \sigma(b^s_j + \sum_{i=1}^{n-2} \sum_{w-1}^{w-1} x_i[j \times w + k] \times S[i][j \times w + k]$$

$$+ \sum_{k=0}^{w-1} x_{n-1}[j \times w + k] \times S[n-1][j \times w + k]$$

$$+ \sum_{k=0}^{w-1} c[j \times w + k] \times S[n][j \times w + k])$$

(B.10)

$$\phi_\theta(t(x_1, \ldots, x_{n-1}))$$

$$= \sum_{j=0}^{n_w-1} \sigma(b^t_j + \sum_{i=1}^{n-2} \sum_{k=0}^{w-1} x_i[j \times w + k] \times T[i][j \times w + k]$$

$$+ \sum_{k=0}^{w-1} x_{n-1}[j \times w + k] \times T[n-1][j \times w + k])$$

(B.11)

The scores in Equation B.10 and Equation B.11 are equal when the embedding and bias of $t$ are as set follows.
B.2. Theoretical Analysis

1. $T[i][k] = S[i][k] \quad \forall \, 1 \leq i \leq n - 1, \forall \, 0 \leq k < d$, and
2. $b'_j = b'_s + \sum_{k=0}^{w-1} S[i][j \times w + k] \times c[j \times w + k] \quad \forall \, 0 \leq j < n_w$ 

---

**Theorem 4.8 (Set Union)** For arbitrary relations $s$ and $r$ with the same arity, there exists a parametrization for relation $t$ in ReAlE such that for arbitrary entity set $\bar{x}$

$$\phi_\theta(t(\bar{x})) \geq \max(\phi_\theta(s(\bar{x})), \phi_\theta(r(\bar{x})))$$

**Proof** Given that ReAlE embeds entities in non-negative vectors and examining the scoring functions of each of the relations $t$, $r$, $s$, it can be observed that the above inequality holds by setting the following values.

1. $T[i][k] = \max(S[i][k], R[i][k]) \quad \forall \, 1 \leq i < |\bar{x}|$ and $0 \leq k < d$.
2. $b'_j = \max(b'_s, b'_r) \quad \forall \, 0 \leq j < n_w$.

---

In the lemma that follows, recall that the relation complement function (if it exists) is some linear function $f(\phi_\theta(r(\bar{x}))) = \phi_\theta(-r(\bar{x}))$ for arbitrary relation $r$ and entities $\bar{x}$ that also has the form $f(\sigma(x)) = \sigma(c \times x)$ with $c$ as a constant. For instance, when $\sigma$ is sigmoid, $f(x) = 1 - x$ and $c = -1$ ($f(\sigma(x)) = 1 - \sigma(x) = \sigma(-x)$) and when $\sigma$ is tanh, $f(x) = -x$ and $c = -1$ ($f(\sigma(x)) = -\sigma(x) = \sigma(-x)$).

**Theorem 4.9 (Set Difference)** For arbitrary relations $r$ and $s$ with the same arity, if $f$ is a linear relation complement function and $f(\sigma(x)) = \sigma(c \times x)$ with $c$ as a constant, there exists a parametrization for relation $t$ in ReAlE such that for arbitrary entities $x_1, \ldots, x_n$

$$\phi_\theta(t(\bar{x})) \leq \min(\phi_\theta(s(\bar{x})), f(\phi_\theta(r(\bar{x}))))$$

**Proof**

As $f$ is the relation complement, we have:

$$\phi_\theta(-r(x_1, \ldots, x_n)) = f(\phi_\theta(r(x_1, \ldots, x_n)))$$

131
B.2. Theoretical Analysis

As \( f \) is linear, we can distribute it inside the summation as follows:

\[
f(\phi_\theta(r(x_1, \ldots, x_n))) = \sum_{j=0}^{n_w-1} f(\sigma(b^j_t + \sum_{i=1}^{|r|} \sum_{k=0}^{w-1} x_i[jw+k] \times R[i][jw+k])
\]

Now, as \( f(\sigma(x)) = \sigma(c \times x) \), we can distribute it inside the \( \sigma \) as follows:

\[
f(\phi_\theta(r(x_1, \ldots, x_n))) = \sum_{j=0}^{n_w-1} \sigma(b^j_t \times c + \sum_{i=1}^{|r|} \sum_{k=0}^{w-1} x_i[jw+k] \times R[i][jw+k] \times c)
\]

Therefore, for the above inequality to hold, the bias terms and embedding values of \( t \) must be at most that of each of \( s \) and the complement of the score of \( r \). Examining the scoring functions of each of the relations \( t, r, s \), it can be observed that the lemma holds when the following is set.

1. \( T[i][k] = \min(S[i][k], R[i][k] \times c) \quad \forall \ 1 \leq i < |x| \) and \( 0 \leq k < d \).
2. \( b^j_t = \min(b^j_s, b^j_r \times c) \quad \forall \ 0 \leq j < n_w. \)

The following theorem establishes the ability of ReAlE to jointly capture the relational algebra operations discussed above. This is of interest, particularly as Abboud et al. [3] claimed "Capturing multiple inference patterns jointly is significantly more challenging [than capturing them singly]." Here the parametrizations do not interfere with each other; each relation has its own parametrization.

**Theorem 4.10 (Composition)** For an arbitrary set of relations \( S_r \) and arbitrary non-empty composition of operations \( S_{op} \) from the set renaming, projection, selection, set union, and set difference, there exists a parametrization for relation \( t \) in ReAlE with \( t \) as the resulting relation of applying \( S_{op} \) to \( S_r \).

**Proof** Given any operation in \( S_{op} \) composed of \( k \) primitive operations, we prove the theorem by induction on \( k \) for arbitrary \( k \).

**Base case:** For one operation \( (k = 1) \), according to Theorems 4.4 to 4.9, ReAlE is able to represent single operations for arbitrary relation(s) as input to the operation. So the base case holds.
B.2. Theoretical Analysis

Induction step (operation $k$ to $k+1$): Assume a parametrization exists for all compositions of length $k$. Now consider an operation composed of $k+1$ operations from $S_{op}$. If we ignore the last operation in the sequence, we know by the induction hypothesis that there exists a parametrization for a relation that represents the composition of the first $k$ operations. Adding the $(k+1)^{st}$ relation back to the sequence, the preceding theorems (Theorems 4.4 to 4.9) show that we can define a parametrization for this last operation based on that of the previous that is independent of the parameters of the first $k$ operations. This completes the proof.

Theorem 4.11 (Joint Representation) ReAL-E is able to jointly represent a set of relations each being the result of a relational algebra operations renaming, projection, selection, set difference, and set union or a composition of the operations.

Proof All the parametrizations proposed in the proofs of Theorems 4.4 to 4.10 are solely based on defining the parametrization of the output relation $t$ based on the input relation(s) without changing that of other relations (including $t$). Since each relation $t$ uses its own parametrization without affecting any parameters of other relations, then all such relations can be represented concurrently. This proves the theorem.

B.2.4 Representing Relational Algebra with HypE

So far, we showed that ReAL-E is fully expressive and can represent the relational algebra operations renaming, projection, selection, set union, and set difference. We also showed that most other models for knowledge hypergraph completion are not fully expressive. In this section, we show that even as HypE ([40]) is fully expressive in general, it cannot represent some relational algebra operations (namely, selection) while at the same time retaining its full expressivity. In special cases, such as when all tuples are false, it obviously can, however, we show that in the general case it cannot.

We proceed by first showing in Theorem [B.1] that HypE cannot represent selection for arbitrary entity and relation embeddings while retaining full expressivity. Now, one might think that even if representing selection is not possible for all embeddings, there might be some embedding space for which HypE would be able to represent selection. In Theorem [B.2] we show that when the embedding size is less than the number of entities (as it usually is
B.2. Theoretical Analysis

the case), then there is no setting for which HypE would be able to represent selection.

HypE embeds an entity \( x_i \) and a relation \( r \) in vectors \( x_i \in \mathbb{R}^d \) and \( r \in \mathbb{R}^d \) respectively, where \( d \) be the embedding size. In what follows, we let \( f(x, p) \) be a function that computes the convolution of the embedding of entity \( x \) with the corresponding convolution filters associated with position \( p \) and outputs a vector (see [40] for more details). We let \( \phi_H^t \) be the HypE scoring function.

**Theorem B.1 (HypE Selection 1 (a))** For arbitrary relation \( s \) and arbitrary embeddings for entities \( x_1, \ldots, x_{n-1} \), there exists no parametrization for relation \( t \) in HypE such that

\[
\phi_H^t (t(x_1, \ldots, x_{n-1})) = \phi_H^s (s(x_1, \ldots, x_{n-1}, x_{n-1}))
\]

**Proof**

To see why there is no parametrization for \( t \) that satisfies Equation B.12, we first expand the left and right sides of the equation as follows.

\[
\phi_H^t (t(x_1, \ldots, x_{n-1})) = \phi_H^s (s(x_1, \ldots, x_{n-1}, x_{n-1}))
\]

\[
= \sum_{i=1}^{d} t[i] \times f(x_1, 1)[i] \times \cdots \times f(x_{n-1}, n-1)[i] = \\
\sum_{i=1}^{d} s[i] \times f(x_1, 1)[i] \times \cdots \times f(x_{n-1}, n-1)[i] \times f(x_{n-1}, n)[i]
\]

For this equation to hold, the following should hold for arbitrary entities \( x_1, \ldots, x_{n-1} \), and \( \forall \ 1 \leq i \leq d \).

or \[
\left\{ \begin{array}{l}
  f(x_1, 1)[i] \times \cdots \times f(x_{n-1}, n-1)[i] = 0 \\
  t[i] = s[i] \times f(x_{n-1}, n)[i]
\end{array} \right. \] (B.13)

For an entity \( x \) at position \( p \) in a tuple, the output of function \( f(x, p) \) depends on the embedding of entity \( x \) and the convolution filters associated with position \( p \). Note that these convolution filters are shared among all relations in the knowledge hypergraph and are not specific to relation \( s \) or \( t \).

As we want Equation B.13 to hold for arbitrary entity embeddings, it is easy to see that there exists at least one setup for convolution filters and entity embeddings for which none of the factors in the product \( f(x_1, 1)[i] \times \cdots \times f(x_{n-1}, n-1)[i] \) is zero.
B.2. Theoretical Analysis

Now consider such an embedding setting. The only way Equation B.13 is satisfied is when we set \( t[i] = s[i] \times f(x_{n-1}, n)[i] \) for all \( 1 \leq i \leq d \). Observe that by setting the embedding of relation \( t \) to the embedding of \( s \times f(x_{n-1}, n) \) for a particular entity \( x_{n-1} \), we have effectively set it to a fixed value. Now consider an entity \( x_k \) such that \( f(x_k, n) \neq f(x_{n-1}, n) \), and apply the selection \( t \) to \( x_k \) by replacing \( x_{n-1} \) with \( x_k \) in Equation B.12. The equality condition in the equation will not hold, because none of the conditions in Equation B.13 hold. Therefore, in this setup, \( t \) fails to represent \( s \) for arbitrary entity embeddings.

Given that relation \( t \) should represent selection of \( s \) for all the entities in the hypergraph, setting it to a fixed value will make it incapable of representing selection for arbitrary entities. We can thus conclude that there is no parametrization for HypE such that it can represent selection for arbitrary embeddings of relations and entities.

We now show that when the embedding size is less than the number of entities \(|E|\) in the hypergraph, there is no setting for which HypE would be able to represent selection without losing full expressivity.

**Theorem B.2 (HypE Selection 1 (b))** For arbitrary relation \( s \), there exists no parametrization for relation \( t \) in HypE having scoring function \( \phi^H_\theta \) such that for arbitrary entities \( x_1, \ldots, x_n \) and embedding dimension \( d < |E| \), where \(|E|\) is the number of entities in the knowledge hypergraph, HypE remains fully expressive and

\[
\phi^H_\theta(t(x_1, \ldots, x_{n-1})) = \phi^H_\theta(s(x_1, \ldots, x_{n-1}, x_{n-1}))
\]

**Proof**

Consider \(|s| = 2\) and \(|t| = 1\) and tuples \( t(x_j) \) and \( s(x_j, x_j) \) such that relation \( t \) is a selection of \( s \). We will show that when the embedding dimension \( d \) of the entities is smaller than the number of entities in the knowledge hypergraph, there is no parametrization of HypE that gets \( t \) to represent \( s \) while retaining full expressivity.

For the sake of contradiction, assume that the lemma statement is false. Then there exists a parametrization for \( t \) such that HypE is fully expressive and that

\[
\phi^H_\theta(t(x_j)) = \phi^H_\theta(s(x_j, x_j))
\]
Expanding the score function for $s$ and $t$ we get

$$
\sum_{i=1}^{d} t[i] \times f(x_j, 1)[i] = \sum_{i=1}^{d} s[i] \times f(x_j, 1)[i] \times f(x_j, 2)[i] 
$$

$$
\Rightarrow \sum_{i=1}^{d} f(x_j, 1)[i] \times (t[i] - s[i] \times f(x_j, 2)[i]) = 0 
$$

For this equation to hold for arbitrary entities $x_1, x_2, \ldots, x_n$, it should hold for all $1 \leq i \leq d$ and all $1 \leq j \leq |E|$ for which $s(x_j, x_j)$ is true. Assuming that $d < |E|$, we need to have the following hold for all $1 \leq j \leq |E|$ and $1 \leq i \leq d$.

$$
\text{or} \begin{cases} 
  f(x_j, 1)[i] = 0 \\
  t[i] = s[i] \times f(x_j, 2)[i] 
\end{cases} 
$$

We claim that to satisfy the above equation simultaneously for all possible $i$ and $j$, there must be at least one entity $x_k$ for which the convolution function returns a zero vector; that is, $f(x_k, 1)[i] = 0$ for all $0 \leq i \leq d$. To see why this is true, assume the contrary; that is, for each convolution filter $f(x_j, 1)$ has at least one bit that is different than zero for arbitrary entity $x_j$. Without loss of generality, let the $j$th bit $f(x_j, 1)[j] = K_j$, for all $1 \leq j \leq |E|$ and $K_j \in \mathbb{R}^*$. Then, to satisfy Equation B.14 at index $j$, we must set $t[j] = s[j] \times f(x_j, 2)[j]$. As Equation B.14 must be satisfied for all entities, then all other entities $x_k$ with $k \neq j$ must have their $j$-th bit set to zero. Thus $f(x_k, 1)[j] = 0$ for all $0 \leq j \leq d$ and $j \neq k$. See Figure B.4. Since we have $d < |E|$, and by the pigeon-hole principle, there must be at least one entity $x_{d+1}$ such that $f(x_{d+1}, 1) = 0$ for all $1 \leq i \leq d$. This contradicts the original assumption, and thus there exists at least one $k$ for which $f(x_k, 1)$ returns a zero vector.

This would further imply that any tuple having $x_k$ in the first position will have a score $\phi_{\theta}^H$ of zero. This would be regardless of the relation in the tuple, or whether or not it is true. This violates the full-expressivity of HypE, thus contradicting the original assumption that the lemma statement is False. Therefore, when $d < |E|$, there is no parametrization for $t$ such that HypE represents selection while retaining full expressivity. 

\[\blacksquare\]
B.3. Implementation Details

We implement ReAlE in PyTorch [109] and use Adagrad [35] as the optimizer and dropout [133] to regularize the model. We perform early stopping and hyperparameter tuning based on the MRR on the validation set. We fix the maximum number of epochs to 1000 and embedding size to 200. We tune $lr$ (learning rate) and $w$ (window size) using the sets $\{0.05, 0.08, 0.1, 0.2\}$, and $\{1, 2, 4, 5, 8\}$ (first five divisors of 200). We tune $\sigma$ (nonlinear function) using the set $\{\text{tanh}, \text{sigmoid}, \text{exponent}\}$ for the jF17K dataset. As $\text{sigmoid}$ outperforms $\text{tanh}$ and $\text{exponent}$ on jF17K, we only tried $\text{sigmoid}$ for other datasets. For the experiment on REL-ER and also in Section 4.3.7 (ReAlE with a fixed negative ratio and batch size), we fixed the negative ratio and batch size of all baselines and our model to 10 and 128 respectively.

Reported results for the baselines on jF17K, FB-AUTO, and M-FB15K are taken from the original paper except for that of GETD [97] and BoxE [3] on M-FB15K. The original paper of GETD only reports results for arity 3 and 4 as trained and tested separately in the corresponding arity. However, in our experimental setup, we train and test in a dataset containing relations of different arities. For that, we train and test GETD. As GETD learns a tensor of dimension $|r|$ for each relation $r$, it needs $d^{r}$ (with $d$ as embedding size) number of parameters. The original paper proposes smart strategies to reduce the number of parameters to be learned by the model. However, we still need to store the relation embedding and thus need to store $d^{r}$ floating-point numbers for each relation $r$. Because of our memory limitation (12GB GPU), we could only train the GETD model for embedding size of less than
B.3. Implementation Details

10. The sign "-" in Table 1 indicates that the corresponding paper has not provided the results.

For the experiment on the synthetic dataset, we compare our model with m-TransH\(^4\) [153], HypE\(^5\) [40], and BoxE\(^6\) [3], which are the only competitive baselines that have provided the code.

For all the experiments we use a single 12GB GPU (NVIDIA Tesla P100 PCIe 12 GB).

\(^4\)https://github.com/wenjf/multi-relational_learning
\(^5\)https://github.com/ElementAI/HypE
\(^6\)https://github.com/ralphabb/BoxE
Appendix C

Supplementary Material for Chapter 5

C.1 Supervision Starvation in Erdős-Rényi and Scale-free networks

We start by defining some new notation that helps simplify the proofs and analysis in this section. We let \( l_v \) be a random variable indicating that \( v \) is a labeled node, with \( \overline{l}_v \) indicating its negation, \( c_{v,u} \) be a random variable indicating that \( v \) is connected to \( u \) with an edge, with \( \overline{c}_{v,u} \) indicating its negation, and \( c_{l_v} \) be random variable indicating that \( v \) is connected to at least one labeled node with \( \overline{c}_{l_v} \) indicating its negation (i.e., it indicates that \( v \) is connected to no labeled nodes).

**Theorem 5.1** Let \( G(n, m) \) be an Erdős-Rényi graph with \( n \) nodes and \( m \) edges. Assume we have labels for \( q \) nodes selected uniformly at random. The probability of an edge being a starved edge with a two-layer GCN is equal to the following:

\[
(1 - \frac{q}{n})(1 - \frac{q}{n-1}) \prod_{i=1}^{2q} (1 - \frac{m - 1}{\binom{n}{2} - i})
\]

**Proof** To compute the probability of an edge being a starved edge, we first compute the probability of the two nodes of the edge being unlabeled themselves and then the probability of the two nodes not being connected to any labeled nodes. Let \( v \) and \( u \) represent two nodes connected by an edge.

With \( n \) nodes and \( q \) labels, the probability of a node being labeled is \( \frac{q}{n} \). Therefore, \( Pr(\overline{l}_v) = (1 - \frac{q}{n}) \) and \( Pr(\overline{l}_u | \overline{l}_v) = (1 - \frac{q}{n-1}) \). Therefore, \( Pr(\overline{l}_v \land \overline{l}_u) = (1 - \frac{q}{n})(1 - \frac{q}{n-1}). \)

Since there is an edge between \( v \) and \( u \), there are \( m-1 \) edges remaining. Also, there are \( \binom{n}{2} - 1 \) pairs of nodes that can potentially have an edge between them. Therefore, the probability of \( v \) being disconnected from the first labeled node is \( 1 - \frac{m-1}{\binom{n}{2}-1} \). If \( v \) is disconnected from the first labeled...
C.1. Supervision Starvation in Erdős-Rényi and Scale-free networks

node, there are still $m - 1$ edges remaining and there are now $\binom{n}{2} - 2$ pairs of nodes that can potentially have an edge between them. So the probability of $v$ being disconnected from the second node given that it is disconnected from the first labeled node is $1 - \frac{m-1}{\binom{n}{2}-2}$. With similar reasoning, we can see that the probability of $v$ being disconnected from the $i$-th labeled node given that it is disconnected from the first $i - 1$ labeled nodes is $1 - \frac{m-1}{\binom{n}{2}-i}$.

We can follow similar reasoning for $u$. The probability of $u$ being disconnected from the first labeled node given that $v$ is disconnected from all $q$ labeled nodes is $1 - \frac{m-1}{\binom{n}{2}-q-1}$. That is because there are still $m - 1$ edges remaining and $\binom{n}{2} - q - 1$ pairs of nodes that can potentially be connected with an edge. We can also see that the probability of $u$ being disconnected from the $i$-th labeled node given that it is disconnected from the first $i - 1$ labeled nodes and that $v$ is disconnected from all $q$ labeled nodes is $1 - \frac{m-1}{\binom{n}{2}-q-i}$.

As the probability of the two nodes being unlabeled and not being connected to any labeled nodes in the graph are independent, their joint probability is the multiplication of their probabilities computed above and it is equal to $(1 - \frac{q}{n})(1 - \frac{q}{n-1}) \prod_{i=1}^{q}(1 - \frac{m-1}{\binom{n}{2}-i})$.

Barabási–Albert and scale-free networks. We also extend the above result for Erdős-Rényi graphs to the Barabási–Albert [9] model. Since Barabási–Albert graph generation results in scale-free networks with a scale parameter $\gamma = -3$, we present results for the general case of scale-free networks as it makes the analysis simpler and more general. In what follows, we compute the probability of an edge being a starved edge in a scale-free network.

Let $G$ be a scale-free network with $n$ nodes, $q$ labels (selected uniformly at random), and scale parameter $\gamma$. Then, if we select a random edge between two nodes $v$ and $u$, the probability of the edge between them being a starved edge is:

$$Pr(l_v) \cdot Pr(l_u | l_v) \cdot Pr(cl_v | c_{v,u}, l_v, l_u) \cdot Pr(cl_u | c_{v,u}, l_v, l_u, cl_v).$$

Each of these terms can be computed as follows ($\binom{a}{b}$ represents the number of combinations of selecting $b$ items from a set with $a$ items):

- $Pr(l_v) = 1 - \frac{q}{n}$
C.2. Implementation Details

- $Pr(l_u | l_v) = (1 - \frac{n}{n-1})$
- $Pr(cl_v | c_v, u, l_u, l_v) = \frac{\sum_{k=1}^{n-1} k^\gamma \binom{n-q-2}{k-1} \binom{n-2}{k-1}}{\sum_{k=1}^{n-1} k^\gamma}$

For a large enough network, $Pr(cl_v | c_v, u, l_v, l_u, cl_u)$ can be approximated as $Pr(cl_v | c_v, u, l_v, l_u)$ and it can be computed similarly as the previous case.

With the derivation above, for a scale-free network with $n = 2708$ and $q = 140$ (corresponding to the stats from CORA), the probability of an edge being a starved edge for $\gamma = -3$ is 0.87 and for $\gamma = -2$ is 0.76.

C.2 Implementation Details

We implemented our model in PyTorch [109]. We used deep graph library (DGL) [145] for the sparse operations, and used Adam [81] as the optimizer. We performed early stopping and hyperparameter tuning based on the accuracy on the validation set for all datasets except wine and cancer. For these two datasets, the validation accuracy reached 100 percent with many hyperparameter settings, making it difficult to select the best set of hyperparameters. Instead, we used the validation cross-entropy loss for these two datasets.

We fixed the maximum number of epochs to 2000. We use two-layer GCNs for both GNN_C and GNN_DAE as well as for baselines and two-layer MLPs throughout the chapter (for experiments on OGBN-ARXIV, although the original paper uses models with three layers and with batch normalization after each layer, to be consistent with our other experiments we used two layers and removed the normalization). We used two learning rates, one for GNN_C as $l_C$ and one for the other parameters of the models as $l_{DAE}$. We tuned the two learning rates from the set \{0.01, 0.001\}. We added dropout layers with dropout probabilities of 0.5 after the first layer of the GNNs. We also added dropout to the adjacency matrix for both GNN_C and GNN_DAE as dropout_C and dropout_DAE respectively and tuned the values from the set \{0.25, 0.5\}. We set the hidden dimension of GNN_C to 32 for all datasets except for OGBN-ARXIV for which we set it to 256. We used cosine similarity for building the kNN graphs and tuned the value of $k$ from the set \{10, 15, 20, 30\}. We tuned $\lambda$ (\lambda controls the relative importance of the two losses) from the set \{0.1, 1, 10, 100, 500\}. We tuned $r$ and $\eta$ from the sets \{1, 5, 10\} and \{1, 5\} respectively. The best set of hyperparameters for each dataset chosen on the validation set is in Table C.1.
C.3. More Experiments and Analysis

For GRCN [163], DGCNN [150], and IDGL [23], we used the code released by the authors and tuned the hyperparameters as suggested in the original papers. The results of LDS [47] are directly taken from the original paper. For LP [175], we used scikit-learn python package [110].

All the results for our model and the baselines are averaged over 10 runs. We report the mean and standard deviation. We ran all the experiments on a single GPU (NVIDIA GeForce GTX 1080 Ti).

For self-training, we considered the label predictions for the top \( \zeta \) most confident unlabeled nodes as ground truth labels and added them to the training labels. Here, \( \zeta \) is a hyperparameter. We tuned its value from the set \{50, 100, 200, 300, 400, 500\}.

C.2.1 kNN Implementation

For our MLP generator, we used a kNN operation to sparsify the generated graph. Here, we explain how we implemented the kNN operation to avoid blocking the gradient flow. Let \( M \in \mathbb{R}^{n \times n} \) with \( M_{ij} = 1 \) if \( v_j \) is among the top \( k \) similar nodes to \( v_i \) and 0 otherwise, and let \( S \in \mathbb{R}^{n \times n} \) with \( S_{ij} = \text{Sim}(X'_i, X'_j) \) for some differentiable similarity function \( \text{Sim} \) (we used cosine). Then \( \tilde{A} = \text{kNN}(X') = M \circ S \) where \( \circ \) represents the Hadamard (element-wise) product. With this formulation, in the forward phase of the network, one can first compute the matrix \( M \) using an off-the-shelf k-nearest neighbors algorithm and then compute the similarities in \( S \) only for pairs of nodes where \( M_{ij} = 1 \). In our experiments, we compute exact k-nearest neighbors; one can approximate it using locality-sensitive hashing approaches for larger graphs (see, e.g., [56, 84]). In the backward phase of our model, we compute the gradients only with respect to those elements in \( S \) whose corresponding value in \( M \) is 1 (i.e., those elements \( S_{ij} \) such that \( M_{ij} = 1 \)); the gradient with respect to the other elements is 0. Since \( S \) is computed based on \( X' \), the gradients flow to the elements in \( X' \) (and consequently to the weights of the MLP) through \( S \).

C.3 More Experiments and Analysis

C.3.1 Symmetrization

In the adjacency processor, we used the following equation:

\[
A = D^{-\frac{1}{2}} \left( \mu(\tilde{A}) + \mu(\tilde{A})^T \right) D^{-\frac{1}{2}}
\]
C.3. More Experiments and Analysis

Table C.1: Best set of hyperparameters for different datasets chosen on the validation set.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Generator</th>
<th>lr</th>
<th>lrDAE</th>
<th>dropout</th>
<th>dropoutDAE</th>
<th>k</th>
<th>λ</th>
<th>r</th>
<th>η</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORA</td>
<td>FP</td>
<td>0.01</td>
<td>0.01</td>
<td>0.5</td>
<td>0.25</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CORA</td>
<td>MLP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CORA</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>15</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CITESEER</td>
<td>FP</td>
<td>0.01</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
<td>30</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>CITESEER</td>
<td>MLP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CITESEER</td>
<td>MLP-D</td>
<td>0.001</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CORA390</td>
<td>FP</td>
<td>0.01</td>
<td>0.01</td>
<td>0.25</td>
<td>0.5</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CORA390</td>
<td>MLP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CORA390</td>
<td>MLP-D</td>
<td>0.001</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CITESEER370</td>
<td>FP</td>
<td>0.01</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
<td>30</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>CITESEER370</td>
<td>MLP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>30</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>CITESEER370</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.01</td>
<td>0.25</td>
<td>0.5</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>PUBMED</td>
<td>MLP</td>
<td>0.01</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
<td>15</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>PUBMED</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.01</td>
<td>0.25</td>
<td>0.25</td>
<td>15</td>
<td>100</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>OGBN-ARXIV</td>
<td>MLP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>15</td>
<td>10</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>OGBN-ARXIV</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.001</td>
<td>0.5</td>
<td>0.25</td>
<td>15</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>WINE</td>
<td>FP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.5</td>
<td>0.5</td>
<td>20</td>
<td>0.1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>WINE</td>
<td>MLP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.5</td>
<td>0.25</td>
<td>20</td>
<td>0.1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>WINE</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.01</td>
<td>0.25</td>
<td>0.5</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>CANCER</td>
<td>FP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.5</td>
<td>0.25</td>
<td>20</td>
<td>0.1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>CANCER</td>
<td>MLP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.5</td>
<td>0.5</td>
<td>20</td>
<td>0.1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>CANCER</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.01</td>
<td>0.25</td>
<td>0.5</td>
<td>20</td>
<td>0.1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>DIGITS</td>
<td>FP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>20</td>
<td>0.1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>DIGITS</td>
<td>MLP</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>20</td>
<td>0.1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>DIGITS</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.001</td>
<td>0.25</td>
<td>0.25</td>
<td>20</td>
<td>0.1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>20NEWS</td>
<td>FP</td>
<td>0.01</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
<td>20</td>
<td>500</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>20NEWS</td>
<td>MLP</td>
<td>0.001</td>
<td>0.001</td>
<td>0.25</td>
<td>0.5</td>
<td>20</td>
<td>500</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>20NEWS</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.01</td>
<td>0.25</td>
<td>0.25</td>
<td>20</td>
<td>100</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>MNIST (1000)</td>
<td>MLP</td>
<td>0.01</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
<td>15</td>
<td>10</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>MNIST (2000)</td>
<td>MLP-D</td>
<td>0.01</td>
<td>0.001</td>
<td>0.5</td>
<td>0.5</td>
<td>15</td>
<td>100</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>MNIST (3000)</td>
<td>MLP</td>
<td>0.01</td>
<td>0.01</td>
<td>0.5</td>
<td>0.5</td>
<td>15</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
C.3. More Experiments and Analysis

Figure C.1: The performance of SLAPS on cora and citeseer with different adjacency symmetrizations.

which symmetrized the adjacency matrix by taking the average of $\mu(\tilde{A})$ and $\mu(\tilde{A})^T$. Here we also consider two other choices: 1) $\max(\mu(\tilde{A}), \mu(\tilde{A})^T)$, and 2) not symmetrizing the adjacency (i.e., using $\mu(A)$). Figure C.1 compares these three choices on cora and citeseer with an MLP generator (other generators produced similar results). On both datasets, symmetrizing the adjacency provides a performance boost. Compared to mean symmetrization, max symmetrization performs slightly worse. This may be because max symmetrization does not distinguish between the case where both $v_i$ and $v_j$ are among the $k$ most similar nodes of each other and the case where only one of them is among the $k$ most similar nodes of the other.

C.3.2 Fixing a prior graph manually instead of using self-supervision

So far, we validated Hypothesis 1 by adding a self-supervised task to encourage learning a graph structure that is appropriate for predicting the node features, and showing in our experiments how this additional task helps improve the results. Here, we provide more evidence for the validity of Hypothesis 1 by showing that we can obtain good results even when regularizing the learned graph structure toward a manually fixed structure that is appropriate for predicting the node features.

Toward this goal, we experimented with cora and citeseer and created a cosine similarity graph as our prior graph $A^{\text{prior}}$ where the edge weights correspond to the cosine similarity of the nodes. We sparsified $A^{\text{prior}}$ by connecting each node only to the $k$ most similar nodes. Then, we added
C.3. More Experiments and Analysis

![Figure C.2: The performance of SLAPS and regularization toward a manually defined prior structure on CORA and CITIPRSE when using the MLP generator.](image)

A term $\lambda ||A - A^{prior}||_F$ to the loss function where $\lambda$ is a hyperparameter, $A$ is the learned graph structure (i.e., the output of the graph generator), and $||.||_F$ shows the Frobenius norm. Note that $A^{prior}$ exhibits homophily with respect to the node features because the node features in CORA and CITIPRSE are binary, so two nodes that share the same values for more features have a higher similarity and are more likely to be connected.

The results can be viewed in Figure [C.2]. According to the results, we can see that regularizing toward a manually designed $A^{prior}$ also provides good results but falls short of SLAPS with self-supervision. The superiority of the self-supervised approach compared to the manual design could be due to two reasons.

- Some of the node features may be redundant (e.g., they may be derived from other features) or highly correlated. These features can negatively affect the similarity computations for the prior graph in $A^{prior}$. As an example, consider three nodes with seven binary features $[0, 0, 0, 1, 1, 1, 1]$, $[0, 0, 0, 0, 0, 0, 0]$ and $[1, 1, 1, 1, 1, 1, 1]$ respectively and assume the last two features for each node are always equivalent and are computed based on a logical and of the 4th and 5th features\footnote{For the first node in the example, the 4th and 5th features are both 1 so their logical and is also 1 and so the last two features for this node are both 1. The computation for the other two nodes is similar.}. Without these two features, the first node is more similar to the second than the third node, but when considering these derived features,
C.3. More Experiments and Analysis

it becomes more similar to the third node. This change in node similarities affects the construction of $A^{\text{prior}}$ which can deteriorate the overall performance of the model. The version of SLAPS with the self-supervised task, on the other hand, is not affected by this problem as much because the model can learn to predict the derived node features based on other features and without heavily relying on the graph structure.

- While many graph structures may be appropriate for predicting the node features, in the manual approach we only regularize toward one particular such structure. Using the self-supervised task, however, SLAPS can learn any of those structures; ideally, it learns the one that is more suited for the downstream task due to the extra supervision coming from the downstream task.

C.3.3 Why not compare the learned graph structures with the original ones?

A comparison between the learned graph structures using SLAPS (or other baselines) and the original graph structure of the datasets we used may not be sensible. We explain this using an example. Before getting into the example, we remind the reader that the goal of structure learning for semi-supervised classification with graph neural networks is to learn a structure with a high degree of homophily. Following Zhu et al. [172], we define the edge homophily ratio as the fraction of edges in the graph that connect nodes that have the same class label.

Figure C.3 demonstrates an example where two graph structures for the same set of nodes have the same edge homophily ratio (0.8 for both) but have no edges in common. For our task, it is possible that the original graph structure (e.g., the citation graph in CORA) corresponds to the structure on the left but SLAPS (or any other model) learns the graph on the right, or vice versa. While both these structures may be equally good, they do not share any edges. Therefore, measuring the quality of the learned graph using SLAPS by comparing it to the original graph of the datasets may not be sensible. However, if a noisy version of the initial structure is provided as input for SLAPS, then one may expect that SLAPS recovers a structure similar to the cleaned original graph and this is indeed what we demonstrate in Section 5.5.4 (Paragraph “Noisy graphs”)

---

*We are disregarding the features for simplicity sake.*
C.3. More Experiments and Analysis

Figure C.3: Two example graph structures. Node colors indicate the class labels. Solid lines indicate homophilous edges and dashed lines indicate non-homophilous edges. The two graphs exhibit the same degree of homophily yet there is no overlap between their edges.
Index

AdaEdge, 81
attributed graph, 7
directed graph, 6
embedding, 12
enforcing a subsumption, 22
fully expressive, 12
graph, xiv, 6, 7
graph neural networks, 14
head, 9
hypergraph, xiv, 7, 10
knowledge graph, 12
knowledge hypergraph, 11
knowledge hypergraph
completion, 32
latent graph learning, 71
link prediction, 11, 32
mean reciprocal rank, 17
node attributed graph, xiv, 7, 8
node classification, 11
non-trivial subsumption, 21
primitive operations, 50
reification, 33
relational algebra, 50
relational graph, xiv, 7, 9
relational hypergraph, xiv, 9, 10
scoring function, 12
self-training, 81
semi-supervised learning, 11
semi-supervised node
classification, 11
similarity graph, 71
star-to-clique, 34
starved edges, 78
tail, 9
triple, 9
trivial subsumption, 21
tuple, 9
undirected graph, 6
weighted graph, xiv, 6, 8