Representation Learning for Knowledge Graph Completion and Meta-Learning Recommender System

by

Yuxun Lu

Dissertation

submitted to the Department of Informatics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

March 2023

Acknowledgments

I would like to show my gratitude to my supervisors, Professor Ryutaro Ichise and Professor Hideaki Takeda, for all their support of the research in my last three years. The work introduced in this thesis is under their insightful comments and helps. I indeed appreciate the discussions and time shared by them, especially during the busiest period in their work. Also, I appreciate the time from the committee members, Professor Satoh, Professor Takasu, and Professor Sugiyama for their help in reviewing and discussing research in this thesis.

During the research, I received a lot of helpful questions and comments from my laboratory mates, especially from Doctor Rungsiman Nararatwong, Doctor Ziwei Xu, Wilebada Arachchige Tiroshan Madhushanka, and Rumana Ferdous Munne. Thanks for all the discussions and questions in the weekly lab meeting. These questions help me in many aspects of the research, especially in presenting and introducing my work.

As the quote claims, *unus pro omnibus, omnes pro uno.* I appreciate the support from my friends, Chang Xu (M. D.) Huacong Xu (J. D.) and Fengming Han (Ph. D.) for sharing their ideas on improving the skills of writing and organizing materials. I especially appreciate the discussions of mathematical details on machine learning from Dr. Han when I was in the early grade of my study.

Besides, I genuinely thank my parents, Huiwen Lu and Xiaomei Hu, and my uncle Huili Lu and aunt Dongmei Liu, and my sister Yukun Lu, for the emotional and financial support during the three years of my studying career. It would be extremely difficult for me to keep going without their selfless and unconditional support.

Last but not least, I'm grateful for having some friends that pursuit their Ph. D. courses. It makes me feel that I'm not alone in the last three years. The last sentence is for you all, *cognoscetis veritatem et veritas vos liberabit*!

Abstract

Representation learning has become a critical topic in many domains, such as natural language processing and computer vision. This thesis concerns representation learning on the knowledge graph. It contains two topics. The first topic focuses on learning implicit type representations to improve the type-agnostic knowledge graph completion models. The second topic turns to the application of knowledge graphs. It concerns learning representations for item features extracted from the knowledge graphs to help alleviate the user-side cold start problem in the recommender system. These two topics can constitute a pipeline: task 1 can add new facts to a knowledge graph according to existing ones, and these facts in the knowledge graph are used as features in task 2 to help improve the recommender system's performance.

Research in the first topic starts with a method that uses global entity co-occurrence to improve the link prediction of bilinear type-agnostic knowledge graph completion models. The entity co-occurrence is used with a loss function that is adapted from the language models to calibrate entity and relation representations in the bilinear knowledge graph completion models. In addition, it induces a relation-specific embedding to represent acceptable implicit entity types. Experiments show that this method has more improvement compared to a similar baseline approach designed for the bilinear knowledge graph completion models.

The second research in the first topic drops the requirement of entity co-occurrence and increases the number of relation-specific embeddings to capture implicit entity types. Compared to the first research and other similar unsupervised type representation learning methods, the method in the second research can capture more diverse implicit entity types and can be applied to both bilinear and translational knowledge graph completion models. Experiments show significant improvements for translational

knowledge graph completion models, even on the knowledge graph that it is challenging to define types on. The visualization of learned embeddings indicates that the proposed method can capture diverse implicit entity types by indirectly utilizing entities in relations.

In the second topic, the proposed method utilizes hypernetwork to generate parameters in the recommender network for the user-side cold-start problem. The contents of different attributes constitute item features in the second topic. Compared to the model-agnostic meta-learning approaches, the proposed method can adapt representations of both item attribute contents and the user interests in the item attributes themselves faster. In addition, the proposed method is robust towards different sizes of support and query sets in the meta-learning setting.

The research in the thesis focus on learning appropriate representations for the underlying task: the first topic is link prediction and entity clustering. The second topic is the recommendation for users with little data. The algorithm for the tasks is simple: the proposed methods in the first topic adapt existing type-agnostic knowledge graph completion models. The proposed method in the second topic uses a neural network that takes facts in the knowledge graph about items for the recommendation and another neural network for generating parameters for the recommender network. Both neural networks are with a simple structure. Experiment results in the thesis reveal that the (relatively) simple algorithms can have satisfying performance with appropriate learned representations.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Representation learning has become an important topic along with the rapid development of deep learning because the performance of machine learning techniques essentially depends on the underlying representations of data [3]. Representation learning often refers to distributed representation learning in the deep learning era. Input data is represented over many computing elements to extract active patterns, and each computing element is involved in different input data [4]. The computing elements are often represented by low-dimensional dense vectors; therefore, they are also referred as "embeddings." Distributed representation learning combined with deep learning has achieved great success in diverse domains, such as natural language processing [5, 6] and image recognition [7].

This thesis concerns representation learning on the knowledge graph. A knowledge graph contains real-world facts in the form of triples, namely, (head, relation, tail). For example, the fact "NII is located in Tokyo" can be expressed as (NII, is-located-in, Tokyo).

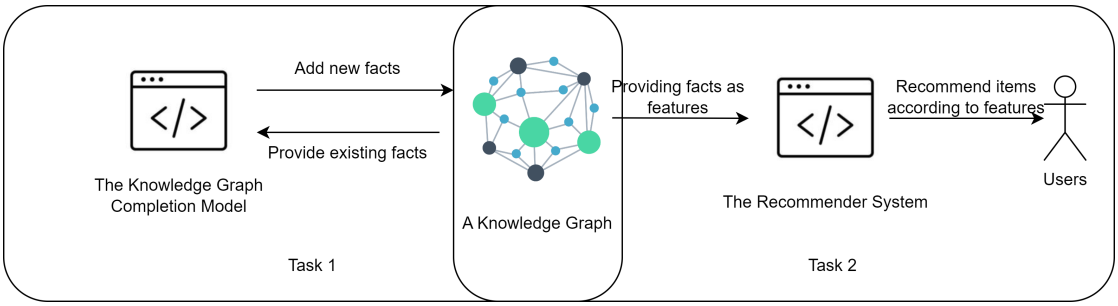


Figure 1.1: The connection of two tasks discussed in the thesis.

Facts in the knowledge graph can help downstream tasks, such as the Recommender System [8] and Question Answering [9]. As representation learning should be discussed with specific tasks [10, 3], this thesis considers two tasks: (1) adding new facts to a knowledge graph according to existing ones; and (2) taking facts in a knowledge graph to help the recommender system in a low-resource scenario. To be specific, these two tasks are (1) calibrating representations of these facts themselves to improve the link prediction performance; and (2) utilizing facts and fact structures to improve the meta-learning recommender system performance on the user-side cold-start problem. The first topic is about the knowledge graph itself, and the second topic is about integrating knowledge graphs into the downstream task, recommendation. Both topics focus on learning distributed representations of facts to improve the performance of the given task. In addition, the proposed methods for these two tasks can constitute a pipeline: adding new facts to a knowledge graph according to the existing ones to provide features for recommender systems. Figure 1.1 presents the connection between these two tasks.

As shown in Figure 1.1, the essential component in both tasks is the facts in the knowledge graph. All proposed methods in this thesis concern learning distributed representations of the facts in the knowledge graph to help the given tasks in the pipeline. Note that, in task 2, i.e., learning representations of facts to help recommender systems, the proposed method and the baseline models do not need other frameworks to utilize the facts in the knowledge graph (as introduced later) - the task can benefit from well-learned representations of facts without depending on other machine learning techniques (to show a counterexample, integrating knowledge graphs into question answering depending on language models and semantic analysis.) This

property is the reason why the recommender system is chosen for the downstream task.

The first topic concerns learning representations to extend the ability of knowledge graph completion (KGC) models. Knowledge graph completion models learn entity and relation embeddings from facts in the knowledge graph. KGC models use these embeddings to answer queries. For example, "Who is the president of the United States?" can be expressed as (?, president-of, United States). These embeddings are also used for adding new facts into the knowledge graph according to existing ones. Suppose there is a fact (Alice, born-in, United States) in the knowledge graph, we can add (Alice, citizen-of, United States) into the knowledge graph, because, according to the existing fact, this fact is very probable to be true. Most KGC models learn type-agnostic embeddings. They ignore entity types and relation type constraints (e.g., "Alice" probably has type "Human," "Animal," or "Creature," and the relation, "born-in," probably requires the last term to be an entity with the type "Location.") I proposed an unsupervised method that can enable KGC models to learn type-aware entity embeddings to improve their link prediction ability. In addition, the proposed method can also infer entity type according to entity embeddings. Details of this topic are introduced in Section 1.2.

The first topic is about learning representations to help enrich the contents (i.e., facts) in the knowledge graph itself, whereas the second topic is about utilizing the contents (facts and entity types) in the downstream application. Especially, this topic focus on improving the recommender system in a data-scarce scenario, i.e., meta-learning in the recommender system. This topic concerns the user-side cold-start problem, a problem caused by scarce user-item interactions and results in inferior recommendation results. The intrinsic reason of this problem is that the system cannot sufficiently optimize its parameters (i.e., the parameters in the recommender system, and the representations of items and users.) due to the shortage of user-item interactions. As a result, the sub-optimal parameters in the system cannot accurately reflect user interests. I proposed a method based on the hypernetwork [11] and reinforced it with the knowledge graph. Different from model-agnostic meta-learning methods, the proposed system requires fewer training data. The proposed system utilizes properties (i.e., facts) about the items from an external knowledge graph. It uses a hypernetwork to generate weights and biases to capture user interests in the

properties in the recommender system.

The first and second topics use contents from the knowledge graph, and the motivation behind these two topics are the same: learning representations (rather than drastically changing the algorithm) to perform better on corresponding tasks. The detailed backgrounds of the two topics are presented in the following sections.

1.2 Improving Knowledge Graph Completion Models with Type Representations

Knowledge graph stores facts about the real world in the form of triples, namely, (head, relation, tail). For example, "NII is located in Tokyo" can be expressed as (NII, is-located-in, Tokyo) in a knowledge graph. The head and the tail are entities in the knowledge graph. Let $\mathcal{E}$ and $\mathcal{R}$ denote the entity and relation sets, respectively. A knowledge graph can be expressed as a rank three tensor: $\mathcal{G} = \mathcal{E} \times \mathcal{R} \times \mathcal{E}$.

An essential task related to knowledge graphs is link prediction. Given a triple with one missing entity, namely (?, relation, tail) or (head, relation, ?), predict the plausible candidates from $\mathcal{E}$ to fulfill the missing entity. The link prediction can be used for adding new facts into the knowledge graph according to existing facts and answering queries. Knowledge graph completion models learn embeddings of $e \in \mathcal{E}$ and $r \in \mathcal{R}$ to provide a feasible link prediction approach. KGC models have a score function to use learned entity and relation embeddings to estimate the plausibility of a given triple. For a query (?, relation, tail) or (head, relation, ?), KGC models evaluate the plausibility of all entities in $\mathcal{E}$ to replace the missing entity. The entities with high scores are ranked higher in the prediction.

Most KGC models [12, 13, 14, 15, 14, 16, 17, 18, 19, 20, 21, 22, 23] ignore entity type constraints in relations. For example, for the relation "born-in," the head entities should be type "People," and the tail entities should be type "Locations." As a result, such ignorance degrades the link prediction performance. Many studies [24, 25, 26, 27, 28] attempted to add type information to the KGC models. These methods depend on existing type annotations in the knowledge graph to learn and use type embeddings to justify the entity type compatibility in relations. However, the type annotations are not guaranteed, and the granularity is too general to math certain entity type constraints

in relations [29].

Recent studies [1, 29] have focused on unsupervised type representation learning. The unsupervised type representation learning depends on only triples in the knowledge graph. Embeddings that represent entity type and type constraints are learned based on triples in the knowledge graph. Each relation has its embeddings for entity type constraints in the head and tail positions, and the entity embeddings are used for evaluating plausibility with respect to all relations. These methods use a type-agnostic KGC model as the base model to evaluate the interdependency of the head and the tail in a triple, and type compatibility is evaluated using entity-specific type embeddings and relation-specific type constraint embeddings.

Existing unsupervised methods have two feature spaces, one for entity and relation embeddings in the base model (i.e., a type-agnostic KGC model) and another one for the entity type and type constraint embeddings. The inconsistency of feature space results in difficulties balancing type compatibility and triple plausibility. False triples with high type compatibility may have a high plausibility estimation. For example, the triples (Donald Trump, born-in, Tokyo) and (Aragaki Yui, born-in, New York) are not true, but the type of entities "Donald Trump" and "Tokyo," "Aragaki Yui" and "New York" matches the type constraints (namely, "People" and "Location") of head and tail in relation "born-in," respectively. Additionally, many relations take diverse entity types on the head and the tail. For example, the relation "influenced-by" can take entity with following types: "Philosopher," "Actor/Actress," "Celebrity," "Scholar." However, existing methods use only one embedding to represent such constraints. They fail to capture the multiple type constraints in the relation.

I proposed two methods to alleviate these two issues by unifying the feature spaces. In the proposed methods, entity embedding locations are used to represent implicit entity types. The first method defines entity co-occurrences and uses the statistics, pointwise mutual information, to adjust entity embedding locations in bilinear KGC models. The second method drops the requirement of entity co-occurrences in the first method and generalizes the idea of the first method to bilinear and translational KGC models. In addition, although the first method uses only one embedding to capture type constraints on the head or tail location, the second method can use multiple embeddings for each relation to capture the various entity type constraints.

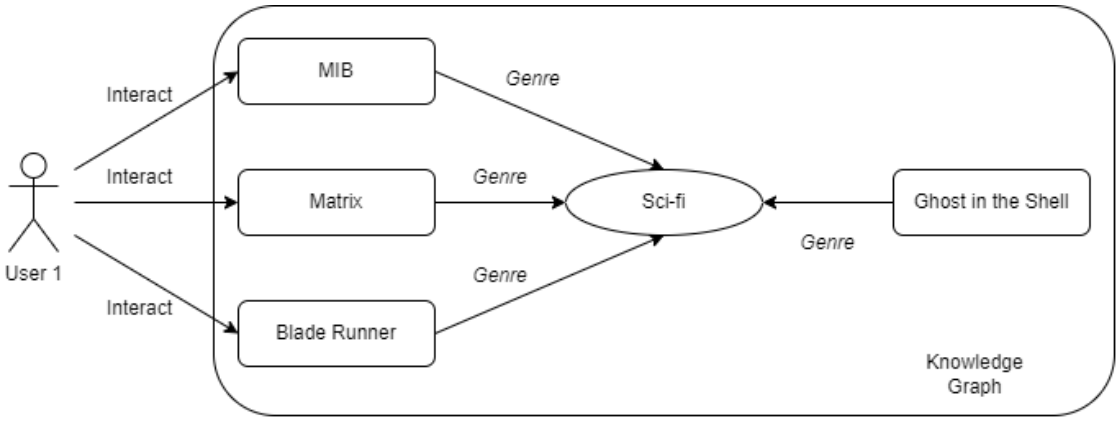


Figure 1.2: An example of knowledge graph based recommendation.

1.3 Knowledge Graph Enriched Meta-learning Recommender Systems for the User-side Cold-start Problem

As mentioned above, an important application of knowledge graphs is to improve the quality of recommender systems. For example, Amazon and Alibaba Inc. established knowledge graphs that include tremendous facts about the products for retail [30, 31]. These facts are used in their recommender systems to improve online retail services. Recent studies show that facts in knowledge graphs can help capture user interests [8, 32, 33]. These methods extend the traditional recommender systems by considering the high-order preference propagation between items. For example, in the "Movie" domain, there are many facts in the knowledge graph about attributes of movies, i.e., "Director," "Genre," "Actor/Actress"..., the contents of these attributes (i.e., "The Wachowskis" for "Director," "Sci-fi" for "Genre," and "Keanu Reeves" for "Actor/Actress") constitute movie descriptions. Figure 1.2 shows an example of the knowledge graph based recommendation with the attribute "Genre." Suppose User 1 has interacted with the movie "Matrix," "MIB," and "Blade Runner." Because these three movies have the genre "Sci-fi," the recommender system with knowledge graph will propagate such preference to other Sci-fi movies so that the Sci-fi movie will be recommended to User 1 with high priority. In this example, a movie with genre "Sci-fi" (i.e., the "Ghost in the Shell") will be recommended to User 1.

The knowledge graph based recommender system, akin to traditional recommender systems, suffers from the cold-start problem. These recommender systems require a lot of user-item interactions to learn adequate representations of users, items, and the contents in item attributes. If the number of user-item interactions is insufficient, the not well-optimized representations will result in inferior recommendation results. In the extreme case where a new user registered or a new item has been added and no interaction is available, it is referred as the user-side (item-side) cold-start problem.

This thesis concerns the user-side cold-start problem. Because in practice, the new item can be classified into pre-defined categories and recommended to users with interests in the category. However, for the new users, it is hard to use the same method, as the system has no clue about the user's interest. In practice, the system will require the new user to choose some categories from a pre-defined range and recommend the in-trend items that belong to those categories to the user. For example, Reddit will show a list of communities to its new users and recommend content according to the user's choices. Therefore, for the user-side cold-start problem, it is vital for the recommender system to rapidly capture the new user's interests and provide them with personalized recommendations.

Recent studies [34, 35, 36, 37, 38, 39, 40, 41, 42, 43] use the meta-learning framework on the user-side cold-start problem. These studies use the Model Agnostic Meta-Learning (MAML) [2] framework. The MAML-based methods include a neural network to recommend items to users. The parameters in the neural network are learned from the samples from the existing users and are used as the initial parameter for new users. The MAML-based recommender system reduces the optimization steps (i.e., the requirement of user-item interactions) to capture user interests rapidly. Moreover, because the parameters in the neural network are user-specific, it can provide personalized recommendations. From the representation learning perspective, the personalized parameters and other embeddings in the neural network recommender constitute the representation of the user preference.

Some MAML-based methods [38, 36, 34] need the user demographic information (i.e., user profile, such as age, gender, and job.) Such requirement raises privacy concerns. Furthermore, if the user demographic information is not sufficiently diverse, the recommendation quality would be inferior [44, 45]. In addition, some MAML-based methods [35, 46, 42] require a large number of records in the sample to learn the

initial parameters in the recommender neural network. Such requirement restricts the application of the system because the number of user-item interactions are not always sufficient [47].

Another problem in the MAML-based framework is related to the capability of learning personalized user-interest representations. A recent study [48] shows that the MAML-based methods tend to reuse representations (i.e., user and item embeddings and parameters in the neural network) rather than learning unique, user-specific neural network parameters. As a result, it is difficult for the MAML-based methods to capture user interests over the item attributes.

The critical factor in alleviating the cold-start problem is whether the model can fast adapt the parameters in it to the area where are nearby to the unknown optimal parameters in the parameter space. Recent studies [49, 11, 50, 51] indicate that the hypernetwork can achieve this goal.

I proposed a hypernetwork-based meta-learning recommender system for the user-side cold-start problem in this thesis. The proposed method generates the parameters in the recommender neural network instead of gradually optimizing the parameters after initialization. The proposed method learns a user-interest embedding as the representation for the user preference over different item attribute contents extracted from the knowledge graph. The parameters in the recommender neural network are generated by the hypernetwork using the user-interest embedding.

The proposed method eliminates the user demographic information requirement. It can capture user interests over different item attributes (e.g., genre, actor/actress, and directors) and the contents of the item attributes (e.g., Sci-fi, Will Smith, and Don Peterman). Because the optimization steps for learning user-interest embedding of new users are few, the proposed method can capture user preference in the case where user-item interactions are scarce (e.g., the cases of furniture and second-hand cars.)

1.4 Motivation and Contribution

The proposed methods in this thesis aim to learn better representations to help the downstream task. The methods used for representation learning depend on the task.

For the type representation learning for KGC models, this thesis describes two methods that can alleviate the issue due to inconsistent feature spaces in unsupervised

type representation learning for KGC models. The common motivation of the proposed methods is using entity embedding locations in the feature space to represent entity type and associate embeddings to relations to capture the feature space areas in which compatible entity embeddings tend to appear. The first method utilizes global statistics to adjust entity embedding locations in bilinear KGC models. The second method drops the requirement of global statistics and extends the method for both bilinear and translational KGC models. The contributions of the proposed methods are listed below.

- The proposed methods learn entity type and type constraint representations based on unsupervised learning. Unlike supervised type representation methods, the proposed methods eliminate the requirement of type annotations and can capture dynamic entity type constraints in different relations.
- The proposed methods unify feature spaces of type, entity, and relation representations. Such unification enables better balancing of type compatibility and triple plausibility. The improvement of the underlying type-agnostic KGC model for link prediction is better than other unsupervised methods.
- One proposed method can extend to both bilinear and translational KGC models. Unlike existing methods that restrict the category of underlying KGC models, the proposed method has a broad application range.

For the meta-learning recommender system for the user-side cold-side problem, I proposed a hypernetwork-based recommender system. The motivation is to learn embeddings to generate the parameters in the recommender neural network for users. All items are described by attributes in the knowledge graph. The proposed method decouples the representation of the parameters in the recommender neural network based on the hypernetwork. Each user has a user-interest embedding used in the hypernetwork to generate the personalized parameters in the recommender neural network. The weights, biases, and item attribute content embeddings are used to capture the user preference in the recommender neural network. The contributions of the proposed method are listed below.

- The proposed method requires less data than the MAML-based methods. Instead of optimizing all parameters in the recommender neural network after initializa-

tion (i.e., the MAML setting), the proposed method needs only to update the user-interest embedding used for the hypernetwork.

- The proposed method does not depend on demographic information. Because the proposed method uses the hypernetwork to generate the parameters in the recommender neural network, it does not need the user demographic information to provide appropriate initial parameters for different categories of users.

- For the scenario where the user-item interactions are scarce, the proposed method outperforms the MAML-based methods. In addition, experimental results show that the proposed method learns more appropriate initialization parameters for the underlying recommender network than other approaches.

1.5 Notation Convention

The thesis follows the notation convention used in most Graduate Texts in Mathematics book series [1] and research articles about the knowledge graph and machine learning.

Scalar numbers are in the regular font (e.g., a, b, c). Vector and matrix are in bold font. All vectors are lowercase (e.g., $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$), whereas matrices are in uppercase (e.g., $\mathbf{A}, \mathbf{B}, \mathbf{C}$). All vectors are column vectors by default.

The elements of vectors and matrices are in bold font with the same letter for the vector (matrix). The subscript denotes the position of the element. For example, the second element in the vector $\mathbf{v}$ is presented as $\mathbf{v}_2$. Similarly, the element in the first row and second column of the matrix $\mathbf{M}$ is presented as $\mathbf{M}_{1,2}$.

Tensors are in hand-writing font, e.g., $\mathcal{A}, \mathcal{B}, \mathcal{C}$. For tensors, it is similar to the notation convention for vector and matrix. The subscript denotes the element location. For a three rank tensor $\mathcal{A}$, $\mathcal{A}_{2,3,1}$ is a scalar. Note that in machine learning convention, the loss function ($\mathcal{L}$) is denoted by the hand-writing letters as well. In this case, the letter's property can be distinguished according to its context.

Sets are denoted in uppercase letters with the regular font (e.g., A, B, C) with three exceptions:

[1] cf. https://www.springer.com/series/136.

1. Number sets are in black board font. The sets for natural number, integer, rational number, real number, and complex number are $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, respectively. *It is worth to note that the natural number set includes 0*, as defined in the ISO 80000-2 standard.

2. Sets used for optimizing models (e.g., training set $\mathcal{D}_{\text{train}}$, validation set $\mathcal{D}_{\text{valid}}$, test set $\mathcal{D}_{\text{test}}$) are in the hand-writing font. The tensor and dataset can be distinguished according to the letter's context.

3. Some scalars are in capitalized regular letters to emphasis they are a member of experiment settings (epoch number N, batch size B, etc.) or have special meanings (e.g., the number of parameters).

Occasionally, the superscript on the letter indicates its range, e.g. $\mathbb{Z}^+ = \mathbb{N} - \{0\}$.

1.6 Thesis Structure

This thesis includes six chapters. The structure of the thesis is organized as follows.

The first chapter briefly represents the background, motivation, and contribution of the two research topics in the representation learning on of knowledge graph. It also introduces the notation convention used all over the thesis.

The second chapter introduces the related work of the two research topics. The second chapter presents the concept of the knowledge graph, knowledge graph completion models, and the type representation learning for the KGC models. The second chapter also reviews the concept of the model-agnostic meta-learning, the development of the meta-learning recommender system for the user-side cold-start problem, and the hypernetwork. For both topics, the corresponding sections in the second chapter have a discussion about current flaws in the reviewed existing methods.

The third and fourth chapter are related to the first research topic, i.e., the type representation learning for KGC models. Two methods I proposed are presented in these two chapters. The third chapter presents a method for the bilinear KGC models. The method proposed in the fourth chapter extends the method in the third chapter to both bilinear KGC models and translational KGC models and simplifies the data

requirement. Both methods use unsupervised learning to obtain implicit entity type and type constraint representations.

The fifth chapter presents the proposed method for the meta-learning recommender system. The proposed method learns representations for generating user-specific parameters in the neural network recommender. The representations are learned based on the hypernetwork. Item attributes and the contents of these attributes constitute item descriptions in the proposed recommender system. These attributes and contents are extracted from the external knowledge graph. By taking hypernetwork to generate parameters in the neural network recommender, the proposed system can capture the user interests in both item attributes and contents of item attributes.

The last chapter includes conclusions and summaries of the proposed methods from the perspective of representation learning on knowledge graph.

2 Related Work

This chapter introduces related work of research topics in this thesis. It starts with the introduction of knowledge graph and knowledge graph completion models. After that, it reviews the development of type representation learning in knowledge graph completion models. Next, it introduces the research incorporating knowledge graphs into recommender systems and the efforts to alleviate the user-side cold-start problem. For the related work of each topic in the thesis, this chapter discusses the shortcomings of current methods.

2.1 Knowledge Graph and Knowledge Graph Completion Models

The strict definition of knowledge graph is contentious [52]. Generally speaking, Knowledge graphs contains facts about the real world. These facts are organized in a

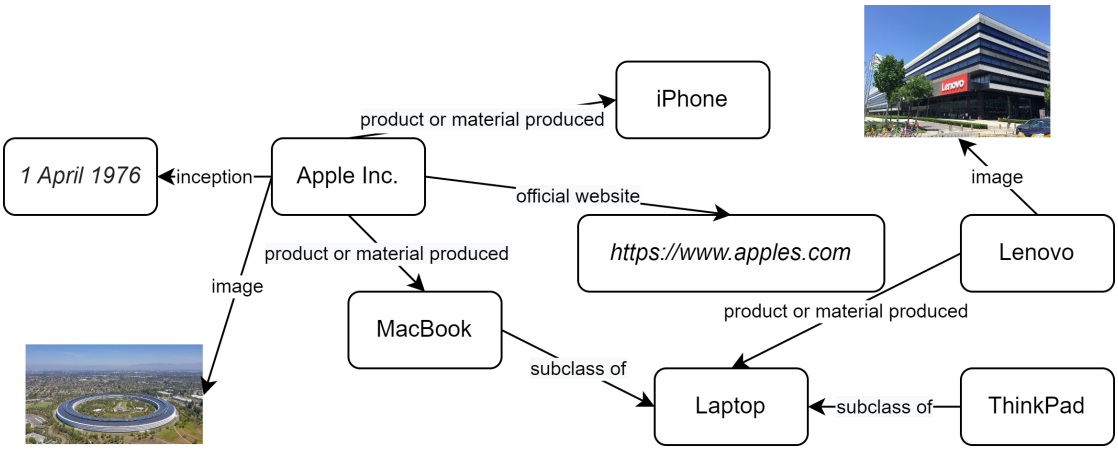


Figure 2.1: An example of a knowledge graph.

graph structure, and can be expressed in the form of triple (head, relation, tail). [1] For example, the fact "Apple Inc. is the manufacture of the M1 chip." is expressed as (Apple Inc., manufacture-of, M1 chip).

In most cases, the "head" in the triple is an entity, whereas the "tail" can be an entity, a literal, or another digital resource. Figure 2.1 shows an example of a knowledge graph as a directed graph. The facts were extracted from Wikidata (https://www.wikidata.org/) on August 7, 2022. Texts on the arcs are the name of relations. Entities are in rounded rectangles with standard font, whereas literals are with the italic font. The start vertices are "head" in the triples, and the end vertices are "tail" in the triples. For example, the triple (Apple Inc., product or material produced, iPhone) is presented as an arc starting from the vertex "Apple Inc." and ending at the vertex "iPhone." Triples with tails that are not entities describe the *property* of the head entity. For example, the "Apple Inc." has a property, "Inception," and the value of this property for "Apple Inc." is *1 April 1976*.

A knowledge graph can store facts about a specific domain or facts in the world. The choice of domain depends on the purpose of the knowledge graph. For example, WordNet [53] has facts about relations between English words; PharmKG [54] contains facts of the biomedical domain, whereas Freebase [55] are not focusing on a specific domain. The Freebase contains facts abut the real world.

A knowledge graph can accompany along with a group of schemes. The schemes

[1] In some literature, the form of the triple is also expressed as (subject, predicate, object). This naming convention is from the semantic triple.

describe how to organize facts in the knowledge graphs. For example, suppose there is a scheme (Book, has-attribute, Author), it can put constraints on the arguments (i.e., head and tail) of the relation "written-by": the head entity in "written-by" must be "Book"; and the tail entity in "written-by" must be "Author." If there is a fact (12 Rules for Life, written-by, Jordan Peterson) in the knowledge graph with this scheme, it can be inferred that:

1. 12 Rules for Life is a Book (according to the constraint on head entity in the relation "written-by.")

2. Jordan Peterson is a Author (according to the constraint on tail entity in the relation "written-by.")

3. Jordan Peterson is the Author of the Book 12 Rules for Life (according to the scheme and the fact.)

The schemes can be also organized in the graph structure. In this case, they are similar to ontology. Altough the schemes can help extend and analyze facts in knowledge graphs, they are not mandatory components in the knowledge graph.

The facts in knowledge graphs are in a discrete form. It is difficult to use them directly for other tasks. Early studies of utilizing these facts are in the perspective of relational learning. They use bayesian networks [56] or Markov networks [57] to analyze existing facts and to discover new facts according to existing ones. These methods need prior knowledge about the knowledge graph, i.e., the schemes or the ontology used for the knowledge graph. Because prior knowledge is not guaranteed, this requirement restricts the application of these methods and the knowledge graph.

Recent studies [18, 17, 19, 20, 21, 23, 22, 14, 13, 15, 12, 16, 14] model the knowledge graph in the view of representation learning. Entities and relations are represented as embeddings in these models. A score function $f(h, r, t)$ takes the embeddings of head h, relation r, and tail t to evaluate the plausibility of the triple (h, r, t) [2]. These methods do not depend on prior knowledge, and the learned embeddings can be used in models for other tasks, such as the recommender system with a knowledge graph [32]. An essential task on the knowledge graph is the link prediction, i.e., given a triple (?, r, t)

[2] Although t can be literal or another digital resource, in the KGC models, t is only considered to be an entity.

or (h, r, ?) with a missing entity, predict the candidates to fulfill the missing entity. These methods are effective in link prediction, and link prediction can help add new facts to the knowledge graph according to existing facts. Therefore, these methods are referred as Knowledge Graph Completion models.

The score function is vital in the KGC models because it evaluates the plausibility of triples. Therefore, KGC models are categorized by the form of their score functions. The KGC models in different categories are introduced in the following subsections.

2.1.1 Bilinear Knowledge Graph Completion Models

Bilinear KGC models use tensor decomposition to learn entity and relation embeddings. As the category indicated, their score functions are in bilinear form.

A representative bilinear KGC model is the RESCAL [18]. As mentioned in Section 1.2, a knowledge graph can be presented as a tensor $\mathcal{G} = \mathcal{E} \times \mathcal{R} \times \mathcal{G}$, where $\mathcal{E}$ is the set of entities, and $\mathcal{R}$ is the set of relations. If a fact (h, r, t) is in the knowledge graph, $\mathcal{G} = 1$, otherwise, $\mathcal{G} = 0$[3].

RESCAL factorizes the tensor $\mathcal{G}$ by

$$\mathcal{G}_{h,r,t} \approx f_{\text{RESCAL}}(h, r, t) = \mathbf{h} \times \mathbf{R} \times \mathbf{t}, \tag{2.1}$$

where $\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$, $\mathcal{R} \in \mathbb{R}^{d \times d}$. d is the dimension of the vector $\mathbf{h}$ and $\mathbf{t}$. $\mathbf{R}$ is a $d \times d$ matrix. $\mathbf{h}$ and $\mathbf{t}$ are embeddings for entity h and t, respectively. $\mathbf{R}$ is the embedding for the relation r. $f_{\text{RESCAL}}(h, r, t)$ is the score function in RESCAL. Entity embeddings in RESCAL are vectors, whereas relation embeddings are matrices.

The relation embeddings in RESCAL are difficult to learn due to the matrix form. DistMult [17] changes the relation embeddings in RESCAL to diagonal matrix form. The score function in DistMult is

$$f_{\text{DistMult}}(h, r, t) = \langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle = \sum_{i=1}^{d} \mathbf{h}_i \mathbf{r}_i \mathbf{t}_i, \tag{2.2}$$

[3]Note that, a triple is not in the knowledge graph not necessarily implies the triple is not a fact because most knowledge graphs are incomplete. The "closed world assumption" presumes that all facts are included in the knowledge graph. Triples not in the knowledge graph are all false, whereas the "open world assumption" presumes that triples not in the knowledge graph are not known.

where $\mathbf{r}$ is a vector that represents main diagonal elements. $\mathbf{h}$ and $\mathbf{t}$ are vectors. $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$. d is the dimension. DistMult factorizes the tensor $\mathcal{G}$ in the same way of RESCAL (this is the same for all following models if there is no additional explanation.)

Note that, in Equation 2.2, $f_{\text{DistMult}}(h, r, t) = f_{\text{DistMult}}(t, r, h)$. This property indicates that DistMult can successfully model symmetric relations (e.g., the relation "spouse-of"), but it lacks the ability of modeling asymmetric or anti-symmetric relations (e.g., "descendant-of.") ComplEx ameliorates DistMult by transferring the embeddings from real domain $\mathbb{R}$ to complex domain $\mathbb{C}$. The score function in ComplEx is

$$\begin{aligned} f_{\text{ComplEx}}(h, r, t) &= \text{Re}(\langle \mathbf{r}, \mathbf{h}, \bar{\mathbf{t}} \rangle) = \text{Re}(\sum_{i=1}^{d} \mathbf{r}_i \mathbf{h}_i \bar{\mathbf{t}}_i) \\ &= \langle \text{Re}(\mathbf{r}), \text{Re}(\mathbf{h}), \text{Re}(\mathbf{t}) \rangle + \langle \text{Re}(\mathbf{r}), \text{Im}(\mathbf{h}), \text{Im}(\mathbf{t}) \rangle \\ &+ \langle \text{Im}(\mathbf{r}), \text{Re}(\mathbf{h}), \text{Im}(\mathbf{t}) \rangle - \langle \text{Im}(\mathbf{r}), \text{Im}(\mathbf{h}), \text{Re}(\mathbf{t}) \rangle, \end{aligned} \tag{2.3}$$

where $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^d$. $\bar{\mathbf{t}}$ is the conjugate of complex embedding $\mathbf{t}$. $\text{Re}(\cdot)$ and $\text{Im}(\cdot)$ are functions that take the real and imaginary parts of a complex number, respectively. The score function in ComplEx is not symmetric because of the last term in Equation (2.3), therefore, ComplEx can model asymmetric and anti-symmetric relations.

ANALOGY [20] generalizes DistMult and RESCAL by restricting the relation matrix $\mathbf{R}$ in Equation (2.1) to be a normal matrix. Entity embeddings in ANALOGY can recover embeddings in ComplEx, and ANALOGY can support analogical inference in knowledge graphs. Other bilinear models such as TuckER [21] and SimplE [23] factorize $\mathcal{G}$ using approaches different from RESCAL. TuckER uses Tucker factorization [58] and SimplE utilizes CP factorization [59].

2.1.2 Translational Knowledge Graph Completion Models

Different from bilinear models whose motivation is from tensor decomposition, translational KGC models are inspired by language models such as word2vec [60] and Glove [61]. Similar to word embeddings with the property that $\mathbf{e}_{\text{king}} - \mathbf{e}_{\text{male}} + \mathbf{e}_{\text{female}} \approx \mathbf{e}_{\text{queen}}$, the relation embeddings in translational models transfer head entity embeddings to tail entity embeddings (this is the origin of this category's name.)

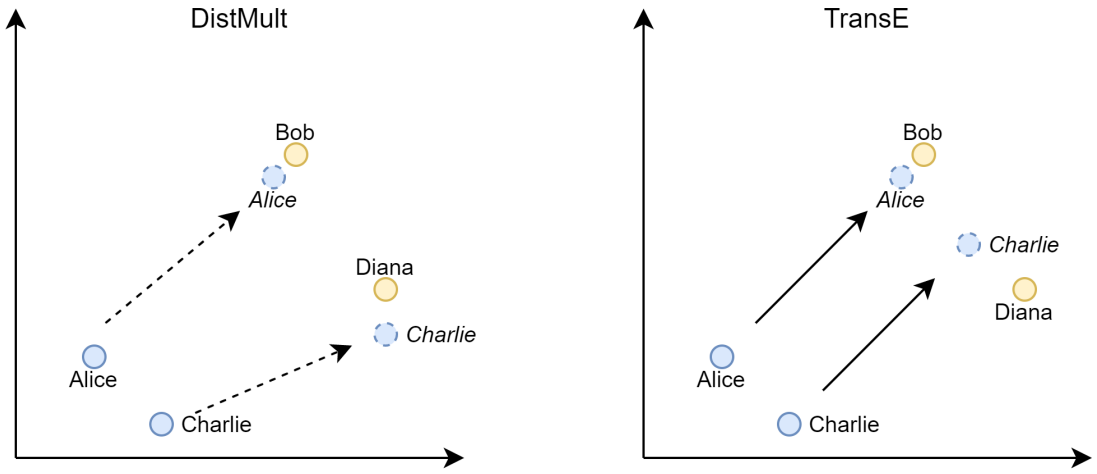


Figure 2.2: The hypothetical example for illustrating differences between the biliear model DistMult and the translational model TransE.

A typical translational model is TransE [12]. The score function in TransE is

$$f_{\text{TransE}}(h, r, t) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{1,2}, \tag{2.4}$$

where $\|\cdot\|_{1,2}$ represents the ℓ-1 or ℓ-2 norm. $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$. Note that, different from bilinear models introduced previously, the value of the score function $f_{\text{TransE}}(h, r, t)$ is in $\mathbb{R}^-$.

Figure 2.2 illustrates the differences between the typical bilinear model, DistMult, and the typical translational model, TransE, by a hypothetical example. Suppose there are four entities, "Alice," "Bob," "Charlie," and "Diana," and one relation, "spouse-of," in the knowledge graph. The knowledge graph has two facts, (Alice, spouse-of, Bob) and (Charlie, spouse-of, Diana). The example assumes all entity embeddings are in the identical locations in these two feature spaces to address the differences. The circles with names in the regular font are entity embeddings, and the dotted circles with names in the italic font in Figure 2.2 are entity embeddings after transforming by the relation embedding.

First, note that the arrows in Figure 2.2 are different. The dotted arrows in DistMult represent the shift of entity embeddings by the relation embedding. Because DistMult has a bilinear score function, the relation embedding (recall that it is a vector containing main diagonal elements of a diagonal matrix, cf. Equation 2.2.)

scales elements in the entity embeddings. The effects of scaling are presented by dotted arrows to address that DistMult does not directly move entity embeddings. In contrast to DistMult, the relation embedding in TransE moves entity embeddings with the same offsets - the length and the direction of the offset are the same (cf. Equation 2.4.), as indicated by the arrows in TransE case. As a result, the score from $f_{\text{DistMult}}(\mathbf{e}_{\text{Alice}}, \mathbf{r}_{\text{spouse-of}}, \mathbf{e}_{\text{Bob}})$ is the same as $f_{\text{DistMult}}(\mathbf{e}_{\text{Bob}}, \mathbf{r}_{\text{spouse-of}}, \mathbf{e}_{\text{Alice}})$, but $f_{\text{TransE}}(\mathbf{e}_{\text{Alice}}, \mathbf{r}_{\text{spouse-of}}, \mathbf{e}_{\text{Bob}}) \neq f_{\text{TransE}}(\mathbf{e}_{\text{Bob}}, \mathbf{r}_{\text{spouse-of}}, \mathbf{e}_{\text{Alice}})$, i.e., the DistMult can model symmetric relations, but TransE cannot.

Second, note that the moved entity embeddings of "Alice" and "Bob," "Charlie" and "Diana" diverge in a different level. The distance between the transferred entity embedding of "Alice" and the entity embedding of "Bob" is smaller than it for "Charlie" and "Diana" case. If the embedding of "Diana" deviates, TransE would evaluate the triple (Charlie, spouse-of, Bob) with high plausibility. This phenomenon becomes severe in N-to-N relations, such as "authors-of" and "act-in," and because of the same reason, TransE cannot model N-to-1 and 1-to-N relations.

Many translational models extend TransE to overcome the weaknesses mentioned above. TransH [14] projects head and tail entity embeddings to a relation-specific hyperplane. It has a relation-specific normal vector for the projection. The plausibility is evaluated similar to Equation 2.4, but TransH computes the distance of the projected entity embeddings translated by the relation embedding.

TransR [15] uses another strategy to project entity embeddings. Instead of a normal vector, TransR uses matrices on head and tail entity embeddings for projection. The score function in TransR is

$$f_{\text{TransR}}(h, r, t) = \|\mathbf{M}_r\mathbf{h} + \mathbf{r} - \mathbf{M}_r\mathbf{t}\|_2. \tag{2.5}$$

where $\mathbf{M}_r \in \mathbb{R}^{m \times n}$ is a relation-specific projection matrix. $\mathbf{h}, \mathbf{t} \in \mathbb{R}^n$ and $\mathbf{r} \in \mathbb{R}^m$. Note that, TransR evaluates the *implausibility* of triples, because its score function is in the form of distance, and f_{TransR} is in $\mathbb{R}^+$.

There are many other translational models. Generally, these models extend TransE by different strategies for projecting entity embeddings. For example, TransD [13] associates two embeddings in addition to the entity embedding for every entity, and the projection matrices for every relation are different for head and tail entity embeddings

to mitigate the impact of entity types.

2.1.3 Knowledge Graph Completion Models in Other Forms

There are some KGC models based on specific feature spaces [62, 16, 63] or neural network [22]. HoloE [64] uses holographical embeddings for entities and relations. It has been proved that embeddings in HoloE and ComplEx are equivalent under the Fourier transformation [65].

RotatE uses a feature space in complex domain $\mathbb{C}$. The relation embeddings in RotatE are complex vectors with norm 1. The relation embeddings rotate head embeddings, and the plausibility of a triple is evaluated by the distance between the rotated head embedding and the tail embedding. QuaternionE [62] extends RotatE by using quaternion numbers as the elements of entity and relation embeddings. Mathematically, the feature space of RotatE is isomorphic to the special orthogonal group SO(2), and the feature space of QuaternionE is isomorphic to SO(4). The feature space of TorusE [63] is in the torus manifold. These three methods use feature spaces related to the Lie group.

KGC models based on neural network benefit from the rapid development of deep learning. ConvE [22] takes a convolutional neural network to evaluate triples. ParamE [66] uses neural network weights as relation embeddings to assess the plausibility of triples.

2.2 Type Representation Learning for KGC Models

KGC models introduced in Section 2.1 ignore entity type when evaluating the triple plausibility. Such ignorance results in inaccurate link prediction results. Although the KGC models have some ability to infer the entity type according to the training data [18], the effectiveness is not satisfying [1].

The entity type can be a part of the schemes (if they exist) defined on the knowledge graph [67]. The entity type is also described in the triple form. For example, in the knowledge graph shown in Figure 2.1, "MacBook" is a subclass of "Laptop." This type-alike description is expressed by the "subclass of" relation [4]. Even in large-scale

[4]There is another relation, "instance of," in the Wikidata for describing entity type.

knowledge graphs, these schemes are often manually designed and maintained [68]. Therefore, the type information may not be accurate or exist in the knowledge, especially for the large-scale knowledge graphs. This is the main challenge in the type representation learning for KGC models.

This thesis categories existing type representation learning for KGC models by the underlying method. The supervised type representation learning for KGC models uses embeddings to encode the pre-defined entity types. The unsupervised method uses only triples in the knowledge graph to infer entity types and type constraints in relations. A common point in these two categories of type representation learning methods is that they take a type-agnostic KGC model as the base model and modify the score function to consider type compatibility in the head and tail of relations for link prediction. The following two subsections introduce these two methods in detail.

2.2.1 Supervised Type Representation Learning for KGC Models

The supervised type representation learning for KGC models encodes pre-existing entity types in the knowledge graph into additional parameters to evaluate the match of entities and types constraints in relations. Research in this category can be traced back to the RESCAL model, whose authors extended the original RESCAL model by encoding rdf:tag and rdf:range into additional parameters. When optimizing parameters, only entity embeddings that match the type restrictions are updated [28].

TKRL [24] considers a hierarchical structure of types when encoding entity types. TKRL uses TransE as its base model. When encoding entity types, TRKL introduces additional matrices for each relation to represent entity types and entity type constraints in the relation. Entity embeddings are projected by weighted type matrices, and in the score function they are projected again by the matrices representing the type constraints in the relation. Only entity embeddings that match the type constraints will have a higher score in link prediction.

ConnectE [25] captures entity types and type constraints by two score functions. These score functions map entity embeddings by a type matrix to match their type embeddings. The two score functions of types and triples jointly assess the plausibility of facts.

As addressed at the beginning of this section, the entity type annotations are not

guaranteed in the knowledge graph. Furthermore, even the annotations exist, since they are often a part of the schemes, their definitions are very general to keep the facts and schemes consistent. Moreover, an entity's type may shift according to the contexts in relations [29].

Take the "Apple Inc." in Wikidata as an example. On August 8, 2022, Wikidata describes the type of entity "Apple Inc." with the following types (by the "instance of" relation): "enterprise," "business," "brand," "public company," "corporation," "technology company," "computer manufacturer." Suppose there is a relation, "manufacture-of," and the tail entities in this relation are mobile devices such as "iPad," "iPhone," "Xperia," "Pixel," "Galaxy" and the head entities are the manufacturers of these devices, i.e., "Apple Inc." "Sony," "Google," and "Samsung." In this example, the entity type of "Apple Inc." is inappropriate, as they are too general to reflect these companies' property accurately. An appropriate entity type for "Apple Inc." in this relation could be "mobile device manufacturer." In addition, if the tail entities are PCs or laptops, such as "ThinkPad," "MacBook," "VAIO," "Chromebooks," and the head entities are manufacturers of these devices, "Lenovo," "Apple Inc." "Sony," "Google," etc. The entity type of the same entity, "Apple Inc." in the same relation, "manufacture-of," should be "computer manufacturer," which is presented in the WikiData. However, the entity types (described by the "instance of" relation) of "Google" in WikiData do not take it as a "computer manufacturer."

The above example indicates that the entity types could be inferred from the contexts of a specific relation, i.e., the entity's types should be consistent with the types of other entities that also appeared in the same location (head or tail), and the type constraints on arguments (i.e., head and tail) of a relation should be inferred by all entites that co-occurred in the relation. This inspires the unsupervised type representation learning methods for KGC models, which is introduced in the following subsection.

2.2.2 Unsupervised Type Representation Learning for KGC Models

Unsupervised type representation learning methods do not rely on the annotated entity types. They do not assume such information exists. The entity type representations

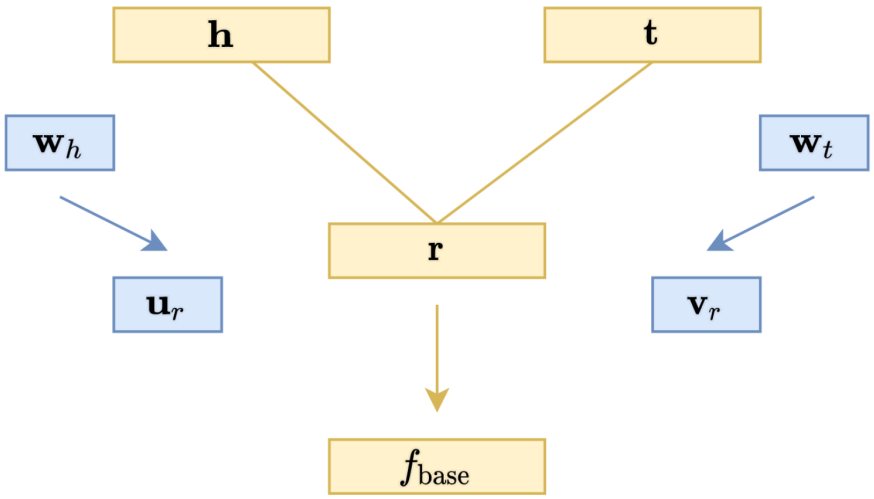


Figure 2.3: The structure of TypeDistMult and TypeComplEx [1].

are learned from the triples in which the entities appear. Because the entity type information does not exist, the unsupervised method can distinguish whether two entities are in the same type(s) but cannot represent their explicit type(s).

Jain et al. proposed TypeDistMult and TypeComplEx [1] to extend DistMult and ComplEx. TypeDistMult and TypeComplEx add a new feature space for type representations. They associate every entity with a type embedding. For every relation, they add two embeddings to represent type constraints on head and tail. The score function in these methods is

$$f_{\text{Jain}}(h, r, t) = \sigma(\langle \mathbf{w}_h, \mathbf{u}_r \rangle)\sigma(f_{\text{base}}(h, r, t))\sigma(\langle \mathbf{w}_t, \mathbf{v}_r \rangle), \tag{2.6}$$

where $f_{\text{base}}(h, r, t)$ is the score function of the base model. $f_{\text{base}}(h, r, t)$ takes the entity and relation embeddings $\mathbf{h}, \mathbf{r}, \mathbf{t}$. If the base model is DistMult, $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^n$. For ComplEx as the base model, $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^n$. $\mathbf{w}_h$ and $\mathbf{w}_t$ are entity type embeddings associated to head and tail entities, respectively. $\mathbf{u}_r, \mathbf{v}_r$ are relation-specific embeddings representing type constraints on head and tail locations of the relation r. $\mathbf{w}_h, \mathbf{w}_t, \mathbf{u}_r, \mathbf{v}_r \in \mathbb{R}^m$. $\sigma(\cdot)$ is the sigmoid function and $\langle \cdot, \cdot \rangle$ is the inner product.

The structure of TypeDistMult and TypeComplEx is shown in Figure 2.3. The different colors (yellow and blue) denote that the type embeddings $(\mathbf{w}_h, \mathbf{w}_t, \mathbf{u}_r, \mathbf{v}_r)$ and entity and relation embeddings $(\mathbf{h}, \mathbf{r}, \mathbf{t})$ are not in the same feature space. The

three terms in Equation 2.6 constitute a logical "AND" relation - only triples that have high plausibility (evaluated by $f_{\text{base}}(h, r, t)$) and high type compatibility (evaluated by $\sigma(\langle \mathbf{w}_h, \mathbf{u}_r \rangle)$ for the head location and $\sigma(\langle \mathbf{w}_t, \mathbf{v}_r)$ for the tail location.) will appear in the top of link prediction results.

AutoEter [29] takes a different strategy from TypeDistMult and TypeComplEx. AutoEter assemblies different type-agnostic KGC models to learn entity type representations. The type compatibility is evaluated by a method similar to TransE, and RotatE evaluates the triple plausibility. In addition, AutoEter uses a method that varies from TransR with the attention mechanism to encode type information inferred from triples.

The inconsistent feature space is the main problem in AutoEter, TypeDistMult, and TypeComplEx. They separate type embeddings from the entity and tail embeddings, and their score functions have three components. One is inherited from the base model, and the others evaluate entities' type compatibility on head and tail locations. Due to the inconsistent feature space, it is challenging to balance the type compatibility and triple plausibility. Recall the example in Section 1.2, false triples with high type compatibility (e.g., (Donald Trump, born-in, Tokyo)) may be incorrectly assigned with a high overall plausibility (i.e., false positive.)

Another problem is that current approaches use only two embedding for representing type constraints in relations. Many relations can take more than one type as their head and tail. For example, the relation "influence-by," the head and tail entities can have following types: "scientist," "philosopher," "politician," "comedian," "actor/actress," etc. It is difficult for a single embedding to represent such diverse type constraints.

The last problem is that current approaches restrict the type of base models that they take. TypeComplEx and TypeDistMult are designed for the bilinear KGC models, and AutoEter is an assembling method based on RotatE. It is difficult for these approaches to be generalized to other KGC models.

The unsupervised type representation learning in KGC models is a novel trend in the research of representation learning for knowledge graphs. The first research topic in this thesis belongs to this domain. The proposed methods aim to alleviate the issues mentioned above. Details of the proposed methods are in Chapter 3 and Chapter 4.

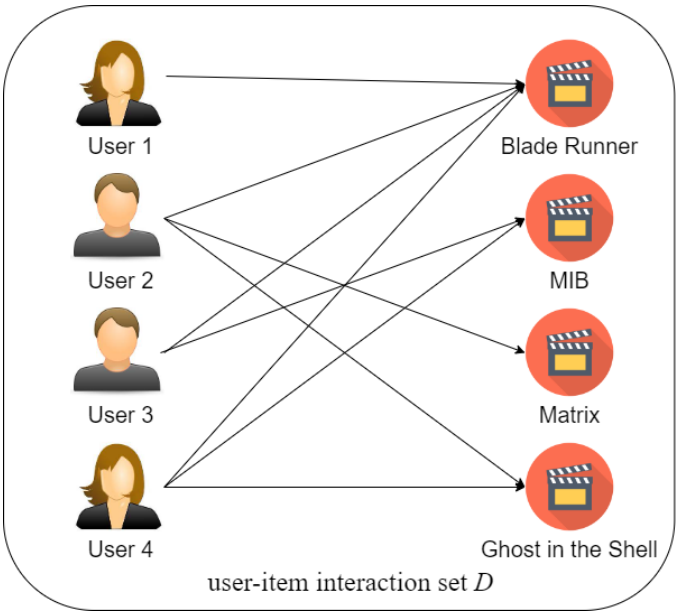


Figure 2.4: An example of user-item interactions as a bipartite graph.

2.3 Meta-learning for the User-side Cold-start Problem in Recommender Systems

Many online service providers use the knowledge graph to improve their service:

- Google [5] established Google Knowledge Graph to refine the research results [6].
- Amazon [7] built Amazon Product Graph [30] to help users discover their preferred goods.
- eBay[8] constructed eBay Product Knowledge Graph [69] to recommend items based on their properties to users.

These examples indicate an essential task of the knowledge graph is to improve the recommendation results. The mission of recommender systems is to suggest items to users based on their historical interactions.

[5] https://www.google.com/
[6] cf. https://www.blog.google/products/search/introducing-knowledge-graph-things-not/
[7] https://www.amazon.com/
[8] https://www.ebay.com/

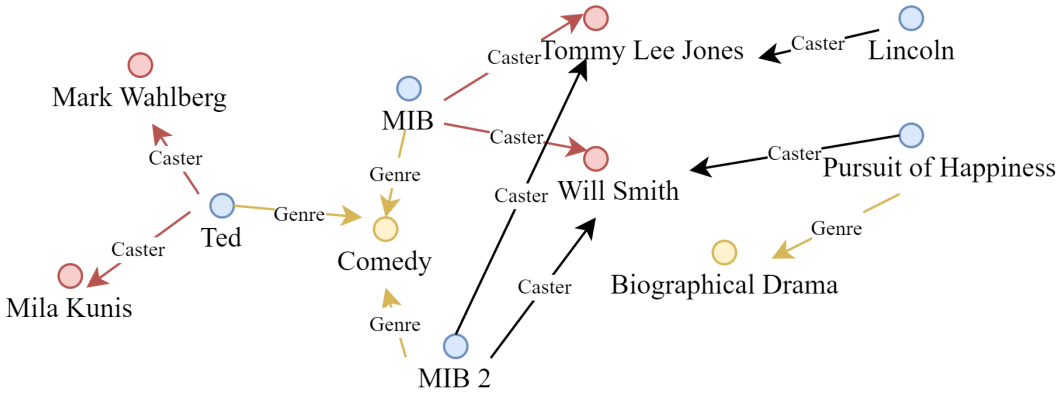


Figure 2.5: An example of capturing the high-order user preference by the knowledge graph.

Let D denote the user-item interactions in the system. D can be expressed as a bipartite graph: $D = U \times I$. U is the set of users, and I is the set of items. The recommender system suggests items $i \in I - I(u)$ to a user $u \in U$ according to u's interaction history $I(u)$. $I(u) = \{i_n, ..., i_m\}$. $I(u) \subset I$.

A basic principle for recommender systems is that they assume similar users tend to choose similar items.

Figure 2.4 shows an example of the user-item interactions in a bipartite graph. The example has four users and four items in the movie domain. The arrows in Figure 2.4 represent users' interactions with items. The interaction could be explicit (e.g., rate, like, vote) or implicit (e.g., click, save, block.) Let the interactions in the example to be "watch." Because User 3 and 4 both watched "Blade Runner" and "MIB," they could be treated as similar users. Therefore, the system can recommend "Ghost in the Shell" in User 4's historical interactions to User 3.

Many approaches [33, 8, 70, 71, 32, 72, 73] use the knowledge graph to introduce item properties to improve the recommendation results. These recommender systems use relations in the knowledge graph to capture high-order user preferences. An example is shown in Figure 1.2. In that example, the user has a high-order preference over the property "Genre" with the content "Sci-fi."

These models use undirected paths and graph convolutional neural-network (GCN) in the knowledge graph to capture such user interests. Suppose there is a knowledge graph in Figure 2.5 for the recommender system in the Movie domain, these methods

use the knowledge graph as a heterogenous information network (HIN) [9]. Different types of entities about movies are in different colors: red for "Caster," yellow for "Genre," and blue for the movie itself. They follow the pre-defined meta-paths in the HIN to integrate neighbors of an item for the representation. Assume the in-use meta-paths are "Movie—Caster—Movie" （M—C—M) and "Movie—Genre—Movie" (M—G—M), the representation (i.e., embedding) for the movie "MIB" will be constituted by "Comedy" and "MIB 2" (by M—G—M), "Will Smith" and "MIB 2" (by M—C—M), "Comedy" and "Ted" (by M—G—M), "Will Smith" and "Pursuit of Happiness" (by M—C—M), "Tommy Lee Johnes" and "Lincoln" (by M—C—M.) If a user watched the movie "MIB," the 2-hop neighbor movies (i.e., Ted, MIB 2, Pursuit of Happiness, and Lincoln) of "MIB" found according to the meta-paths are more likely to be recommended.

In general cases, these methods will integrate the user-item bipartite graph (cf. Figure 2.4 to the knowledge graph. The relations between users and items are user behaviors on items (watch, like, dislike, vote, down-vote, save, block, etc.) The meta-path can include the user behaviors for recommendation.

Training recommender systems with the knowledge graph need a tremendous number of data because of the significant number of parameters in the system. Therefore, similar to the traditional recommender systems, these recommender systems suffer from the cold-start problem as introduced in Section 1.3. The item-side cold-start problem is not extremely severe in these systems because if the item information is provided, the knowledge graph can provide appropriate neighbors according to the meta-paths. However, the user-side cold-start problem is still critical, because the knowledge graph has no users' information.

The cold-start problem's critical factor is capturing user interests with little data. Many studies [40, 34, 38, 75, 35, 41, 36, 42, 43, 46, 40, 76, 77, 39, 37] attempt to alleviate the user-side cold-start problem are with Model-agnostic Meta-learning framework [2] introduced in the following sections.

[9] A HIN is an undirected graph. The discussion about differences between HIN and knowledge graph is beyond the scope of this thesis. Interested readers can refer to the "Heterogeneous Network Representation Learning" [74] for further reading.

2.3.1 Model-agnostic Meta-learning Framework

Finn, Abbeel, and Levine proposed the model-agnostic meta-learning at the International Conference on Machine Learning in 2017 [2]. This section serves as the theoretical background of the following subsection. It briefly summarizes the overall MAML framework.

The task in MAML is defined as

$$T = \{\mathcal{L}(\mathbf{x}_1, \mathbf{a}_1, ..., \mathbf{x}_H, \mathbf{a}_H), q(\mathbf{x}_1), q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t), H\}, \tag{2.7}$$

where $\mathbf{a}_i$ is an output from a model f (a neural network). $\mathbf{x}_j$ is a observation. $q(\mathbf{x})_k$ is a observation distribution, and $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$ is the transition distribution. H is an episode length. In this thesis, all cases are with $H = 1$, as it is the setting for i.i.d. supervised learning. The loss function $\mathcal{L}(\mathbf{x}_1, \mathbf{a}_1, ..., \mathbf{x}_H, \mathbf{a}_H) \in \mathbb{R}$ provides feedbacks from a defined task (classification, regression, etc.)

Let $p(T)$ be a distribution over tasks T. $p(T)$ is the distribution that a model ought to be adapted to. In the process of meta-learning, a task T_i is sampled from $p(T)$ and a partial of data in T_i is used for optimizing the model f with the loss $\mathcal{L}_{T_i}$. The rest data in T_i is used for testing the model f. f is improved by considering how the test error on new data from q_i varies w.r.t. the parameters.

Note that the *test error* on the *sampled task* T_i serves as a *training error* of the entire meta-learning process. When the training stage is over, new samples that are different from samples that have been drawn from $p(T)$ are used for testing the performance of f in the validation and test stages.

Figure 2.6 shows an example with three drawn task and corresponding gradients from the losses $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$. θ is the parameters in f. At the beginning of training, θ is randomly initialized. Three tasks T_1, T_2, T_3 are drawn from the distribution $p(T)$ and the corresponding gradients $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ are used for optimizing θ in f.

After the training stage, the model takes θ for new tasks and uses partial data from the new tasks to optimize θ. After the training stage, θ will not be updated globally. For each new task, f uses the θ that is optimized in the *training* stage (i.e., meta-learning stage) as the initial parameters (the θ at the end of the arrow in Figure 2.6) and θ will be only updated according to the partial data in the new task, as shown by the dotted line in Figure 2.6. Each new task has its own parameter, $\theta_1^*, \theta_2^*, \theta_3^*$.

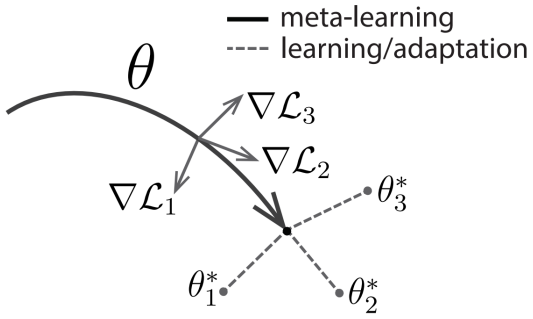


Figure 2.6: An example cited from the MAML paper [2] to introduce the framework. The figure shows the feature space of the parameters in f.

Formally, the entire training stage is described in Algorithm 2.1.

Algorithm 2.1 Model-Agnostic Meta-Learning (from the MAML paper [2])

Require: $p(\mathcal{T})$: distribution over tasks.
Require: α, β: step size hyperparameters.
1 Randomly initialize parameters θ;
2 **while** *not done* **do**
3 Sample batch of tasks $T_i \sim p(T)$
4 **forall** T_i **do**
5 Evaluate $\nabla_\theta \mathcal{L}_{T_i}(f_\theta)$ w.r.t. K examples in T_i
6 Compute θ_i' by gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta)$
7 Update $\theta := \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta_i'})$

Note that, Algorithm 2.1 is independent from the task type of the underlying model f. The task type decides the way of implementing gradient descent in line 5. After optimization, the parameter θ in f can be used for initializing the model f on new tasks. In addition, MAML only assumes the i.i.d. of tasks.

The MAML framework is widely used in methods that attempt to alleviate the user-side cold-start problem because it supports fast parameter adaption. The recommender systems based on MAML can provide personalized recommendation. The related research is introduced in the following section.

2.3.2 Meta-learning Recommender Systems for the User-side Cold-start Problem

The meta-learning recommender systems for the user-side cold-start problem implement the framework in Section 2.3.1 by setting each task as a user. The model f in the meta-learning framework is a neural network for the recommendation.

MeLU [38] is a representative model that adopts meta-learning for the cold-start problem. Each user is a task in MeLU. The user's interactions are used for training. It uses a neural network $f(\mathbf{u}, \mathbf{i})$ that takes a user embedding $\mathbf{u}_n$ and $\mathbf{i}_m$ to estimate the user's preference. The user embeddings are constructed by user demographic, and item properties construct the item embeddings. MeLU predicts a score to represent user preference:

$$\hat{y}_{n,m} = f_\theta(\mathbf{u}_n, \mathbf{i}_m), \tag{2.8}$$

where $\hat{y}_{n,m}$ is the predicted preference of the user n for the item m. f is the neural network for estimating user preference with the parameters θ. $\mathbf{u}_n$ and $\mathbf{i}_m$ are embeddings for user n and item m, respectively.

The user-item interaction dataset D is divided into training, validation, and test sets. In each dataset, every user's interactions are divided to two sets: the *support* set $\mathcal{S}_u$, and the *query* set $\mathcal{Q}_u$. The support set is used for updating θ locally, whereas the query set is used for updating the global parameters that correspond to θ. Items in $\mathcal{S}_u$

Algorithm 2.2 The Meta-learning stage for MeLU.

Require: α, β: step size hyperparameters.

Randomly initialize parameters $\mathbf{u}, \mathbf{i}$ for all users and items;
Randomly initialize parameters θ;
while *not done* **do**
  Sample batch of users $B \sim p(U)$
  for *every user* $j \in B$ **do**
    Set $\theta^j = \theta$
    Evaluate $\nabla_{\theta^j} \mathcal{L}_j(f_{\mathbf{u},\mathbf{i},\theta^j})$
    Local update $\theta^j := \theta^j - \alpha \nabla_{\theta^j} \mathcal{L}_j(f_{\mathbf{u},\mathbf{i},\theta^j})$
  Global update $\mathbf{u} = \mathbf{u} - \beta \sum_{j \in B} \nabla_{\mathbf{u}} \mathcal{L}_j(f_{\mathbf{u},\mathbf{i},\theta^j})$
  Global update $\mathbf{i} = \mathbf{i} - \beta \sum_{j \in B} \nabla_{\mathbf{i}} \mathcal{L}_j(f_{\mathbf{u},\mathbf{i},\theta^j})$

and $\mathcal{Q}_u$ are not overlapped. Algorithm 2.2 displays the meta-learning stage for MeLU.

The loss function $\mathcal{L}_j$ is

$$\mathcal{L}_j = \frac{1}{|\mathcal{D}|} \sum_{k \in D} (y_{j,k} - \hat{y}_{j,k})^2, \tag{2.9}$$

where $\mathcal{D}$ is the support set $\mathcal{S}_j$ for local update and the query set $\mathcal{Q}_j$ for global update. j is a sampled user, and k is a item that user j consumes. $\hat{y}_{j,k}$ is the predicted score by f_{θ^j} for user j about item k, and $y_{j,k}$ is the real score.

Note that Algorithm 2.2 implements Algorithm 2.1 for the cold-start problem in the recommender system. The entire meta-learning stage in MeLU has two phases: local update and global update.

- Local update refers to line 11 to line 15 in Algorithm 2.2. The parameter optimization in line 14 and line 15 uses partial user-item interactions of user j. The parameter θ^j is the user-specific parameters for user j. The optimization in local update uses data in the support set $\mathcal{S}_j$ for user j. This stage is named "local update" because updates (optimization) in θ^j do not affect parameters for other users.

- Global update refers to line 16 to line 17 in Algorithm 2.2. This stage uses the rest of the user-item interactions of user j, i.e., the interactions that are not used for local updates in the query set $\mathcal{Q}_j$ for user j, to optimize user and item embeddings $\mathbf{u}$ and $\mathbf{i}$. Because these embeddings are shared across all users (the user embedding is constructed by their profiles: age, gender, job, etc.) Therefore, this stage is named "global update" because it optimizes these global embeddings.

In the validation and test stages, MeLU takes a modified version of Algorithm 2.2. The global update in line 16 and 17 are replaced by testing the model f_{θ^j} for every user j in B on the corresponding query set $\mathcal{Q}_j$.

The experiments in MeLU show that the parameters in f can fast adapt to users. Therefore, it can alleviate the user-side cold-start problem and provide personalized recommendations. Many approaches [40, 34, 75, 35, 41, 36, 42, 43, 46, 40, 76, 77, 39, 37] have been proposed to improve these abilities. All these methods follow a similar training and testing setting to MeLU:

- Taking support set for local update;
- using query set for global update;
- testing the model after few updates on the support set.

MAMO [34] extends MeLU by a memory-augumented neural network [78]. It associates a user profile memory and a user memory to provide different global parameters for different users. Every user is described by a mixture of components in the user profile memory (a matrix). The user memory is used for personalized recommendations for every user. In addition, MAMO has a parameter memory that saves the global parameters for initialization in the new user's case. All of these memories are global parameters, which means they are shared across all users. In the local update, the user-specific parameter in the recommender network f is initialized by using these three memories and locally updated by the user's interactions.

PAML [41] modifies MeLU on the learning rate aspect. They change the stochastic gradient descent in Algorithm 2.2, lines 15 - 17. They proposed an approach that takes an adaptive learning rate (similar to AdaGrad [79]) to replace the SGD process in the MeLU model. The adaptive learning rate is user-specific, so it enables the model to adapt to preference changes for different users at different speeds.

Meta-SSIN [37] considers the problem of capturing user interests in multiple scenarios. It uses three neural networks, meta-scenario interest units for computing scenario representations, a scenario attention unit to compute the similarity between items and scenarios, and a recommender network to calculate the user's interest in an item. Meta-SSIN focuses on providing diverse personalized recommendation results. It does not concern the cold-start problem.

Different from Meta-SSIN, some methods [76, 39] adjust MeLU for specific domains. For example, DCDIR [76] use the meta-learning approach for providing personalized recommendations in the insurance domain, and MFNP [39] is in the travel domain. It recommends the next place that a user wants to travel by meta-learning.

MetaHIN [36] methods attempt to integrate the knowledge graph to the meta-learning process. It takes the meta-paths approach introduced at the beginning of Section 2.3 to fast capture users' high-order preference. MetaTL [40] use the user-item bipartite graph, and use embeddings for user behaviors as translation vectors between the user embedding and the item embedding.

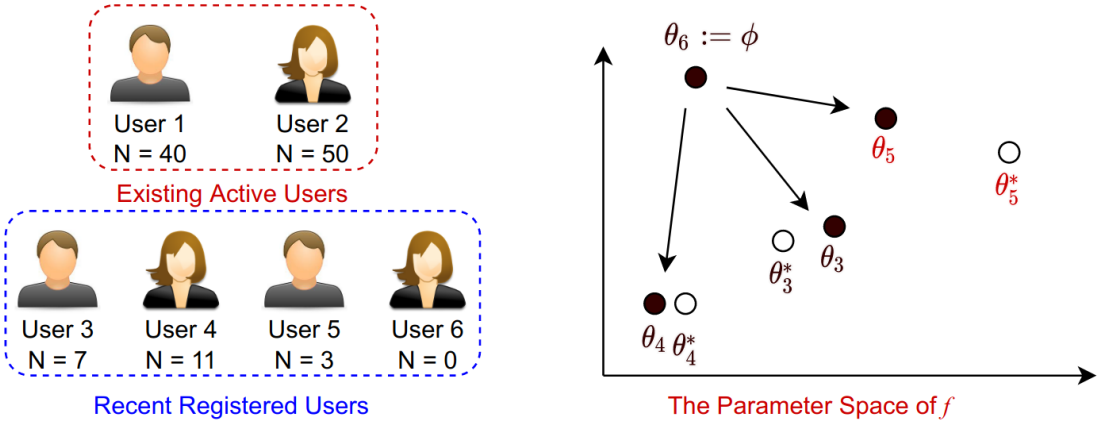


Figure 2.7: An example of MAML-based methods and their flaws

These methods need the user profile, i.e., user demographic (gender, job, age, location, etc.) This requirement raises privacy concerns, and users are probably unwilling to provide factual information. In addition, online services in multiple regions with different jurisdictions may have potential legal issues (cf. General Data Protection Regulation, GDPR [80].) In addition, if the aspects in the user profile is not diverse, it is difficult to adapt the parameters θ in f_θ quickly, because the initial parameters (i.e., the learned global parameter) in f may need many interactions to be optimal for the user [41].

Figure 2.7 shows a hypothetical example of the MAML-based methods. N is the number of user-item interactions. The global parameter ϕ is learned from existing users: User 1 and User 2. For new registered users (User 3 to User 6), the parameters in the recommender network f_θ are initialized by the global parameter ϕ. For the new users with (relatively) many interactions, their parameters in f can reach the areas close to the optimal location. User 3 (7 item interactions) and User 4 (11 item interactions) belong to this case, their parameters are initialized by the ϕ, and their interactions are used to optimize their parameters θ_3 and θ_3, respectively, so they are nearby the corresponding (unknown) optimal parameters θ_3^*, θ_4^* in the parameter space.

However, User 5 has little interactions - they only has 3 interactions. As a result, their parameters θ_5 in the recommender network f are far away from the optimal parameters θ_5^*. In this case, the recommendation results for User 5 are not satisfying.

Some studies [43, 75, 77, 81] attempt to accelerate the parameter adaption. MetaDNN

[43] changes the recommender network f in MeLU to a deeper, wider network. The training procedure in MetaDNN is designed to utilize deep neural networks to capture user interests with fewer interaction data. s^2Meta takes the REINFORCE [82] algorithm to accelerate the parameter adaption. The s^2Meta takes a reinforcement learning approach that is similar to CAVIA [83], which is a mixture approach of reinforcement learning and meta-learning. ProtoCF [77] uses the prototype network [84]. It captures user interests in item prototypes to increase the adaption speed. Similarly, Zhu et al. proposed a method [81] to warm up embeddings by a scaling and shifting neural network. They use existing well-trained embeddings to help optimize the new embeddings. Their work focus on the item-side cold-start problem, but the same technique can apply to the user-side cold-start problem as well.

A central problem in the MAML framework is that, although the approach requires only a few data for every user, it needs many users to train the global parameters: the item embeddings and the embeddings for aspects in the user (if required) need many data to optimize. Besides, the global parameter for initializing f may also need a considerable number of data, depending on the complexity of the underlying recommender network.

Moreover, the embeddings and the global parameter for initializing the underlying recommender network can entangle. If, in the middle of the training stage, the embeddings are well optimized, then in the later parameter adaption phase, the gradients from the well-optimized embeddings are not in a large magnitude. As a result, it is difficult for the recommender system to capture user interests with little data due to this feature reuse problem [48, 85].

If one re-thinks the MAML framework, it is clear that learning a global parameter for initializing is the critical factor for the cold-start problem because it allows the parameter to be adapted with little data. Current methods use gradient-based methods to optimize the global parameter, and if the gradient magnitude is small, the models cannot achieve the fast adaption of parameters.

The proposed method in this thesis aims to alleviate the issue of feature reuse and integrate the knowledge in the knowledge graph to empower the recommender system. Different from existing approaches, the proposed method is based on hypernetwork. Global parameters for initializing the recommender network for new users are not optimized with gradients from existing users along with a trajectory (as shown in

Figure 2.6). Instead, every user has two embeddings - one for generating parameters for the fast-adapt, personalized recommendation, and another for computing their preference over items. Details of the proposed method are in Chapter 5, and the following section reviews hypernetwork and its application in meta-learning.

2.3.3 Hypernetwork and Meta-learning

Hypernetwork [11] is proposed by Ha, Dai, and Le at the International Conference on Machine Learning 2017. A representative property of the hypernetwork is that it allows the parameters in a neural network to be generated by another neural network. Different from previous attempt with the dynamic weights [86], the hypernetwork enables end-to-end training. Hypernetwork uses one or more representations to generate the parameters in the other neural network. Figure 2.8 shows an example of a hypernetwork structure.

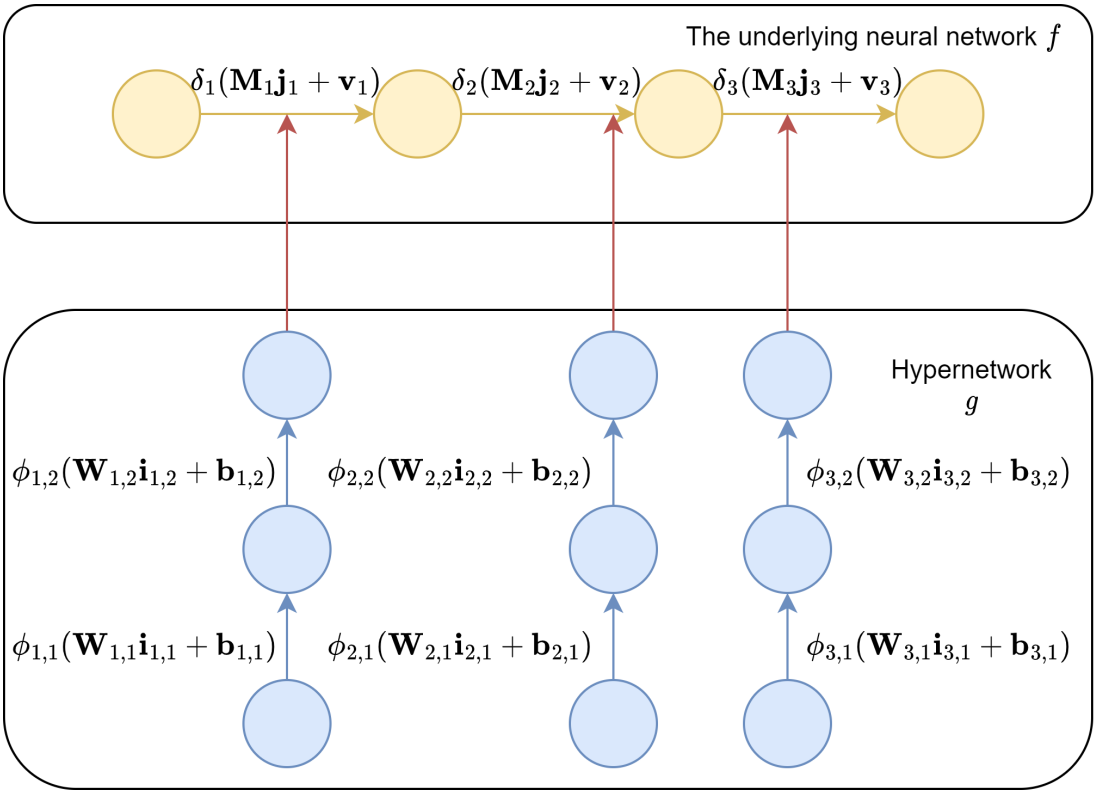


Figure 2.8: An example of hypernetwork structure

In Figure 2.8, there are two neural networks: f for the pre-defined task (classification, regression, etc.) and g for generating parameters in f. $\mathbf{j}_1$ is the input for the underlying neural network, and $\mathbf{j}_3, \mathbf{j}_3$ are transformed input after feed-forward. Similarly, $\mathbf{i}_{m,1}$ is the m-th input for the hypernetwork ($m = 1, 2, 3$) , and $\mathbf{i}_{m,2}, \mathbf{i}_{m,3}$ are transformed input after feed-forward. The parameters in f are

$$\{\mathbf{M}_n, \mathbf{v}_n | n = 1, 2, 3\}. \tag{2.10}$$

These parameters are generated by the corresponding inputs $\mathbf{i}_n$, $n = 1, 2, 3$ in the hypernetwork g, as the brown arrows indicate:

$$\begin{aligned} \mathbf{M}_n, \mathbf{v}_n &= \phi_{n,2}(\mathbf{W}_{n,2}\mathbf{i}_{n,2} + \mathbf{b}_{n,2}) \\ \mathbf{i}_{n,2} &= \phi_{n,1}(\mathbf{W}_{n,1}\mathbf{i}_{n,1} + \mathbf{b}_{n,1}), \end{aligned} \tag{2.11}$$

where $\phi_{n,1}$ and $\phi_{n,2}$ are activation functions in the hypernetwork. Suppose the matrix $\mathbf{M}_n$ contains $K \times V$ elements, and the bias vector $\mathbf{v}_n$ contains K elements. The hypernetwork generates $(K+1) \times V$ elements and splits these elements into corresponding parameters in the underlying neural network [10].

Note that all parameters and inputs in Figure 2.8 can be optimized by back-propagation [87] if all activation functions (δ_n and $\phi_{n,m}$, $n = 1, 2, 3$ and $m = 1, 2$) are differentiable. The parameters that need gradients from the loss function are parameters in the hypernetwork, e.g., $\mathbf{W}_{n,m}$ and $\mathbf{b}_{n,m}$, $n = 1, 2, 3; m = 1, 2$.

Zhao et al. show that the hypernetwork can be used for meta-learning, and it is not affected by the feature reuse as severe as the MAML-based methods do [88].

Formally, Zhao et al. assume that a neural network f with parameters as a vector of real number: $\mathbf{w} \in \mathbb{R}^{N_w}$ that are partitioned into C chunks:

$$\mathbf{w} = [\mathbf{w}^{(1)}, ..., \mathbf{w}^{(C)}].$$

Every chunk is with N_c dimension. $N_c = N_w/C$ (assuming that $N_c \in \mathbb{Z}$). They define a

[10] In the paper of hypernetwork [11], the weights and bias are genereted by a tensor and a matrix, respectively. The modification of generation in this thesis does not affect the explanation.

set of template weight vectors in a matrix:

$$\mathbf{M} = [\mathbf{t}_1, ..., \mathbf{t}_K].$$

$\mathbf{t}_i \in \mathbf{R}^{N_c}$ for $i = 1, ..., K$. The chunk $\mathbf{w}_i$ can be generated by a linear combination of template weight vectors:

$$\mathbf{w}^{(i)} = \sum_{k=1}^{K} \alpha_k^{(i)} \mathbf{t}_k = \mathbf{M}\alpha.$$

$\mathbf{M}$ is shared across all $\mathbf{w}_i$. A study indicates that the parameter α, which represents how to reuse $\mathbf{t}_k$ across layers in f, can be used to share feature information [49]. Similar to Algorithm 2.1, the meta-learning with hypernetwork has local and global update stages. The sampled task $T_i \sim p(T)$ (cf. Algorithm 2.1) also has a support set $\mathcal{S}_{T_i}$ and a query set $\mathcal{Q}_{T_i}$.

At the beginning of the local update, the parameters in the neural network f is generated by $\alpha_{T_i} = \alpha$. The local update optimizes the weight parameter α by gradients from the loss function $\mathcal{L}$:

$$\alpha_{T_i} = \alpha_{T_i} - \eta \frac{1}{|\mathcal{S}_{T_i}|} \sum_{(x,y)\in\mathcal{S}_{T_i}} \nabla_{\alpha_{T_i}} \mathcal{L}\big(f_{\alpha_{T_i}}(x), y\big). \tag{2.12}$$

where η is the learning rate for the local update stage. (x, y) are data in the support set $\mathcal{S}_{T_i}$. f_α is the neural network for predicting y according to x. The subscript α_{T_i} is to address that the parameters in f are generated by a mixture component of $\mathbf{M}$ with weights in α_{T_i}.

In the global update stage, the α and $\mathbf{M}$ are updated:

$$\begin{aligned}
\alpha &= \alpha - \gamma \frac{1}{B} \sum_{i=1}^{B} \frac{1}{|\mathcal{Q}_{T_i}|} \sum_{(x,y)\in\mathcal{Q}_{T_i}} \nabla_\alpha \mathcal{L}\big(f_{\alpha_{T_i}}(x), y\big), \\
\mathbf{M} &= \mathbf{M} - \gamma \frac{1}{B} \sum_{i=1}^{B} \frac{1}{|\mathcal{Q}_{T_i}|} \sum_{(x,y)\in\mathcal{Q}_{T_i}} \nabla_{\mathbf{M}} \mathcal{L}\big(f_{\alpha_{T_i}}(x), y\big).
\end{aligned} \tag{2.13}$$

where γ is the learning rate for the global update stage. B is the number of sampled tasks. $\mathcal{Q}_{T_i}$ is the query set in the sampled task T_i.

The hypernetwork-based meta-learning shares some common points with the MAML-based meta-learning in Algorithm: they both have two stages, one for updating the parameters for providing personalized results (the local update), and the other one for updating the parameters that fit to all sampled tasks (the global update); they assume the tasks are following a certain yet unknown distribution $p(T)$ and use samples T_i from the distribution to learn parameters that are useful for other tasks. However, unlike the MAML framework, hypernetwork-based meta-learning obtains the initial parameters for the underlying neural network f by a linear combination (i.e., weights in α) of vectors (i.e., vectors in $\mathbf{M}$ that span over the parameter space).

Note that Zhao et al. take the initial weights α as a global parameter that needs optimizing. This setting can be changed according to the underlying task. Because the initial parameters for f are not optimized with gradients from the loss function, the initial parameters are not following a trajectory anymore. Therefore, the hypernetwork suffers less from the feature reuse problem caused by the insignificant gradients for learning the initial parameters for f.

In addition, it worth to address that the dimension of feature space is decided by $\mathbf{M}$ because the parameter space is spanned by the column vectors in $\mathbf{M}$. The dimension of α does not determine the dimension of feature space. The hypernetwork is more flexible than the MAML framework because the dimension of the feature space depends on the choice of α and $\mathbf{M}$.

Raphael et al. applied the hypernetwork for image generation under the meta-learning setting [51]. Moreover, Shamsian et al. have shown that the hypernetwork can be used for providing personalized models for a given task [50]. Lamb et al. show that the hypernetwork can adapt to new features rapidly [89]. These studies encouraged the proposed method in this thesis. The proposed method uses a hypernetwork to establish a recommender system that can adapt to user preference with little data and fast-adapt to new features (i.e., item properties). In addition, the proposed method can capture the user's interest in item properties themselves, not only the contents of the item properties. Details of the proposed method are introduced in Chapter 5.

3

Unsupervised Type Constraint Inference in Bilinear Knowledge Graph Completion Models

This chapter introduces the first proposed method for learning type representations in type-agnostic knowledge graph completion models. The proposed method in this chapter focus on unifying the feature space of type representations and entity and relation embeddings. It utilizes global statistics in the knowledge graph, namely, the entity co-occurrences, to adjust and refine the entity emebddings, and associate two type constraint embeddings on the head and tail locations of every relation. The experiments are conducted with DistMult [17] and ComplEx [19] as the base models for the proposed method, and the benchmark models are TypeDistMult and TypeComplEx [1] proposed by Jain et al. The experiment settings follow the instructions in the benchmark's paper and the implementation code in their GitHub repository [1]. The

[1] cf. https://github.com/dair-iitd/KBI

experiment results show that the proposed method can improve link prediction better than the benchmark models and has more preferable entity clustering results. The proposed method in this chapter has been published as a conference paper in IEEE International Conference on Big Knowledge, 2021 [90]. The proposed method in this chapter is a pioneer research that inspires the research in the next chapter.

3.1 Propagate Implicit Type Information Across Relations

As mentioned at the beginning of this chapter, the purpose of this method is to unify the feature spaces of entity and relation embeddings and type representations. The introduction of the proposed method starts with a central definition of this model and the definition of entity co-occurrence, which is a critical concept for the proposed method.

3.1.1 The Assumption and Definition of Entity Co-occurrence

The proposed method has a straightforward assumption: *entities that frequently co-occur in multiple relations are of the same type.* If two entities, e_1 and e_2, If two entities, e_1 and e_2, appear at the same location (head or tail) of a relation, then this appearance counts as one co-occurrence of e_1 and e_2.

This assumption is quite intuitive. If e_1 and e_2 appear together across many relations $r_1, ..., r_n$, they are more probable to be the same type because they both satisfy the (very probable) different type constraints in these relations. Formally, the co-occurrence of entities can be defined as

$$\mathbf{X}_{i,j} = \sum_{r \in \mathcal{R}} \text{head}_r(e_i) \times \text{head}_r(e_j) + \text{tail}_r(e_i) \times \text{tail}_r(e_j), \tag{3.1}$$

where $\mathcal{R}$ is the set of relations in a knowledge graph. $\text{head}(\cdot)$ and $\text{tail}(\cdot)$ are two functions that take the frequency of an entity in the head or tail location of the relation r, respectively. $\mathbf{X}$ is the matrix for entity co-occurrences in all relations. The element $\mathbf{X}_{i,j}$ represents the co-occurrence of entity i and entity j.

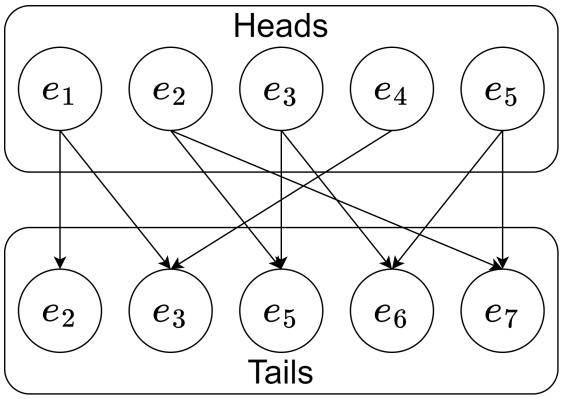


Figure 3.1: An example of entity co-occurrence in a relation.

Figure 3.1 shows an example of the entity co-occurrence in a relation r. In this relation, the head and tail location contains five entities. The arrows indicate the fact. For example, there are four facts about e_3, among which two use e_3 as the head entity, and the other two use e_3 as the tail entity: (e_3, r, e_5), (e_3, r, e_6), (e_4, r, e_3), (e_1, r, e_3). Similarly, e_2 occurs in r as the head for two facts and as the tail for one fact. Therefore, $\text{head}(e_3) = 2$, $\text{head}(e_2) = 2$, $\text{tail}(e_3) = 2$, $\text{tail}(e_2) = 1$, the co-occurrence of e_2 and e_3 in r is

$$\mathbf{X}_{2,3} = \text{head}(e_2) \times \text{head}(e_3) + \text{tail}(e_2) \times \text{tail}(e_3) = 2 \times 2 + 1 \times 2 = 6. \tag{3.2}$$

The definition in Equation 3.1 plays a critical role in inter-relation type information propagation. First, note that in Equation 3.1, if two entities e_i and e_j occur in many relations, $X_{i,j}$ will be large. Second, note that $X_{i,j}$ will be significant if e_i or e_j has a high frequency in all relations. Therefore, entities with high frequency are responsible for propagating type information across relations.

3.1.2 Empirical Conditional Probability of Entities Defined by Co-occurrence

The co-occurrence matrix $\mathbf{X}_{i,j}$ is symmetric and is used for defining empirical conditional probability of two entities. The entity and type constraint embeddings are calibrated according to this empirical conditional probability. The empirical conditional probability

is defined as follows.

$$P(e_i|e_j) = \frac{\mathbf{X}_{i,j}}{\sum_{n=1}^{|\mathcal{R}|} \mathbf{X}_{i,n}}. \tag{3.3}$$

$|\mathcal{R}|$ represents the number of relations. According to Equation 3.1 and Equation 3.3, if $P(e_i|e_j)$ is large, e_i and e_j are likely to appear together in the same argument (head or tail) across all relations. Therefore, they are more probable to have the same type because they satisfy multiple type constraints in the same location of many relations.

Consider three entities, e_i, e_j, and e_k that e_i and e_j both frequently co-occur with e_k, but e_i and e_j never co-occur, the type implied by the co-occurrence of e_i, e_k and e_j, e_k should be propagated from e_i to e_j by e_k. This propagation is by $P(e_k|e_i)$ and $P(e_k|e_j)$. Take a function $F(e_i, e_j, e_k)$ as

$$F(e_i, e_j, e_k) = \frac{P(e_k|e_i)}{P(e_k|e_j)}. \tag{3.4}$$

The definition is similar to the method that measures the word relatedness in GloVe [61] model. In GloVe, e_k represents the contextual words. The idea is to calibrate embeddings of e_i, e_j, and e_k by Equation 3.4.

3.1.3 Calibrate Entity Embeddings with Empirical Conditional Probability

This section concerns about use F in 3.4 to refine entity embeddings with type-agnostic bilinear knowledge graph completion models. The method in this subsection is similar to the approach of learning word embeddings in the GloVe.

The general motivation is to express F in Equation 3.4 in the form of entity embeddings. Because F is a scalar, and entity embeddings $\mathbf{e}_i, \mathbf{e}_j, \mathbf{e}_j \in \mathbb{R}^n$ where n is the embedding dimension. Therefore, the F can be constituted by the inner product as follows. For entity embeddings with complex numbers as elements, the real and imaginary parts are concatenated to map them in real domains.

$$F\big((\mathbf{e}_i - \mathbf{e}_j)^T \mathbf{e}_k\big) = \frac{P(e_k|e_i)}{P(e_k|e_j)}. \tag{3.5}$$

Let $F(\mathbf{e}_i^T \mathbf{e}_k)$ be defined as

$$F\left(\mathbf{e}_i^T \mathbf{e}_k\right) = P(e_k|e_i) = \frac{\mathbf{X}_{i,k}}{\sum_{n=1}^{|\mathcal{R}|} \mathbf{X}_{i,n}}. \tag{3.6}$$

Substitute Equation 3.6 into Equation 3.5:

$$F\left((\mathbf{e}_i - \mathbf{e}_j)^T \mathbf{e}_k\right) = \frac{F\left(\mathbf{e}_i^T \mathbf{e}_k\right)}{F\left(\mathbf{e}_j^T \mathbf{e}_k\right)}. \tag{3.7}$$

The next step is to choose a function that satisfies both Equation 3.6 and Equation 3.5. A choice [2] is to use the exponential function:

$$F(\cdot) = \exp(\cdot).$$

The inner product becomes

$$\mathbf{e}_i^T \mathbf{e}_k = \log(P(e_k|e_i)) = \log(\mathbf{X}_{i,k}) - \log(\mathbf{X}_i). \tag{3.8}$$

Equation 3.6 enables the model to refine its entity embeddings according to co-occurrences. Because Equation 3.6 takes the inner product to represent the empirical conditional probability, it naturally fits into the bilinear knowledge graph completion models.

The loss function for calibrating entity embeddings is

$$\mathcal{L}_{\text{refine}} = \sum_{i,j=1,X_{i,j}>0}^{|\mathcal{E}|} g(\mathbf{X}_{i,j})\left(\mathbf{e}_i^T \mathbf{e}_j + b_i + b_j - \log \mathbf{X}_{i,j}\right)^2, \tag{3.9}$$

where $g(\cdot)$ is a weighting function:

$$g(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max}, \\ 1 & \text{otherwise.} \end{cases} \tag{3.10}$$

$x_{\max}$ and α are two hyperparameters. $g(\cdot)$ composes an upper bound of the co-occurrence to prevent the co-occurrences from dominating the entity embeddings over the facts in the knowledge graph. In Equation 3.9, b_i and b_j are two biases that absorb

[2] this is inherited from GloVE

$\log(\mathbf{X}_i)$ because $\log(\mathbf{X}_i)$ is independent from e_k. $\mathbf{e}_i$ and $\mathbf{e}_j$ are entity embeddings for e_i and e_j. Equation 3.5 to Equation 3.9 propagate the implicit type information inferred by co-occurrence from e_i to e_j via e_k as introduced in the previous Subsection 3.1.2.

During the optimization, $\mathbf{e_i}$ and $\mathbf{e_j}$ are moving forward to the $\mathbf{e_k}$ because $\mathbf{X}_{i,k}$ and $\mathbf{X}_{j,k}$ are significant, and according to the Equation 3.8, $\mathbf{e_i}^T\mathbf{e_k} \sim \mathbf{X}_{i,k}$ and $\mathbf{e_j}^T\mathbf{e_k} \sim \mathbf{X}_{j,k}$. Consequentially, $\mathbf{e}_i$ and $\mathbf{e}_j$ will locate in the area where is close to $\mathbf{e}_k$. Therefore, this property satisfies the assumption in Subsection 3.1.1 and make entity with the same implicit type locate nearby each other in the feature space.

Note that, the loss function in Equation 3.9 is ill-defined if $\mathbf{X}_{i,j} = 0$. The solution for this issue is to use only non-zero elements in $\mathbf{X}$ to fine-tuning entity embeddings.

The loss function in Equation 3.9 aims to calibrate the entity embeddings, making co-occurred entities close to each other in the feature space. To incorporate type constraints, learning embeddings that represent such constraints is necessary. This aim needs help from bilinear type-agnostic KGC models. The following section includes details about achieving this aim.

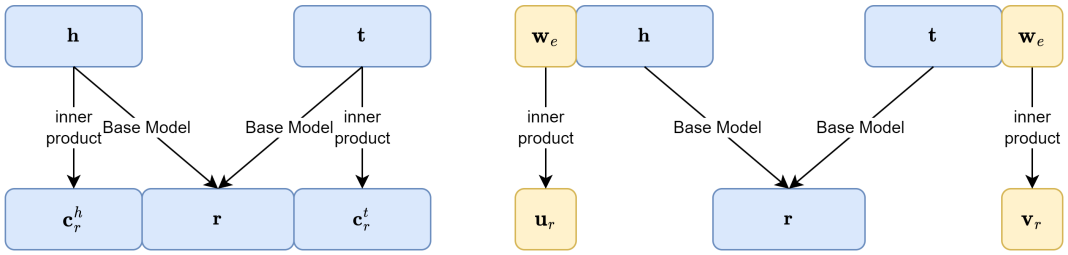
3.2 Incorporate Intra-relation Type Constraint

The previous section introduces the loss function for learning implicit *inter-relation* entity types, whereas this section is about *intra-relation* type constraints, i.e., how to incorporate the entity co-occurrence statistics into relations based on bilinear type-agnostic models. The score functions in the base models are modified to take (implicit) type constraints to evaluate the triple plausibility. A loss function refines the entity embeddings in the base KGC models, and two embeddings are associated with every relation for representing the implicit type constraints.

The extended score function is

$$f(h, r, t) = \sigma(\langle \mathbf{h}, \mathbf{c}_r^h \rangle)\sigma(f_{\text{base}}(h, r, t))\sigma(\langle \mathbf{t}, \mathbf{c}_r^t \rangle), \tag{3.11}$$

where $\sigma(\cdot)$ is the sigmoid function. $\langle \cdot, \cdot \rangle$ is the inner product. $\mathbf{h}, \mathbf{t}$ are entity embeddings for the head or tail argument in the relation r. $\mathbf{c}_r^h$ and $\mathbf{c}_r^t$ are two relation-specific embeddings representing the implicit type constraints in head and tail. For bilinear



(a) The proposed method. (b) The benchmark [1].

Figure 3.2: Comparison between the proposed method and the benchmark.

model that uses complex domain as its feature space, the real and imaginary parts of entity embeddings in $\mathbb{C}^n$ are concatenated to map them into $\mathbb{R}^{2n}$ where n is the dimension of the feature space.

The design of the score function in Equation 3.11 obeys the following criteria:

1. The type compatibility and the triple plausibility should be considered when the model evaluates a triple. This is achieved by combining two terms of type compatibility ($\sigma(\langle \mathbf{h}, \mathbf{c}_r^h \rangle)$, $\sigma(\langle \mathbf{t}, \mathbf{c}_r^t \rangle)$) with the score function inherited from the base model in a logic-and relation: optimally, only triples that are with significant plausibility and best type compatibility in both arguments of the relation should be returned in the top of link prediction results.

2. The type compatibility and the triple plausibility should be balanced when the model evaluates a triple. The sigmoid function, $\sigma(\cdot)$, is responsible for this criteria. First, the sigmoid function assures that all scores, even in the inner product form, have a consistent sign - because its domain is in $(0, 1) \in \mathbb{R}$. Second, during the optimization, the sigmoid function guaranteed that all embeddings in Equation 3.11 are significantly affected by back-propagated gradients only when one of the three terms in the Equation is insignificant.

Figure 3.2 explains the difference between the proposed method and the benchmark [1]. The background colors indicate the feature space of corresponding components. The proposed method has only one feature space. All embeddings in the proposed method are in the feature space used by the base model. The benchmark, as introduced in Section 2.2.2, Equation 2.6, has two different feature space - one feature space

inherited from the base model (blue), and the other one is for evaluating the type compatibility (yellow.) Type compatibility is represented by the inner product of type-related embeddings ($\mathbf{w}_e$, $\mathbf{u}_r$ and $\mathbf{v}_r$.)

The loss function for training entity and relation embeddings from the facts in the knowledge graph is

$$\begin{aligned} P(t|h,r) &= \frac{\exp(\theta f(h,r,t))}{\exp(\theta f(h,r,t)) + \sum_{t'} \exp(\theta f(h,r,t'))}, \\ P(h|r,t) &= \frac{\exp(\theta f(h,r,t))}{\exp(\theta f(h,r,t)) + \sum_{h'} \exp(\theta f(h',r,t))}, \end{aligned} \tag{3.12}$$

$$\mathcal{L}_{\text{KGC}} = -\sum_{(h,r,t)\in\mathcal{G}} \big(P(t|h,r) + P(h|r,t)\big). \tag{3.13}$$

In Equation 3.12, $P(t|h,r)$ is for queries in the form of $(h,r,?)$ and $P(h|r,t)$ is for queries in the form of $(?,r,t)$. (h',r,t) and (h,r,t') are negative examples constituted by corrupting the head entity h or the tail entity t in (h,r,t), respectively. θ is a scalar hyperparameter. The negative sampling method used in Equation 3.12 is uniform sampling as it is for the benchmark [1] and the base models [17, 19].

3.3 Experiment Setting

The proposed method is evaluated on three widely used datasets that are extracted as subsets of existing knowledge graphs. The benchmark is TypeComplEx and TypeDistMult [1]. The base models in the proposed methods are DistMult [17] and ComplEx [19]. The following subsections contain details of the experiment settings.

3.3.1 Dataset

The proposed method is evaluated on three datasets: (1) FB15k-237; (2) WN18RR; and (3) YAGO3-10:

- FB15k-237 is a subset of the knowledge graph Freebase [55]. Most facts in FB15k-237 are about popular cultures, e.g., movies, music, and shows.

Table 3.1: Statistics of datasets.

	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	Train	Valid	Test	Density
FB15k-237	14,541	237	272,115	17,535	20,466	0.0381
WN18RR	40,943	11	86,835	3,034	3,134	0.0053
YAGO3-10	123,182	36	1,079,040	5,000	5,000	0.1610

- WN18RR is a subset of the knowledge graph WordNet [53]. Facts in WN18RR are about relations of English words, such as synonym, hypernym, hyponym.
- YAGO3-10 is a subset of the knowledge graph YAGO [91]. Facts in YAGO3-10 are geographical - the locations where an entity stays or the place in which a person was born.

A statistic, co-occurrence density, has been defined to represent the richness of entity co-occurrences in datasets. Let $|\mathbf{X}|$ be the number of non-zero elements in the matrix $\mathbf{X}$ in Equation 3.1 and $|\mathcal{E}|$ be the entity set cardinality of a knowledge graph. The co-occurrence density is

$$\text{Co-occurrence Density} = \frac{|\mathbf{X}|}{|\mathcal{E}| \times |\mathcal{E}|}. \tag{3.14}$$

The co-occurrence density is in the range of $[0, 1] \in \mathbb{R}$. A large co-occurrence density indicates that entities frequently co-occur across relations. The statistics of these three datasets are summarized in Table 4.10. The density in the table represents the co-occurrence density defined above.

3.3.2 Benchmarks, Hyperparmeters, and Optimization Settings

Base models for testing the proposed method are DistMult and ComplEx. DistMult uses real feature space, whereas ComplEx takes complex domain as its feature space. The DistMult and ComplEx proposed by Jain et al. [1] are benchmarks. The hyperparameter settings follow the instructions in the benchmarks to ensure a fair comparison.

The optimizer used for all models is Adam [92]. All hyperparameters are tuned by grid-search. The embedding dimension is set to 200. α in Equation 3.9 is 0.75. $x_{\max}$ is

250 for FB15k-237 and YAGO3-10, and it is 10 for WN18RR. Other hyperparameters, such as batch size, learning rate, and regularizer weight, are optimized from the following ranges:

1. learning rate and ℓ_2 regularizer weights: $\{1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}\}$;

2. batch size: $\{4096, 8192, 16384\}$;

All models are optimized up to 1,000 epochs with early stopping [93]. All models are optimized by the loss function in Equation 3.13. In addition, the loss function in Equation 3.9 is used periodically - the optimization with Equation 3.9 only happens every K epoch. The grid search range for K is $\{5, 10, 20, 50, 100, 200, 250, 300, 500, 800\}$.

3.3.3 Performance Evaluation Measures

The evaluation metrics are filtered Mean Reciprocal Rank and Hits@K. For facts (h, r, t) in the validation and test sets, one of the entities h or t will be removed, and all entities in the entity set $\mathcal{E}$ are feed into the score function of the model to compute the rank of the removed entity h or t. In addition, all other known correct entities as answers for $(?, r, t)$ or $(h, r, ?)$ are removed before computing the rank. Note that the evaluation criteria follow the setting of the benchmark, and in their setting, they remove other known entities in **training, validation, and test sets.** [3] This is a different setting from what has been used for evaluation which only removes the known correct entities only in the training set for the given query.

Formally, let $\text{rank}(e)$ be the rank of an entity e in the prediction. The MRR is

$$\text{MRR}(e) = \frac{1}{\text{rank}(e)}.$$

If $\text{rank}(e) \leq N$ and e is the correct entity for the query $(h, r, ?)$ or $(?, r, t)$, Hits@N= 1. Otherwise, Hits@N= 0. Hits@N and MRR are computed by all facts in the validation or test sets, and averaged by the number of facts in the corresponding set. The $N = 1, 3$ and 5 are used in Hits@N for evaluation.

[3] cf. code of the official implementation in https://github.com/dair-iitd/KBI.

Table 3.2: Link prediction results. Prefix "Type" is associated to benchmarks. Prefix "Co" is associated to the proposed methods.

		DistMult	ComplEx	TypeDM	TypeCX	CoDM	CoCX
FB15k-23	MRR	0.2888	0.2896	**0.2959**	0.2914	0.2950	0.2904
	Hits@1	0.1995	0.2079	0.2067	0.2086	**0.2091**	0.2063
	Hits@3	0.3194	0.3166	0.3203	0.3188	**0.3262**	0.3188
	Hits@5	0.3818	0.3752	0.3788	0.3748	**0.3823**	0.3779
WN18RR	MRR	0.4257	**0.4683**	0.4286	0.4190	0.4199	0.4079
	Hits@1	0.3963	**0.4107**	0.3908	0.4028	0.3884	0.3878
	Hits@3	0.4375	0.4371	**0.4436**	0.4254	0.4392	0.4147
	Hits@5	0.4566	0.4471	**0.4639**	0.4363	0.4537	0.4290
YAGO3-10	MRR	0.4191	0.4133	0.4376	0.4533	0.4912	**0.4932**
	Hits@1	0.3255	0.3193	0.3405	0.3595	0.4011	**0.4064**
	Hits@3	0.4717	0.4647	0.4920	0.5251	**0.5474**	0.5459
	Hits@5	0.5343	0.5232	0.5496	0.5781	**0.5961**	0.5941

3.4 Experiment Result

Two experiments have been conducted on the three datasets, similar to the approach for evaluating the benchmarks in their paper. The first one is link prediction, and the second one is entity clustering.

3.4.1 Link Prediction

The link prediction results are shown in Table 3.2. CoDM and CoCX are the proposed methods with DistMult or ComplEx as the base model. The prefix "Co" represents "co-occurrence," whereas models with "Type" prefix are the benchmarks with the method proposed by Jain et al. DistMult and ComplEx are two type-agnostic KGC models.

The experiment results show that the proposed methods outperformed the benchmarks on most performance metrics on FB15k-237 and YAGO3-10, because the co-occurrence density in these two datasets is high (cf. Table 4.10.) For the WN18RR, the performance of the proposed method deteriorates. There are two twisted reasons behind the deterioration: (1) the property of WN18RR; (2) the low co-occurrence.

WN18RR is a subset of the knowledge graph, WordNet. Facts in it are about relations between English words. For example, "synonym," "hypernym," "hyponym," and so on. These relations can take a wide range of colossally different words in their arguments (head or tail.) Take words "mammal" and "vehicle" as examples. These two words are hypernym of "human" and "bus." Therefore, "human" is a proper entity for the tail location of "hypernym" if the head entity is "mammal," and "bus" fits the tail location of the same relation if the head entity is "vehicle." Consequentially, this general acceptance of entities in both relations results in obstacles when defining (artificially or not) type constraints for the relation "hypernym," and the same phenomenon happens for other relations in WN18RR that describe relationships between words.

Another reason that restricts the proposed method to reinforce the link prediction performance is the low entity co-occurrences as shown in the Table 4.10. This is a by-product of the previous reason because relations in WN188RR take a wide range of words, and their co-occurrences are low. These two reasons make the gradients from Equation 3.9 and 3.13 help little to calibrate embeddings.

The co-occurrences are critical in the proposed method - the improvements of base models on YAGO3-10 are more significant than these on FB15k-237, and for the low co-occurrence dataset WN18RR, the proposed method deteriorates the base model. The density of FB15k-237, WN18RR, and YAGO3-10 are 0.0381, 0.0053, and 0.1610, as shown in Table 4.10. Because the loss function (Equation 3.9) for calibrating entity embeddings and learning embeddings representing type constraints depends on the co-occurrence, therefore, the effectiveness of this loss function depends on the co-occurrence density.

Table 3.2 shows that the improvements of DistMult are more than those for ComplEx because the feature space in ComplEx is in $\mathbb{C}^n$ where n is the dimension. The number of parameters in the ComplEx is twice as it is in DisMult. The proposed method that uses ComplEx as its base model requires more training examples.

3.4.2 Entity Embedding Clustering

The ability to capture entity types by co-occurrence in the proposed method is examed by all entities that appear in the relation "influence-by" in the FB15k237. The reason for choosing this relation is that this relation contains a diverse entity with different types in the head and tail location, and the number of entities in different arguments is large.

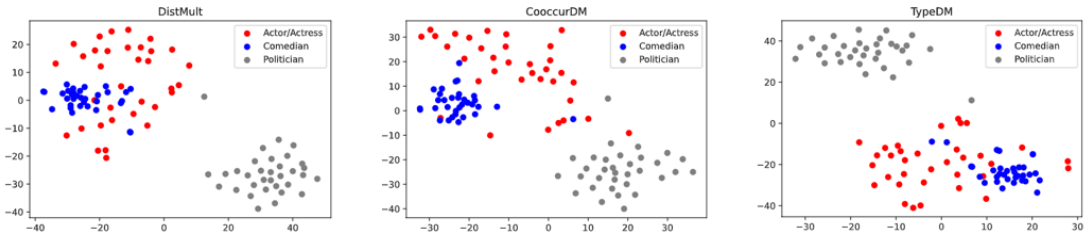


Figure 3.3: Entity clustering results of DistMult, CooccurDM, and TypeDM.

Entity types are annotated by the same entity matched by its Freebase identifier to the corresponding entities in the WikiData [4], and the descriptions in Wikidata for entities are taken as their types. The first concept is taken as the type for entities with more than one type, e.g., "Jodie Foster" has a description of "American actor, film director and producer" by the time of 5th September, 2022. In this case, the type of "Jodie Foster" would be "Actor/Actress."

Figure 3.3 shows the results of clustering entity embeddings by tSNE [94] where entity embeddings are from DistMult, TypeDM (the benchmark), and CooccurDM (the proposed method.) As the figure shows, all models can distinguish "Actor/Actress" and "Comedian" from "Politician", but DistMult cannot distinguish "Comedian" from "Actor/Actress." whereas TypeDM and the proposed method, CooccurDM, can distinguish "Comedian" from "Actor/Actress." But the clusters from entity embeddings in TypeDM have more overlap compared to those from CooccurDM.

Note that the type annotations are not used in training, and neither the benchmark nor the proposed method has the ability to utilize this information, and the differences between types "Actor/Actress" and "Comedian" are vague because some celebrities are with both types, and it is hard to distinguish which one is their primary type - an actress can be a comedian and vice versa. The clustering results indicate that the proposed method can alleviate the vagueness by utilizing entity co-occurrences, as the clusters constituted by embeddings from the proposed method have less overlap.

In addition to testing the ability to distinguish different types by learned entity embeddings, the type constraint embeddings in "influence-by" are clustered together with the corresponding entity embeddings to test the ability to capture relation-specific type constraints. The clustering results are shown in Figure 3.4. The head type

[4]https://www.wikidata.org/

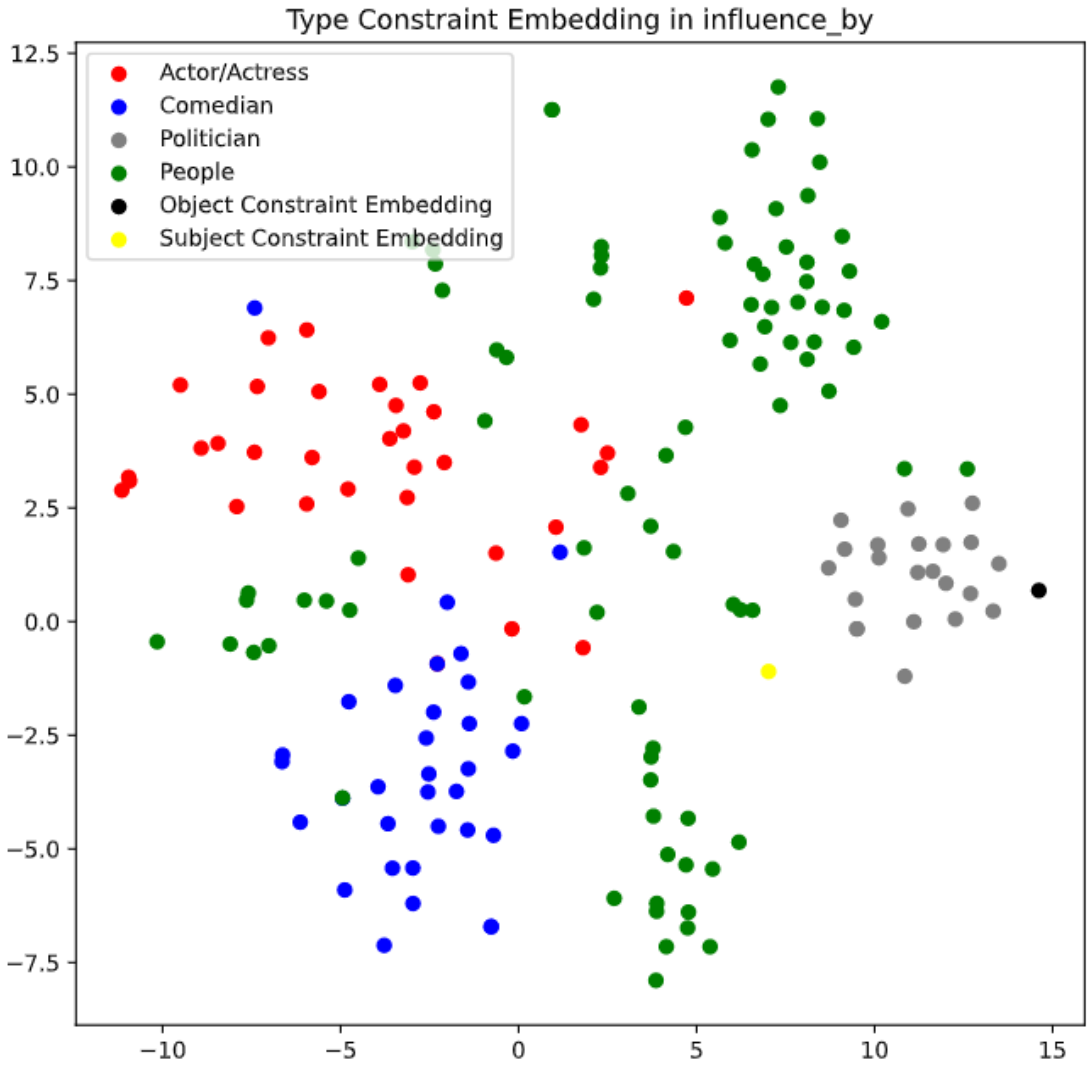


Figure 3.4: Entity clustering results of all entities appeared in "influence-by."

constraint embedding ($\mathbf{c}_r^h$) and the tail type constraint embedding ($_r^t$) are annotated as "subject constraint embedding" and "object constraint embedding" in the figure, respectively.

Note that all entities in the Figure 3.4 belong to the type "People." "Comedian," "Politician," and "Actor/Actress" are three sub-types of the type "People." The clustering results show that the proposed method can distinguish entities with different types, and can capture the relation specific type constraint because the embeddings that represent type constraint on different arguments are close to the entity embeddings

taken by the relation.

3.5 Discussion and Conclusion

This chapter introduced the unsupervised type representation learning method for bilinear KGC models. The proposed method uses entity co-occurrences to infer the implicit entity types in an unsupervised approach. The entity co-occurrences are defined for entities across all relations in the knowledge graph, and a loss function for calibrating entity embeddings is used periodically during the optimization to adjust entity embeddings in the feature space. In the proposed method, the locations of entities in the feature space represent the entity type.

Different from other methods which separate type representations and entity and relation representations to different feature space, the proposed method unifies the feature space. The unification of feature space allows the proposed method to balance the type compatibility and the triple plausibility for answering queries in the link prediction.

Experiments on three widely used datasets show that the proposed method can improve the underlying type-agnostic KGC models' link prediction performance, and the entity clustering results show that the learned entity embeddings can capture the entity type.

However, there are several shortcomings in the proposed method. The first one is that one relation often accepts multiple entity types for the head and the tail arguments, as mentioned in Section 3.4.2. The proposed method only uses two relation-specific embeddings on the arguments to capture such constraints. As a result, the proposed method fails to capture the diverse type constraints in relations.

The second shortcoming is that the current method is only for the bilinear KGC models. The applicable range is not sufficiently general. It would be optimal to extend the similar method to both bilinear and translational models because some KGC models are used for warming-up entity and relation embeddings that are used in other tasks [30, 8]. It is preferable that the proposed method can extend to a general categories of KGC models so that it can be used as a tool for these tasks as well.

The third shortcoming is from the statistic, entity co-occurrences, used in the proposed method. The entity co-occurrences take $O(|\mathcal{E}|^2)$ space and have $O(|\mathcal{G}|)$ time

complexity. For large datasets such as YAGO3-10, it is time-consuming to get entity co-occurrences.

Nevertheless, the research in this chapter plays the role of a pioneer study on unsupervised type representation learning for knowledge graph completion models. The next chapter introduces an improved, generalized unsupervised type representation learning method for both bilinear and translational models. The method introduced in the next chapter is inspired by the research in this chapter. Unlike the research in this chapter, the prerequisites are formalized in the following research rather than an intuition in this chapter.

4

Enhancing Knowledge Graph Completion Models with Unsupervised Type Representation Learning

The research in this chapter is inspired by the research in Chapter 3. It extends the proposed method in Chapter 3 to both bilinear and translational KGC models. The intuitive assumption in Chapter 3 is formalized by describing it mathematically. This research extends the number of relation-specific embeddings representing type constraints to capture the various type constraints. The loss functions for calibrating entity embeddings and type constraint embeddings are modified accordingly to drop the co-occurrence statistics. Extensive experiments have been conducted with the same datasets used in Chapter 3 to examine the proposed method in this chapter on link prediction and entity clustering. Experiment results show that the proposed method outperforms the Type-series approach [1] designed for bilinear models and AutoEter [29], which uses a mixture of KGC models. In addition, the proposed method

can adapt to the KGC model with a complex feature space (i.e., RotatE [16].) The research in this chapter is also in an unsupervised manner. The research in this chapter has been published as a journal paper in Information [95].

4.1 Embeddings for Capturing Implicit Type and Type Constraints

It has been mentioned in Section 3.5 that a relation can have various type constraints on head and tail. The proposed method in this chapter utilizes relation-specific multiple *prototype embeddings* to represent such various constraints. Therefore, it is named **ProtoE** in the following contents.

4.1.1 Formalizing Implicit Type and Type Constraints in relations

Let

$$H_r = \{h_1, h_2, ..., h_n\}$$

and

$$T_r = \{t_1, t_2, ..., t_m\}$$

be two sets of head and tail entities of the relation r, respectively. Let F_r be the set of facts that involve r:

$$F_r = \{(h_1, r, t_1), ..., (h_n, r, t_m)\}.$$

$h \in H_r$ and $t \in T_r$ must satisfy (implicit) type constraints in r. Therefore, the head and tail entities with respect to a relation r imply the type constraints in r. Because all $h \in H_r$ and $t \in T_r$ satisfy the type constraints in r, H_r and T_r can be divided into multiple subsets as follows:

$$H_r = H_r^1 \cup H_r^2 \cup ... \cup H_r^i,$$

$$T_r = T_r^1 \cup T_r^2 \cup ... \cup T_r^j.$$

Entities in H_r^x and T_r^y have the same (unknown) type x or y.

The aim of ProtoE is to capture the different, implicit entity types in H_r^x and T_r^y by relation-specific embeddings.

4.1.2 Unification of Bilinear and Translational KGC Models

The proposed method, ProtoE, unifies bilinear and translational KGC models by the score function. Let $f(h, r, t)$ be a score function in the bilinear KGC models, and let $\mathbf{h}, \mathbf{t}$ be embeddings of the entities h and t, respectively. As introduced in Section 2.1, the relation embeddings $\mathbf{r}$ define the bilinear form used as the score function. Therefore, the $f(h, r, t)$ can be re-written as $f(h, t, ; r)$ to address that the bilinear form in $f(h, r, t)$ is defined by the relation embedding of r.

Suppose there are two triples, (h, r, t) and (h', r, t). (h, r, t) is a fact, whereas (h', r, t) is a false claim. Now, consider the margin of scores from the score function $f(h, t; r)$ for these two triples, $\phi(h, h'; (\cdot, r, t))$. From the property of bilinear form, it is clear that

$$\phi(h, h'; (\cdot, r, t)) = f(h, t; r) - f(h', t; r) = \langle \mathbf{h} - \mathbf{h}', \mathbf{t} \rangle_r = f(h - h', t; r). \tag{4.1}$$

In Equation 4.1, $(\cdot, r, t)$ in ϕ addresses that the margin is about triples related to relation r and its tail entity t. Because the task of the score function is to distinguish actual triples from false triples, the margin

$$f(h - h', t, r) \neq 0.$$

Consequentially, by the property of the bilinear form, it is clear that

$$\Delta\mathbf{h} = \mathbf{h} - \mathbf{h}' \neq \mathbf{0},$$

where $\mathbf{0}$ represents the all-zero vector. Following the same procedure, the scenario where the query is $(h, r, \cdot)$ has a similar score margin property:

$$\Delta\mathbf{t} = \mathbf{t} - \mathbf{t}' \neq \mathbf{0}.$$

t is an entity that makes (h, r, t) a true triple, whereas t' makes (h, r, t') a false triple.

From the derivations above, it can be concluded that entities qualified and disquali-

fied for filling the query naturally have a trend of constituting clusters in the feature space of bilinear KGC models.

Now, consider the case of translational models. Let $f(h, r, t)$ be the score function of a translational model and $\mathbf{h}, \mathbf{t}, \mathbf{r}$ be entity and relation embeddings. Because some translational models (e.g., TransH [14], TransD [13], TransR [15]) take projection to map entity embeddings, the $\mathbf{h}$ and $\mathbf{t}$ represent entity embeddings that after the projection. For TransE, $\mathbf{h}$ and $\mathbf{t}$ are the vanilla entity embeddings because TransE does not use projection. The score margin in translational models has following properties:

$$\begin{aligned} \phi(h, h'; (\cdot, r, t)) &\propto \|\mathbf{h}\|^2 - \|\mathbf{h}'\|^2 + \langle \mathbf{h}' - \mathbf{h}, \mathbf{t} \rangle; \\ \phi(t, t'; (h, r, \cdot)) &\propto \|\mathbf{t}\|^2 - \|\mathbf{t}'\|^2 + \langle \mathbf{h}, \mathbf{t}' - \mathbf{t} \rangle; \end{aligned} \tag{4.2}$$

where (h, r, t) is an actual triple and (h', r, t) and (h, r, t') are false triples. Equation 4.2 shows that the score margin in translational KGC models is proportional to $\mathbf{h}' - \mathbf{h}$ and $\mathbf{t}' - \mathbf{t}$, respectively. Therefore, in both bilinear and translational KGC models, entity embeddings tend to constitute clusters in the feature space according to whether the entity qualifies the query $((?, r, t)$ or $(h, r, ?))$ or not.

4.1.3 Prototype Embeddings for Capturing Implicit Type Constraints

Section 4.1.1 and Section 3.4.2 show that:

1. Implicit type constraints in relations can be represented as multiple subsets that contain the same type of entities (Section 4.1.1.)

2. Entity embeddings in bilinear and translational KGC models tend to constitute clusters in the feature space according to their occurrences in triples (Section 4.1.2.)

Therefore, the type constraints in relations can be captured by relation-specific multiple embeddings. In this research, they are prototype embeddings. Each prototype embedding represents a subset of entities that a relation can accept. The prototype

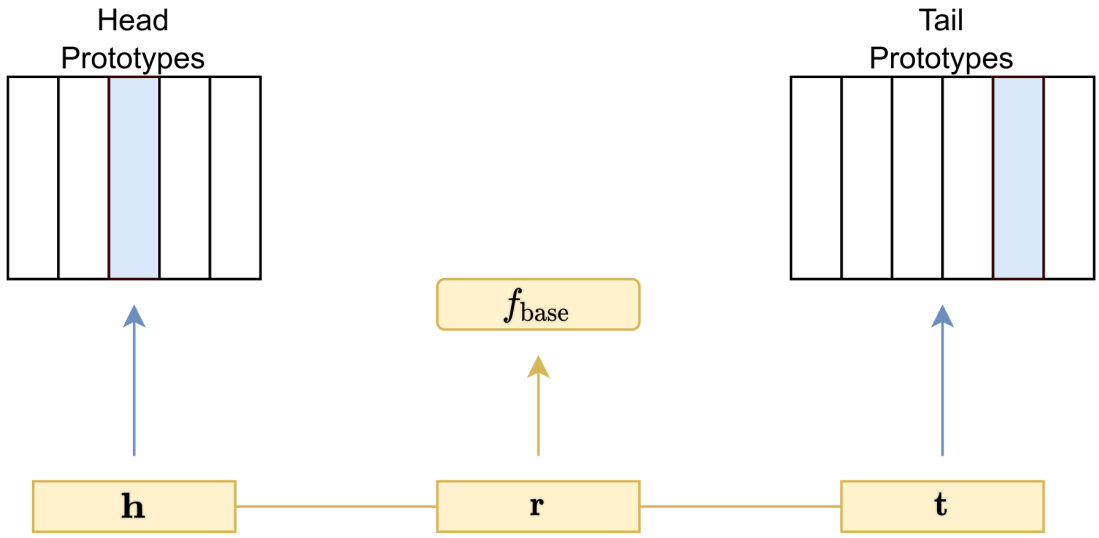


Figure 4.1: Structure of proposed method, ProtoE. Every relation has associated prototype embeddings representing type constraints on head and tail.

embeddings are used in the training stage to calibrate entity embeddings, and as the previous research in Chapter 3, entity embeddings use their locations in the feature space to represent the implicit entity type. The name of the proposed method, ProtoE, comes from this setting.

Figure 4.1 shows the structure of ProtoE. Every relation has prototype embeddings that represent type constraints on the head and tail. Note that the proposed method supports the setting of different numbers of prototype embeddings on different arguments (i.e., head or tail) in relations. Entity embeddings of h or t in triples related to r will be associated with the prototype embeddings $\mathbf{p}_r^h$ or $\mathbf{p}_r^t$ with the maximal similarity. The automatically assigned prototype embeddings (the blue column vectors) and $\mathbf{h}, \mathbf{t}$ (i.e., the entity embeddings of h and t) are used to evaluate the plausibility of the triple (h, r, t).

From the perspective of representation learning, the prototype embeddings represent local areas in the feature space where entity embeddings are more likely to have a high score from the inherited score function in the base model for the relation r. Formally, let $\mathbf{P}_r^h \in \mathbb{R}^{m \times d}$ and $\mathbf{P}_r^t \in \mathbb{R}^{n \times d}$ be two matrices in which column vectors are prototype embeddings for h or t in r, respectively. m and n are the numbers of prototype embeddings associated to head or tail, respectively. d is the dimension of

entity embeddings. The compatibility of entities to r is evaluated as follows.

$$g_{\text{head}}(\mathbf{h}, \mathbf{P}_r^h) = \max \text{softmax}(\mathbf{h}, \mathbf{P}_r^h)$$
$$g_{\text{tail}}(\mathbf{t}, \mathbf{P}_r^t) = \max \text{softmax}(\mathbf{t}, \mathbf{P}_r^t). \tag{4.3}$$

In Equation 4.3, $\mathbf{h}$ and $\mathbf{t}$ are entity embeddings learned by the base model (e.g., TransE [12], TransH [14], DistMult [17], ComplEx [19].) For entity embeddings in the complex domain, the real and imaginary parts are concatenated in the real domain. That is, $\mathbf{h} = [\text{Re}(\mathbf{h}_{\text{complex}}); \text{Im}(\mathbf{h}_{\text{complex}})]$. For some translational models with projection matrices or vectors, $\mathbf{h}$ and $\mathbf{t}$ are the ones after projection. The max softmax function in Equation 4.3 is

$$\max \text{softmax}(\mathbf{e}, \mathbf{P}) = \max \left(\frac{\exp(\langle \mathbf{e}, \mathbf{p}_1 \rangle)}{\sum_{i=1}^{N} \exp(\langle \mathbf{e}, \mathbf{p}_i \rangle)}, \frac{\exp(\langle \mathbf{e}, \mathbf{p}_2 \rangle)}{\sum_{i=1}^{N} \exp(\langle \mathbf{e}, \mathbf{p}_i \rangle)}, ..., \frac{\exp(\langle \mathbf{e}, \mathbf{p}_N \rangle)}{\sum_{i=1}^{N} \exp(\langle \mathbf{e}, \mathbf{p}_i \rangle)} \right),$$

where $\mathbf{e}$ is the entity embedding, and $\mathbf{P}$ represents the matrix that stores prototype embeddings as its column vectors. $\mathbf{p}_i$ is the i-th prototype embedding (i.e., the i-th column vector in $\mathbf{P}$.) N is the number of prototype embeddings.

The crucial components in Equation 4.3 are the inner product between the entity embeddings and the prototype embeddings, i.e., $\langle \mathbf{e}, \mathbf{p}_1 \rangle$ for $i = 1, ..., N$. The intention of Equation 4.3 is from the squared distance:

$$\|\mathbf{e} - \mathbf{p}_i\|^2 = \|\mathbf{e}\|^2 - 2\langle \mathbf{e}, \mathbf{p}_i \rangle + \|\mathbf{p}_i\|^2.$$

Because the squared distance $\|\mathbf{e} - \mathbf{p}\|^2$ is disproportional to the inner product $\langle \mathbf{e}, \mathbf{p}_i \rangle$, it is possible to use the inner product as the type compatibility. As discussed in Section 4.1.2 and the summarized conclusions at the beginning of this subsection, their embeddings for entities in a relation tend to constitute clusters in the feature space. Combining of the max and softmax functions aims to associate entity embeddings to a prototype embedding. In this sense, the associated prototype embedding is the representative embedding for the cluster.

From the previous discussion in Section 4.1.1, it is known that entities that appeared together in many relations are more probable to have the same type because they

satisfy type constraints in multiple relations. Therefore, a critical criterion in ProtoE is to capture type constraints with prototype embeddings by this property. In addition, the discussions in this subsection and Section 4.1.2 guarantee that ProtoE can adopt bilinear and translational KGC models as its base models. The score function and loss functions in the following sections are designed for the criteria and utilize the property for bilinear and translational models.

4.2 Score Function in ProtoE

The score function in ProtoE is in the following form.

$$f_{\text{ProtoE}} = \sigma(\alpha \times g_{\text{head}}(\mathbf{h}, \mathbf{P}_h^r)) \circ f_{\text{base}}(h, r, t) \circ \sigma(\alpha \times g_{\text{tail}}(\mathbf{t}, \mathbf{P}_t^r)). \tag{4.4}$$

where σ is the sigmoid function. $\circ$ represents one of two algebra operations, namely addition ("+") or multiplication ("×"). $\alpha \in \mathbb{R}$ is a scaling hyperparameter. Different algebra operations represent different strategies for balancing the type compatibility (measured by the function g) and triple plausibility (measured by the function f.)

The following subsections discuss the different aspects that affect the link prediction performance, including the impacts of different strategies and the setting of the number of prototype embeddings, and how they would affect the results in link prediction.

4.2.1 Strategies and Design of the Score Function in ProtoE

If the binary algebra operator "$\circ$" is addition "+", the strategy of evaluating triples follows "OR" logic. The ranking of candidates in the prediction result depends on the overall score in Equation 4.4. Because the compatibility for the head and tail locations are in the sigmoid function, therefore the scale of these components is in $(0, 1)$. Thus, entities in the prediction with the logical "OR" strategy would not be the entities that are apparently not appropriate (the score from f_{base} is low) but only satisfy the type constraints (the scores from g are high.)

On the contrary, the multiplication ("×") follows "AND" logic. For all entities in the prediction results, they must have high scores from all three components. Because of the sigmoid functions on type compatibility, the interdependence plausibility represented by f will be scaled proportionally by the type compatibility.

The sigmoid function σ and the hyperparameter α are used to smooth the scores of entity compatibility (represented by g_{head} and g_{tail}.) The sigmoid function uniforms the range of entity type compatibility in $(0, 1)$, and the α and the sigmoid function jointly uniform the score sign represented by f_{ProtoE} unchanged as in the following discussions.

The score function f_{base} from the base model in the extended score function (Equation 4.4) should always represent the plausibility of triples. Bilinear models satisfy this requirement, but it may be different for translational models because some translational models use distance to represent the triple implausibility. Their score function is multiplied by -1 to enable ProtoE to adapt these models. Take TransR as an example. ProtoE adapts the score function in TransR as f_{base} in Equation 4.4 as follows:

$$f_{\text{base}}(h, r, t) = -f_{\text{TransR}}(h, r, t) = -\|\mathbf{M}_r\mathbf{h} + \mathbf{r} - \mathbf{M}_r\mathbf{t}\|^2. \tag{4.5}$$

The purpose of modifying f_{TransR} is to keep f_{base} and g consistent. First, consider the case of queries in $(?, r, t)$ form. In this case, because the correct tail entity is given, the given entity must satisfy the type constraints in the tail of r, therefore

$$\sigma(\alpha \times g_{\text{tail}}(\mathbf{t}, \mathbf{P}_t^p)) \approx 1.$$

The prediction depends on $\sigma(\alpha \times g_{\text{head}}(\mathbf{h}, \mathbf{P}_h^p))$ and f_{base} in Equation 4.4. Only entities that satisfy the type constraints in the head location of r (with a high score from $\sigma(\alpha \times g_{\text{head}}(\mathbf{h}, \mathbf{P}_h^p))$) and make the triple plausibility (with a high score from f_{base}) will take the top locations in the prediction. Similarly, for queries $(h, r, ?)$, the prediction depends on $\sigma(\alpha \times g_{\text{tail}}(\mathbf{t}, \mathbf{P}_t^p))$ and f_{base}. Note that in both cases, the hyperparameter α and f_{base} are critical for the prediction.

Now, consider the modified score function (in Equation 4.5) inherited from the translational models that use distance to measure triple implausibility. The change of sign ensures that the monotonicity of f_{base} is consistent: plausible triples have high scores (i.e., $|f_{\text{base}}|$ is low as $f_{\text{base}} < 0$.) For the same reason, the smooth hyperparameter α is in $\mathbb{R}^-$ for translational models that use distance as triple implausibility. The negative α ensures that if a triple (h, r, t) is true, the score f_{base} will be scaled with smaller $|f_{\text{base}}(h, r, t)$ by other two components with g_{head} and g_{tail} for measuring entity type compatibility.

4.2.2 Effects of the Number of Prototype Embeddings

The number of prototype embeddings affects ProtoE's performance in link prediction. Because the exact numbers of acceptable types in relations are unknown, it is difficult to set the explicit, accurate m and n for the number of prototype embeddings in the head and tail of a relation r, respectively.

If m and n are too large, the prototype embeddings may split the entity embeddings that should be within the same cluster to multiple clusters. If these multiple clusters are not nearby in the feature space, the link prediction performance will deteriorate. Another possibility is that the redundant prototype embeddings are not assigned by any entity embedding. As a result, some of these prototype embeddings are not optimized after initialization. In the latter case, the performance is rarely affected because these orphan prototype embeddings are not affecting entity embeddings.

In contrast, if the m and n are too small, the entity type compatibility functions g_{head} and g_{tail} fail to capture the type constraints and therefore become incapable of help adjusting entity embeddings. In this case, the link prediction performance will deteriorate either.

The score function f_{ProtoE} in Equation 4.4 are used in two loss functions to learn and calibrate all embeddings in ProtoE. The tasks and details of the loss functions are introduced in the following section.

4.3 Loss Functions in ProtoE

The loss functions in ProtoE are similar to the method introduced in Chapter 3. One loss function aims to optimize entity, relation, and prototype embeddings, and another one calibrates the location of these embeddings.

4.3.1 The Loss Function for Learning Embeddings

The loss function for learning entity, relation, and prototype embeddings is similar to the one in Chapter 3. Equation 4.6 is the loss function.

$$
\begin{aligned}
P(h|r,t) &= \frac{\exp(\theta f_{\text{ProtoE}}(h,r,t))}{\exp(\theta f_{\text{ProtoE}}(h,r,t)) + \sum_{(h',r,t)\in\mathcal{D}_{\text{train}}} \exp(\theta f_{\text{ProtoE}}(h',r,t))}, \\
P(t|h,r) &= \frac{\exp(\theta f_{\text{ProtoE}}(h,r,t))}{\exp(\theta f_{\text{ProtoE}}(h,r,t)) + \sum_{(h,r,t')\in\mathcal{D}_{\text{train}}} \exp(\theta f_{\text{ProtoE}}(h,r,t'))}, \\
\mathcal{L}_{\text{KGC}} &= -\sum_{(h,r,t)\in\mathcal{D}_{\text{train}}} \log\big(P(h|r,t) + P(t|h,r)\big).
\end{aligned} \tag{4.6}
$$

where $\mathcal{D}_{\text{train}}$ is the training data. $\theta \in \mathbb{R}^+$ is a hyperparameter. (h', r, t) and (h, r, t') are corrupted triples by replacing h or t in (h, r, t) to a entity h' or t' by uniform negative sampling, respectively. The score function f_{ProtoE} is in Equation 4.4.

Entity and relation embeddings are learned by $\mathcal{L}_{\text{KGC}}$ in Equation 4.6 with f_{base} in f_{ProtoE}. The prototype embeddings appear in the loss function, but it is insufficient to barely use this loss function to optimize them because of the sigmoid functions. The derivative of the sigmoid function $\sigma(\cdot)$ in Equation 4.4 is

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

If $\alpha \times g_{\text{head}}(\cdot)$ or $\alpha \times g_{\text{tail}}(\cdot)$ are too high or too low, the gradients for the wrapped elements with the sigmoid function are in low-magnitude. As a result, the prototype embedding locations change little during the back-propagation. Note that the score function f_{base} in f_{ProtoE} is without the sigmoid function, therefore, the vanishing gradient issue does not affect the optimization of entity and relation embeddings (used in f_{base}.)

4.3.2 The Loss Function for Calibrating Embedding Locations

Due to the gradient issue caused by the sigmoid functions in 4.6, ProtoE induces another loss function to help calibrate embedding locations, mainly for the prototype embeddings. The induced loss function is in the following form.

$$\mathcal{L}_{\text{prototype}} = \sum_{r \in \mathcal{R}} \sum_{(h',r,t),(h,r,t) \in \mathcal{D}_{\text{train}}} \max\left(0, g_{\text{head}}(\mathbf{h}', \mathbf{P}_r^h) - g_{\text{head}}(\mathbf{h}, \mathbf{P}_r^h) + 1\right) + \sum_{r \in \mathcal{R}} \sum_{(h,r,t'),(h,r,t) \in \mathcal{D}_{\text{train}}} \max\left(0, g_{\text{tail}}(\mathbf{t}', \mathbf{P}_r^t) - g_{\text{tail}}(\mathbf{t}, \mathbf{P}_r^t) + 1\right), \tag{4.7}$$

where $\mathcal{D}_{\text{train}}$ is the training set and (h', r, t) and (h, r, t') are corrupted triples. The loss function $\mathcal{L}_{\text{prototype}}$ is constituted by hinge loss to separate entity embeddings whose corresponding entities do not satisfy the implicit type constraints in r.

For entities that satisfy the type constraints, the $g_{\text{head}}(\mathbf{h}, \mathbf{P}_r^h)$ and $g_{\text{tail}}(\mathbf{t}', \mathbf{P}_r^t)$ wrapped by the sigmoid function are close to 1, but for those that do not satisfy the constraints, the sigmoid wrapped $g_{\text{head}}(\mathbf{h}', \mathbf{P}_r^h)$ and $g_{\text{tail}}(\mathbf{t}', \mathbf{P}_r^t)$ are not necessarily close to 0. Note that the g_{head} and g_{tail} are in $(0, 1)$ without the sigmoid (cf. Equation 4.4), and the hinge loss is with margin 1. In this manner, the $\mathcal{L}_{\text{prototype}}$ helps calibrate the locations of the prototype and entity embeddings so that they can be used with the hyperparameter α in the sigmoid function to evaluate type compatibility. Note that the sigmoid function does not appear in Equation 4.7 to prevent diminishing gradients.

4.3.3 The Role and Purpose of Loss Functions in ProtoE

$\mathcal{L}_{\text{KGC}}$ and $\mathcal{L}_{\text{prototype}}$ learn embeddings in the ProtoE, but the purposes and approaches of these two loss functions are slightly different. $\mathcal{L}_{\text{KGC}}$ does not distinguish triples in a relation-specific manner, but $\mathcal{L}_{\text{prototype}}$ iterates over all relations in the training data. $\mathcal{L}_{\text{KGC}}$ learns embeddings for distinguishing correct triples, i.e., facts, (h, r, t) and corrupted triples (h', r, t) and (h, r, t'). In contrast, $\mathcal{L}_{\text{prototype}}$ calibrates the relation-specific prototype embeddings and acceptable and unacceptable entity embeddings for every relation. There are cases where entities in the corrupted triples satisfy the implicit type constraints, but the entities themselves are not proper, e.g., (Tokyo, is-located-in, the U.K.) If the losses in Equation 4.6 and Equation 4.7 are used in the approach of $\mathcal{L} = \mathcal{L}_{\text{KGC}} + \mathcal{L}_{\text{prototype}}$, the model may over-fit and fail to distinguish corrupted triples from facts because too many entity embeddings are pushed too close to the prototype embeddings, resulting in inappropriate entity embedding locations for the base model f_{base}.

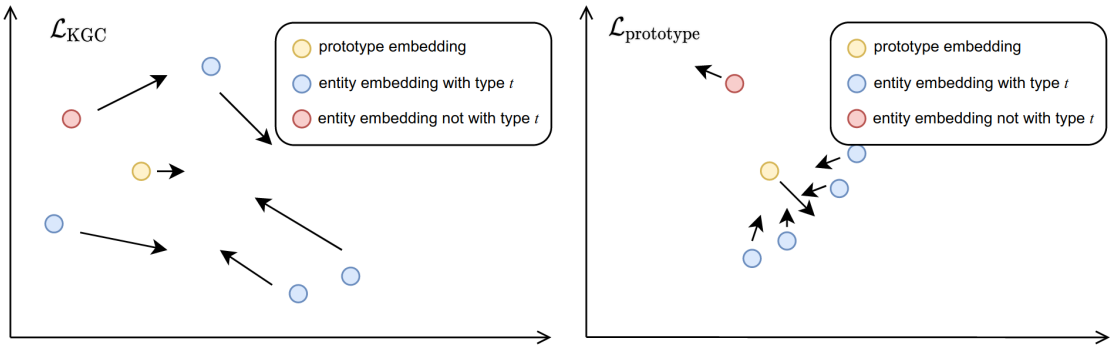


Figure 4.2: Example of effects of $\mathcal{L}_{\text{KGC}}$ and $\mathcal{L}_{\text{prototype}}$ on embedding locations.

Therefore, these two loss functions are used reciprocally with different learning rates to prevent over-fitting. $\mathcal{L}_{\text{KGC}}$ optimizes all embeddings in every epoch, whereas $\mathcal{L}_{\text{prototype}}$ is used only every T epoches. Let β_{KGC} and $\beta_{\text{prototype}}$ denote the learning rate for $\mathcal{L}_{\text{KGC}}$ and $\mathcal{L}_{\text{prototype}}$, respectively. During the optimization, $\beta_{\text{KGC}} > \beta_{\text{prototype}}$.

Figure 4.2 illustrates the effects of these two loss functions. The effects of these two loss functions, $\mathcal{L}_{\text{KGC}}$ and $\mathcal{L}_{\text{prototype}}$, on calibrating embedding locations are on the left and right sides, respectively. The yellow point represents a prototype embedding in a relation r. Blue points are entity embeddings associated with the prototype embedding, i.e., the entity type is represented by the prototype embedding. The red point is an entity embedding that does not satisfy the type constraints in r, i.e., the corresponding entity's type should not be represented by the prototype embedding. The arrows in the Figure 4.2 represent the optimization trend of these embeddings, i.e., the movement in the feature space. Arrows in the left figure are movements accumulated in T epochs by $\mathcal{L}_{\text{KGC}}$, whereas arrows in the right figure represent the T-th epoch by $\mathcal{L}_{\text{prototype}}$. Note that in this illustration example, because of the sigmoid function in Equation 4.4, the movement of the prototype embedding is not as significant as that for entity embeddings in the left figure, and in the right figure, because of the relatively low learning rate, the overall movements are not as significiant as those in the left figure, but the directions are more toward the location where the prototype embedding is located.

Table 4.1: Statistical information of datasets in the experiments. These statistics are introduced in Table 4.10 as well.

Name	Entity	Relation	Train	Valid	Test
FB15k-237	14,541	237	272,115	17,535	20,466
WN18RR	40,943	11	86,835	3,034	3,134
YAGO3-10	123,182	37	1,079,040	5,000	5,000

4.4 Experiment Setting

ProtoE is evaluated on three datasets. These three datasets are used for evaluating the method proposed in Chapter 3 either, namely, FB15k-237, WN18RR, and YAGO3-10. These datasets have been introduced in Section 3.3.1. Therefore, only the statistics (also provided in Section 3.3.1, Table 4.10) are listed in Table 4.1 on account of fast referring for readers.

4.4.1 Baselines and Base Models

The following unsupervised type representation learning approaches are chosen as baselines.

1. TypeDistMult and TypeComplEx [1]. These two methods are baselines in the method proposed in Chapter 3 either. The relation-specific type constraint embeddings and embeddings for implicit entity types are independent of the entity and relation embeddings. Each relation has only one embedding representing the head and tail's implicit, acceptable entity types.

2. AutoEter [29] is a method that takes RotatE [16] as the base model. For capturing implicit entity type and type constraints, it integrates TransE [12], TransR[15] and TransH [14]. The score function for evaluating triple plausibility is from RotatE, and the method measures the type compatibility mixtures TransE, TransR, and TransH.

As mentioned in Section 2.2.2, the unsupervised methods take type-agnostic KGC models as their base models. The TransE [12], TransR [15], RotatE [16], DistMult [17], and ComplEx [19] are chosen as the base models for the proposed and baseline

methods. DistMult and ComplEx are used to test the compatibility of bilinear models, and the other models are used to test the compatibility of translational models with and without entity embedding projection.

4.4.2 Hyperparameter Settings, Base Models and Baselines

The hyperparameter settings for DistMult, ComplEx, and TypeDistMult and TypeComplEx follow those in a previous study [1]. The same method is also applied to TransE by changing f_{base} in Equation 2.6 with f_{TransE} in Equation 2.4. However, the method cannot be applied to TransR due to the numeric issue, because the norm constraints in TransR result in float number underflow in the sigmoid function on f_{base} in Equation 2.6.

The training for TransR followed the instructions in the companion paper [15], with the embedding size changed to 100 for both entities and relations because a large embedding size improves the performance of TransR. The TransR and ProtoETransR were tested on FB15k-237 and WN18RR only because the number of entities in YAGO-3 is much larger than those in the other two datasets (cf. Table 4.1.) As a result, the projection matrix in TransR consumes a lot of memory and causes difficulties in optimization (about 33 hours to train for 1,000 epochs on YAGO3-10 with Nvidia Titan X, and 25 hours with an Nvidia A6000.) For the same reason, RotatE, AutoEter, and ProtoERotatE were not tested on YAGO3-10 due to the long training time of AutoEter.

The RotatE and ProtoERotatE followed the addition strategy in Equation 4.4, and all other models followed the multiplication strategy. Because the negative sampling method for corrupted triples affects performance [16, 96], all models were trained with uniform negative sampling for a fair comparison because it is used in all base models except for RotatE. The number of negative examples was set to 20.

All parameters were randomly initialized by the Glorot uniform initializer [97]. The loss function used in training all models was the softmax loss in Equation 4.6 with $\theta = 20$. The hyperparameters are optimized by the grid search. The optimizer is Adam [92]. The range of gird search is summarized as follows.

1. Max epoch: 1000;

2. Learning rate β_{KGC}: $\{0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001\}$;

3. Prototype loss learning rate $\beta_{\text{prototype}}$: $\{0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001\}$;
4. ℓ_2 regularizer weight β: $\{0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001\}$;
5. Batch size B:$\{1024, 2048, 4096, 8192, 16348\}$;
6. Prototype number N_r and M_r: $\{2, 3, 5, 10, 20\}$;
7. Prototype loss interval T: $\{50, 100, 150, 200, 400, 600\}$;
8. α in Equation (4.4) is -4.0 for translational models that evaluate implausibility and 4.0 for all other type-agnostic KGC models.

The type representation feature space in AutoEter has dimension 20 to make the size the same as that for TypeDistMult, TypeComplEx, and TypeTransE. The criterion for tuning the parameters is the mean reciprocal rank (MRR), and the grid search is conducted with Nvidia A6000 GPU (except for a few combinations for TransR.)

4.4.3 Evaluation Metrics and Evaluation Methods

The evaluation metrics are MRR and Hits@N. The proposed method in Chapter 3 uses the same evaluation metrics, too. For the rapid reference purpose, these two evaluation metrics are briefly introduced as follows.

Facts (h, r, t) in validation and test sets were corrupted to $(h, r, ?)$ and $(?, r, t)$ forms. The corrupted forms were fed into the KGC models as queries, and the score functions in the KGC models ranked all entities in the knowledge graph as the candidates for fulfilling the missing one (represented by the question mark "?".) Note that, different from the evaluation procedure in Chapter 3, only known entities in the **training set** are removed before computing the evaluation metrics because this setting is widely used in all base and baseline models.

The MRR of an entity e is defined as follows.

$$\text{MRR}(e) = \frac{1}{\text{rank}(e)},$$

where $\text{rank}(e)$ represents the rank of entity e. For a fact in the test or validation set (h, r, t), let $\text{MRR}_{\text{head}} = \text{rank}(h)$ for $(?, r, t)$ and $\text{MRR}_{\text{tail}} = \text{rank}(t)$ for $(h, r, ?)$. The MRR of this record is computed as

$$\mathrm{MRR}(h,r,t) = \frac{\mathrm{MRR}_{\mathrm{head}} + \mathrm{MRR}_{\mathrm{tail}}}{2}. \tag{4.8}$$

The overall MRR for a dataset $\mathcal{D}$ is

$$\mathrm{MRR} = \sum_{(h,r,t)\in\mathcal{D}} \frac{\mathrm{MRR}(h,r,t)}{|\mathcal{D}|}, \tag{4.9}$$

where $\mathcal{D}$ represents the validation or test set. $|\mathcal{D}|$ is the cardinality of $\mathcal{D}$. The overall MRR is in $(0, 1]$.

For $(h,r,t) \in \mathcal{D}$, if $\mathrm{rank}(h) \leq N$ for the query $(?,r,t)$, $\mathrm{Hits@N}_{\mathrm{head}}(h,r,t) = 1$. Similarly, if $\mathrm{rank}(t) \leq N$ for $(h,r,?)$, $\mathrm{Hits@N}_{\mathrm{tail}}(h,r,t) = 1$. The Hits@N on the dataset $\mathcal{D}$ is:

$$\mathrm{Hits@N} = \frac{1}{|\mathcal{D}|} \sum_{(h,r,t)\in\mathcal{D}} \frac{\mathrm{Hits@N}_{\mathrm{head}}(h,r,t) + \mathrm{Hits@N}_{\mathrm{tail}}(h,r,t)}{2}. \tag{4.10}$$

Similar to the overall MRR, the $\mathrm{Hits@N} \in [0, 1]$ for the dataset $\mathcal{D}$.

4.5 Experiment Result and Discussion

All models are examined on the two following tasks: (1) link prediction; (2) entity clustering. The first task includes experiments on the effect of the number of prototype embeddings on link prediction, and the second task includes the visualization of prototype and entity embeddings to present the ability to capturing implicit type constraints.

4.5.1 Link Prediction

Table 4.2, 4.3, and 4.4 present link prediction results on different datasets. These results indicate that ProtoE can improve most performance metrics for all base type-agnostic KGC models.

Results in Table 4.2 shows that ProtoE outperforms other methods in most metrics in FB15k-237. The improvements for KGC models with a real feature space is more significant than those with a complex space. The reason is that KGC models with complex feature space have twice the number of parameters as those with a real

Table 4.2: Experimental results for FB15k-237. Best values are indicated in bold.

	FB15k-237				
	MRR	**Hits@1**	**Hits@3**	**Hits@5**	**Hits@10**
DistMult	0.2502	0.1646	0.2755	0.3373	0.4279
TypeDistMult	0.2473	0.1647	0.2695	0.3305	0.4168
ProtoEDistMult	**0.2535**	**0.1668**	**0.2799**	**0.3426**	**0.4301**
ComplEx	0.2509	0.1643	0.2762	0.3394	0.4288
TypeComplEx	0.2431	0.1612	0.2655	0.3233	0.4129
ProtoEComplEx	**0.2514**	**0.1647**	**0.2767**	**0.3408**	**0.4290**
TransE	0.2482	0.1644	0.2749	0.3324	0.4190
TypeTransE	**0.2660**	0.1755	**0.2923**	**0.3573**	**0.4540**
ProtoETransE	0.2609	**0.1778**	0.2858	0.3450	**0.4540**
TransR	0.1901	0.1144	0.2072	0.2606	0.3459
ProtoETransR	**0.1984**	**0.1251**	**0.2131**	**0.2667**	**0.3515**
RotatE	0.2647	**0.1810**	0.2886	0.3482	0.4383
AutoEter	0.2476	0.1752	0.2692	0.3216	0.3953
ProtoERotatE	**0.2660**	0.1804	**0.2897**	**0.3539**	**0.4430**

feature space. Therefore, KGC models with a complex feature space need more training examples because of the enlarged number of parameters. Results in table 4.3 and Table 4.4 also support this conclusion because the averaged number of training examples per relation in these two datasets is larger than it is for FB15k-237, as implied by the statistics in Table 4.1.

Table 4.3 presents experiment results on WN18RR. Because facts in WN18RR describe relationships between English, e.g., `hypernym_of` and `hyponym_of`. Therefore, it is difficult to define the type and type constraints on this dataset. In this case, ProtoE uses fewer prototype embeddings (two prototype embeddings for head and tail, respectively) in relations to capture the areas in the feature space where the density of qualified entities (i.e., more probable for the relation to make the triple plausible) locate. Entity embeddings that have a high inner product with the associated prototype embeddings are more probable to have a high score. As a result, the corresponding entities are more likely to be appropriate for the relations.

Note that the improvements for translational models, TransE and TransR, are more

Table 4.3: Experimental results for WN18RR. Best values are indicated in bold.

	WN18RR				
	MRR	**Hits@1**	**Hits@3**	**Hits@5**	**Hits@10**
DistMult	0.4166	**0.3787**	0.4379	0.4564	0.4817
TypeDistMult	0.4052	0.3767	0.4236	0.4364	0.4513
ProtoEDistMult	**0.4173**	0.3765	**0.4403**	**0.4623**	**0.4872**
ComplEx	0.4133	0.3748	0.4343	0.4537	0.4767
TypeComplEx	0.3942	0.3588	0.4185	0.4325	0.4486
ProtoEComplEx	**0.4171**	**0.3781**	**0.4384**	**0.4569**	**0.4817**
TransE	0.1639	0.0038	0.2846	0.3744	0.4419
TypeTransE	**0.2242**	**0.0665**	**0.3545**	**0.4303**	**0.4861**
ProtoETransE	0.1806	0.0160	0.3138	0.3942	0.4580
TransR	0.1535	0.0099	**0.2787**	**0.3389**	**0.3725**
ProtoETransR	**0.2163**	**0.1533**	0.2524	0.2932	0.3385
RotatE	0.4214	0.3827	0.4399	**0.4606**	0.4901
AutoEter	0.4216	0.3843	**0.4402**	0.4596	0.4901
ProtoERotatE	**0.4232**	**0.3866**	**0.4402**	**0.4606**	**0.4919**

significant than those for bilinear models. The reason is that, in TransE and TransR, if (h_1, r, t) and (h_2, r, t) are two facts about r with the same tail entity t, the (projected) embeddings are approximately the same, i.e., $\mathbf{h}_1 \approx \mathbf{h}_2$, because of the intrinsic property of the score functions in these two models (cf. the discussion in Section 4.1.2.) In ProtoE, the prototype embeddings associated with r will affects the head embeddings $\mathbf{h}_1$ and $\mathbf{h}_2$ in the optimization. The prototype embeddings increase the difference $\mathbf{h}_1 - \mathbf{h}_2$ during optimization and help improve the overall MRR in these two models.

Results in Tables 4.2, 4.3, and 4.4 show that the ProtoE can improve the performance of KGC models in different categories. DistMult and ComplEx are bilinear models. TransE and TransR are translational models, and RotatE is akin to the translational model, but it uses rotation instead of offset vectors to compute distance.

Table 4.4: Experimental results for YAGO3-10. Best values are indicated in bold.

	YAGO3-10				
	MRR	**Hits@1**	**Hits@3**	**Hits@5**	**Hits@10**
DistMult	0.4069	0.3070	0.4595	0.5248	0.5996
TypeDistMult	**0.4591**	**0.3540**	**0.5233**	**0.5813**	**0.6504**
ProtoEDistMult	0.4292	0.3338	0.4789	0.5429	0.6141
ComplEx	0.3960	0.2973	0.4471	0.4471	0.5918
TypeComplEx	**0.4521**	**0.3523**	**0.5072**	**0.5697**	**0.6443**
ProtoEComplEx	0.4260	0.3275	0.4819	0.5444	0.6201
TransE	0.3821	0.2736	0.4378	0.5053	0.5985
TypeTransE	0.2384	0.1515	0.2617	0.3301	0.4176
ProtoETransE	**0.4070**	**0.2964**	**0.4634**	**0.5382**	**0.6291**

4.5.2 Effect of the Number of Prototype Embeddings on Link Prediction

Section 4.2.2 mentioned that the number of prototype embeddings associated with each relation could affect the ProtoE's performance. Therefore, another group of experiments on ProtoE with various base models on all three datasets is conducted to investigate the effect. The results of ablation experiments on the number of prototype embeddings are listed in Table 4.5, 4.6, and 4.7.

The number of prototype embeddings is chosen from the range $\{2, 3, 5, 10\}$. The digit in the parenthesis shows the number of prototype embeddings. These results indicate that:

1. ProtoE models with more prototype embeddings have better performance on the datasets FB15k-237 and YAGO3-10. The insufficient number of prototype embeddings will degrade the performance, as discussed in Section 4.2.2. Most type-agnostic KGC models need at minimal five prototype embeddings for each relation to obtaining performance enhancement by ProtoE. In contrast, for the dataset WN18RR, more prototype embeddings incapacitate the performance due to the vagueness of type in the dataset. In this case, the ProtoE depends on the capability of the base KGC model to capture the feasible local areas for entities that are acceptable in different relations.

2. The properties of the knowledge graph and the type-agnostic KGC models simultaneously decide the appropriate number of prototype embeddings. Take TransE with and without ProtoE in Table 4.7 as an example. The insufficient number of prototype embeddings significantly damages the performance. Furthermore. Different base models need different prototype number settings. For example, the ProtoEDistMult and ProtoEComplEx in Table 4.6 need two prototype embeddings only, whereas ProtoETransE and ProtoERotatE need ten.

Table 4.5: Results of ablation experiments for prototype embedding on FB15k-237. Best values are indicated in bold.

	FB15k-237				
	MRR	**Hits@1**	**Hits@3**	**Hits@5**	**Hits@10**
DistMult	0.2502	0.1646	0.2755	0.3373	0.4279
ProtoEDistMult (2)	0.2491	0.1643	0.2742	0.3384	0.4275
ProtoEDistMult (3)	0.2502	0.1641	0.2744	0.3364	0.4295
ProtoEDistMult (5)	0.2503	0.1651	0.2744	0.3363	0.4270
ProtoEDistMult (10)	**0.2535**	**0.1668**	**0.2799**	**0.3426**	**0.4301**
ComplEx	0.2509	0.1643	0.2762	0.3394	0.4288
ProtoEComplEx (2)	0.2512	**0.1648**	**0.2770**	0.3394	**0.4297**
ProtoEComplEx (3)	0.2488	0.1639	0.2732	**0.3663**	0.4249
ProtoEComplEx (5)	0.2497	0.1645	0.2753	0.3354	0.4240
ProtoEComplEx (10)	**0.2514**	0.1647	0.2767	0.3408	0.4290
TransE	0.2482	0.1644	0.2749	0.3324	0.4190
ProtoETransE (2)	0.1874	0.1276	0.2022	0.2437	0.3080
ProtoETransE (3)	0.1879	0.1294	0.2009	0.2522	0.3249
ProtoETransE (5)	0.2351	0.1543	0.2584	0.3158	0.3978
ProtoETransE (10)	**0.2609**	**0.1778**	**0.2858**	**0.3450**	**0.4540**
RotatE	0.2647	**0.1810**	0.2886	0.3482	0.4383
ProtoERotatE (2)	0.2607	0.1772	0.2839	0.3459	0.4353
ProtoERotatE (3)	0.2591	0.1755	0.2831	0.3446	0.4322
ProtoERotatE (5)	0.2533	0.1694	0.2777	0.3385	0.4265
ProtoERotatE (10)	**0.2660**	0.1804	**0.2897**	**0.3539**	**0.4430**

Table 4.6: Results of ablation experiments for prototype embedding on WN18RR. Best values are indicated in bold.

	WN18RR				
	MRR	**Hits@1**	**Hits@3**	**Hits@5**	**Hits@10**
DistMult	0.4166	**0.3787**	0.4379	0.4564	0.4817
ProtoEDistMult (2)	**0.4173**	0.3765	**0.4403**	**0.4623**	**0.4872**
ProtoEDistMult (3)	0.4086	0.3757	0.4271	0.4454	0.4655
ProtoEDistMult (5)	0.4038	0.3693	0.4249	0.4405	0.4561
ProtoEDistMult (10)	0.4075	0.3700	0.4304	0.4443	0.4671
ComplEx	0.4133	0.3748	0.4343	0.4537	0.4767
ProtoEComplEx (2)	**0.4171**	**0.3781**	**0.4384**	**0.4569**	**0.4817**
ProtoEComplEx (3)	0.4106	0.3746	0.4319	0.4467	0.4690
ProtoEComplEx (5)	0.4099	0.3751	0.4314	0.4461	0.4652
ProtoEComplEx (10)	0.4093	0.3725	0.4296	0.4486	0.4703
TransE	0.1639	0.0038	0.2846	0.3744	0.4419
ProtoETransE (2)	0.0234	0.0134	0.0247	0.0299	0.0415
ProtoETransE (3)	0.0305	0.0006	0.0330	0.0477	0.0766
ProtoETransE (5)	0.0950	0.0009	0.1471	0.2039	0.2551
ProtoETransE (10)	**0.1806**	**0.0160**	**0.3138**	**0.3942**	**0.4580**
RotatE	0.4214	0.3827	0.4399	0.4606	0.4901
ProtoERotatE (2)	0.4200	0.3829	0.4352	0.4574	0.4887
ProtoERotatE (3)	0.4226	0.3848	0.4389	0.4582	0.4928
ProtoERotatE (5)	0.4222	0.3834	**0.4405**	**0.4614**	**0.4938**
ProtoERotatE (10)	**0.4232**	**0.3866**	0.4402	0.4606	0.4919

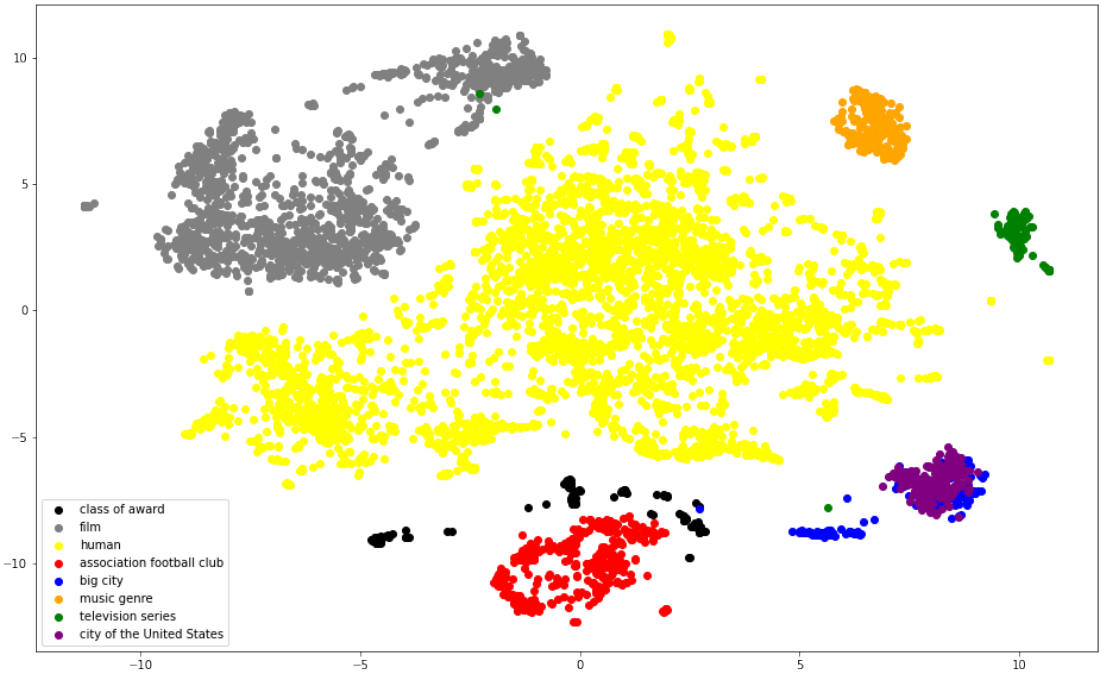
Table 4.7: Results of ablation experiments for prototype embedding on YAGO3-10. Best values are indicated in bold.

	YAGO3-10				
	MRR	**Hits@1**	**Hits@3**	**Hits@5**	**Hits@10**
DistMult	0.4069	0.3070	0.4595	0.5248	0.5996
ProtoEDistMult (2)	0.4025	0.3041	0.4527	0.5199	0.5969
ProtoEDistMult (3)	0.4246	0.3238	**0.4969**	**0.5604**	**0.6360**
ProtoEDistMult (5)	0.4175	0.3220	0.4682	0.5277	0.6046
ProtoEDistMult (10)	**0.4292**	**0.3338**	0.4789	0.5429	0.6141
ComplEx	0.3960	0.2973	0.4471	0.4471	0.5918
ProtoEComplEx (2)	0.3879	0.2880	0.4378	0.5045	0.5836
ProtoEComplEx (3)	0.3874	0.2896	0.4357	0.5001	0.5816
ProtoEComplEx (5)	0.3692	0.2707	0.4175	0.4827	0.5657
ProtoEComplEx (10)	**0.4260**	**0.3275**	**0.4819**	**0.5444**	**0.6201**
TransE	0.3821	0.2736	0.4378	0.5053	0.5985
ProtoETransE (2)	0.0433	0.0359	0.0425	0.0452	0.0433
ProtoETransE (3)	0.0430	0.0359	0.0420	0.0462	0.0527
ProtoETransE (5)	0.1016	0.0452	0.1176	0.1488	0.1966
ProtoETransE (10)	**0.4070**	**0.2964**	**0.4634**	**0.5382**	**0.6291**

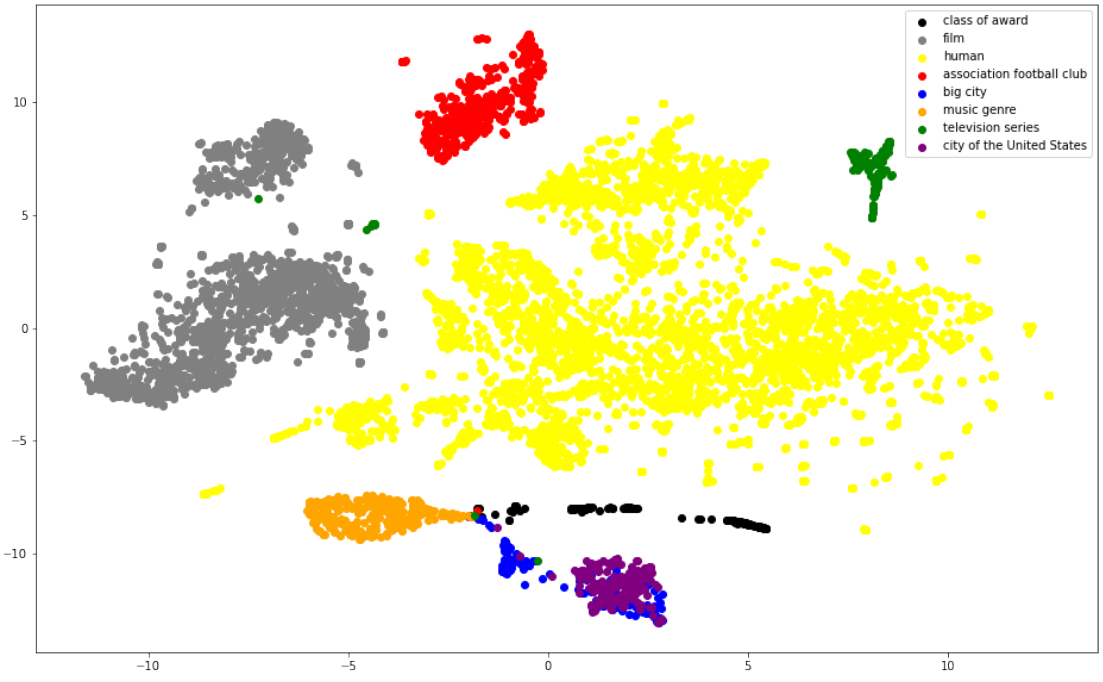
4.5.3 Entity Embedding Clustering

Besides improving the link prediction performance, the ProtoE empowers type-agostic KGC models with the ability to capture implicit entity types, i.e., the co-occurrence tendency of entities. The following experiments evaluate such ability by clustering all entities that are instances of "class of award, film, human, association football club, big city, music genre, television series" and "city of the United States" in the FB15k-237. These instances are extracted from the WikiData by the "instance_of" relation. Entities with these five instances take 53.35% (7,758/14,541) of all entities in the FB15k-237 dataset.

The visualization of entity embeddings uses t-SNE [94], and ProtoE learns entity embeddings with ten prototype embeddings and DistMult as the base model. Figure 4.3 shows the clustering results.



(a) ProtoEDistMult



(b) DistMult

Figure 4.3: Clustering results of entity embeddings for ProtoEDistMult and DistMult.

The clustering results in Figure 4.3 indicate that:

1. DistMult and ProtoEDistMult can distinguish entities with "film, human," and "television series." These three clusters have wide margins, whereas entities with "big city" and "city of the United States" have overlaps because of the shared entities in both instances.

2. Entity embeddings learned by DistMult with "class of award, big city," and "music genre" intersect, whereas the entity embeddings learned by ProtoEDistMult can distinguish them. The ability to distinguish these entities with different instances is brought by the calibrated entity embeddings in ProtoEDistMult with the prototype embeddings associated with relations that appear in facts about these entities.

The prototype embeddings have another role in ProtoE, i.e., capturing the local areas in the feature space in which the embeddings of proper entities for the relation tends to stay. The case study of this role starts with the relation "olympic_sports" [1] as an example. Appendix A presents more case study examples on different relations. Figure 4.4 presents the clustering result.

There are 581 records in training set for this relation, with 40 different head entities and 51 different tail entities. The head entities are the Olympic Games in different years, and the tail entities are sports in these games. Because of the low number of training examples and entities in the relation, some prototype embeddings have become redundant - these embeddings barely change their locations during the optimization. However, this does not prevent the prototype embeddings in the left-top and right-bottom from capturing the proper local areas for entity embeddings that are acceptable in the relation.

Moreover, another clustering result on the relation "performed_in" [2] present the locations of entity and prototype embeddings when the number of training examples in the relation is high. The head entity is an actor or actress and, the tail entity is a film in which they acted in. Prototype embeddings in this relation converge to almost the same location because of the monotonous entity instance.

[1] The full relation path is "/user/jg/default_domain/olympic_games/sports" in FB15k-237.
[2] The full relation path is "/film/actor/film./film/performance/film" in FB15k-237.

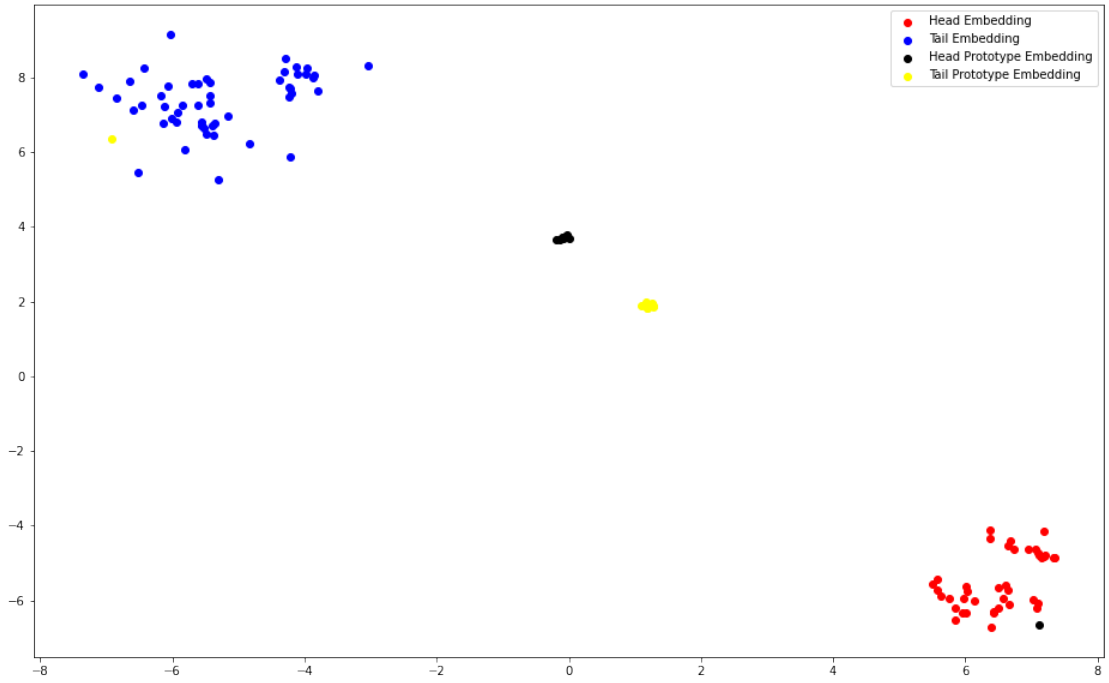


Figure 4.4: Visualization of prototype and entity embeddings in *olympic_sports* from FB15k-237.

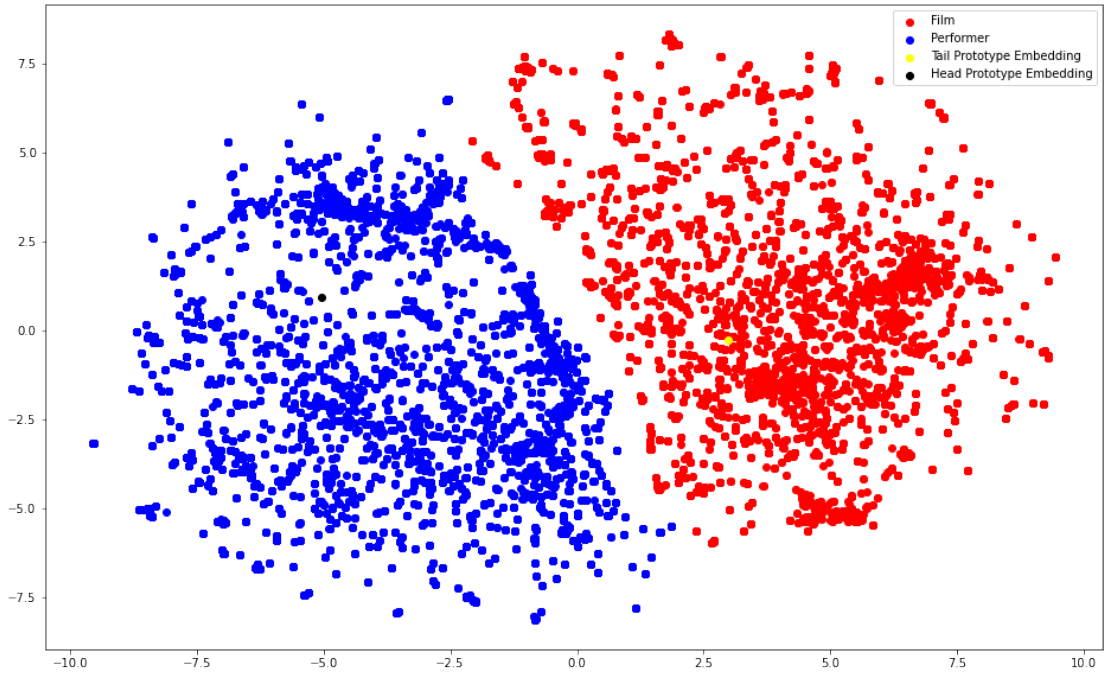
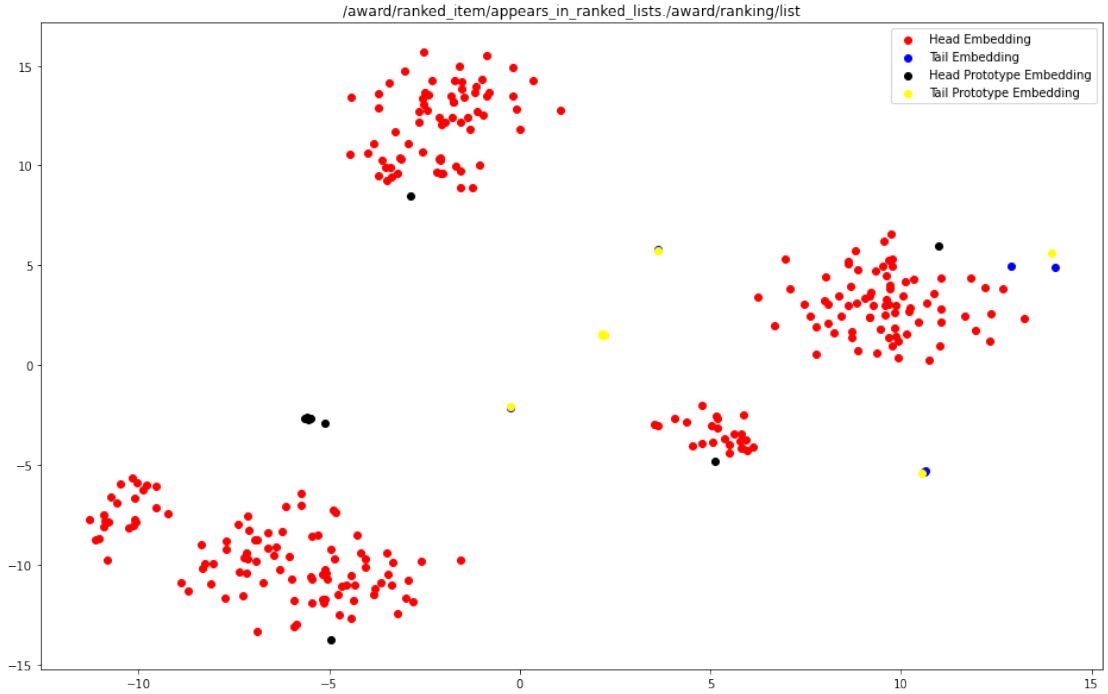


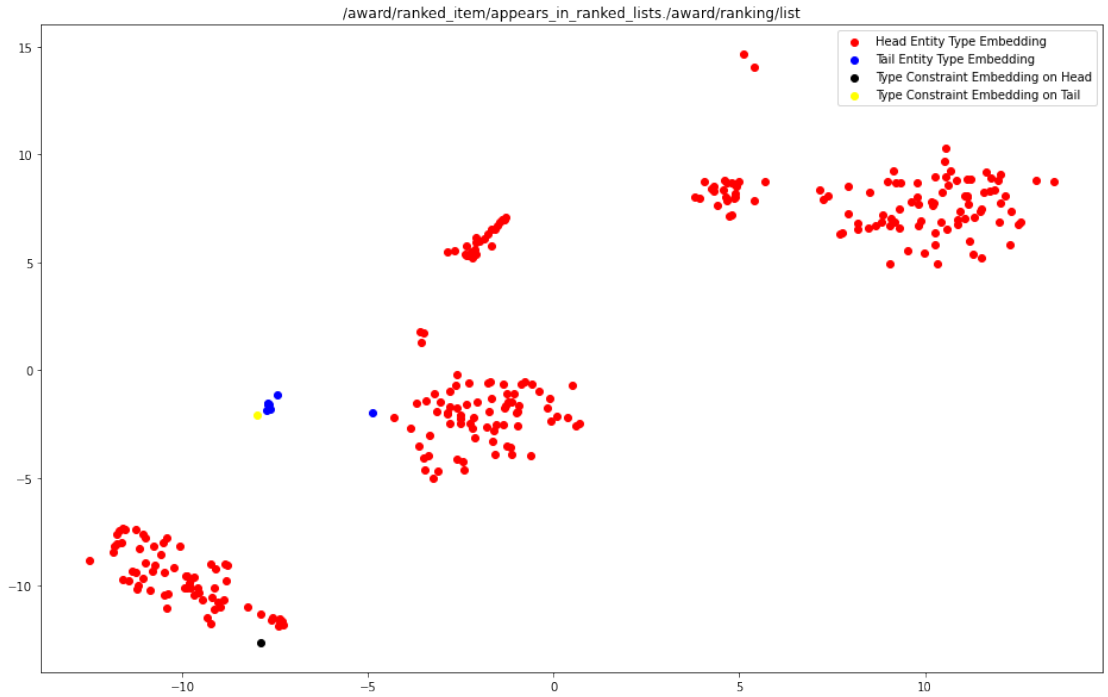
Figure 4.5: Entity and prototype embeddings related to *performed_in* in the dataset FB15k-237.

Figure 3.4 and Figure 4.5 present the case studies about monotonous type constraints, i.e., acceptable entities are in a (relatively) singular local area in the feature space. As mentioned before, ProtoE is capable of capturing multiple type constraints by prototype embeddings. These prototype embeddings play as a anchor of the local areas in which proper entity embeddings stay in. Figure 4.6 presents the case study on a relation (relation path is presented at the top of the figure) about multiple type constraints.

The relation contains 273 different head entities and 7 different tail entities, and 355 records in the training data. The results show that both ProtoEDistMult and TypeDistMult could recognize the multiple implicit head entity types. However, TypeDistMult failed to capture the constraints due to the singular embedding setting. In contrast, ProtoEDistMult uses multiple head prototype embeddings to represent the four clusters where head entity embeddings should stay in the feature space.



(a) Clustering with Entity Embeddings Learned by ProtoEDistMult



(b) Clustering with Entity Embeddings Leanred by TypeDistMult

Figure 4.6: Clustering results for TypeDistMult and ProtoEDistMult.

Table 4.8: Ablation experiment results of ProtoE and the co-occurrence based method on FB15k-237.

	FB15k-237			
	MRR	Hits@1	Hits@3	Hits@5
DistMult	0.2888	0.1995	0.3194	0.3818
CoDM	0.2950	0.2091	**0.3262**	**0.3823**
ProtoEDistMult	**0.2961**	**0.2124**	0.3239	0.3821
ComplEx	0.2896	**0.2079**	0.3166	0.3752
CoCX	0.2904	0.2063	0.3188	0.3779
ProtoEComplEx	**0.2942**	0.2058	**0.3251**	**0.3865**

Table 4.9: Ablation experiment results of ProtoE and the co-occurrence based method on YAGO3-10.

	YAGO3-10			
	MRR	Hits@1	Hits@3	Hits@5
DistMult	0.4191	0.3255	0.4717	0.5343
CoDM	**0.4912**	**0.4011**	**0.5474**	**0.5961**
ProtoEDistMult	0.4229	0.3320	0.4740	0.5274
ComplEx	0.4133	0.3193	0.4647	0.5232
CoCX	**0.4932**	**0.4064**	**0.5459**	**0.5941**
ProtoEComplEx	0.4253	0.3235	0.4819	0.5520

4.5.4 Ablation Experiments with the Co-occurrence Method

The co-occurrence based method proposed in Chapter 3 and the ProtoE follow the principle of adjusting entity embeddings. The co-occurrence based method is applicable to the bilinear models, whereas ProtoE has a more comprehensive application range. To show the differences on their performances for the bilinear models, this section contains experiment results of both methods with the same experiment settings.

Section 3.3.3 mentioned that for a fair comparison, the evaluation metrics follow different filtering methods in this chapter. The evaluation metrics in these ablation experiments follow the same filtering strategy introduced in Section 3.3.3. Table 4.8 and 4.9 present the ablation experiment results comparing ProtoE and the co-occurrence based method in Chapter 3 on FB15k-237 and YAGO3-10 datasets, respectively. The

Table 4.10: Statistics of datasets (the contents are the same as a part of those in Table 3.1.)

	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	Train	Valid	Test	Density
FB15k-237	14,541	237	272,115	17,535	20,466	0.0381
YAGO3-10	123,182	36	1,079,040	5,000	5,000	0.1610

base models in Table 4.8 and 4.9 are bilinear KGC models, namely, DistMult and ComplEx. The performances of DistMult, ComplEx, and the corresponding enhanced variances, CoDM and CoCX are from Table 3.2 in Chapter 3.

For FB15k-237, the ProtoEDistMult outperforms its counterparts CoDM on MRR and Hits@1, whereas CoDM has a better performance on Hits@3 and Hits@5. For ComplEx on the FB15k-237 dataset, ProtoEComplEx has a better improvement compared to CoCX. Meanwhile, for YAGO3-10, the co-occurrence based methods, CoDM and CoCX, outperform their counterparts ProtoEDistMult and ProtoEComplEx on a large margin for all performance measures.

The reason of the different improvements for the co-occurrence based method and the ProtoE is the co-occurrence density (defined in Equation 3.14.) Table 4.10 presents the co-occurrence density of FB15k-237 and YAGO3-10. The high co-occurrence density in YAGO3-10 enables the co-occurrence based method to calibrate the entity embeddings better than the ProtoE to reflect the implicit entity type and type constraints, whereas the relatively low co-occurrence density in FB15k-237 limits the co-occurrence based method's ability to adjust entity embeddings. Note that this effect is consistent with the discussion in Section 3.4.1: the co-occurrence method depends on the entity co-occurrence density.

4.6 Conclusion

This chapter proposed ProtoE, a general method for both bilinear and translational KGC models to capture implicit entity type and type constraints in relations. ProtoE can be treated as an extension of the method proposed in Chapter 3 by dropping the global statistics about entity co-occurrences. Different from supervised methods which

requires type annotations; ProtoE depends on only facts in the knowledge graph and is different from other unsupervised approaches, which could only capture a single type constraint in relations; ProtoE use multiple prototype embeddings to capture multiple impilicit type constraints.

ProtoE can extend both type-agnostic bilinear and translational models. Prototype embeddings in the head and tail locations of relations represent the local areas in which embeddings of entities that satisfy the type constraints are located. Prototype embeddings and entity and relation embeddings in the underlying KGC models are simultaneously optimized with facts in the knowledge graph. An additional loss function helps calibrate the entity and prototype embedding locations to better represent the diverse type constraints. From the perspective of representation learning, the prototype embeddings are representations of the implicit type constraints.

The ability of prototype embeddings to capture implicit type constraints depends on the number of training data about the relation. Prototype embeddings will converge to locations where the density of qualified entity embeddings is high for relations with sufficient training data. In contrast, some prototype embeddings would be orphans in the feature space when the number of training data for the relation is low. In the current setting, the number of prototype embeddings associated with relations is a hyperparameter, and all relations have the same number of prototype embeddings. A possible future work is to use some stochastic process, such as the Chinese restaurant process [98, 99] or determinantal point process [100, 101], to automatically set an appropriate number of prototype embeddings in a relation-specific manner.

The experiment results for link prediction and the case studies by entity clustering show that the ProtoE improves the performance of base KGC models even in the scenario where it is difficult to define the type on. The entity clustering results indicate that the proposed method, ProtoE, captures implicit entity type and type constraints in a better way compared to other unsupervised approaches. The ablation experiment results with the co-occurrence based method in Chapter 3 show that ProtoE is more capable of learning implicit entity type and type constraint representations in the scenario where the entity co-occurrence is low.

5

Hypernetwork based Meta-Learning Recommender System for the User-side Cold-start Problem

As mentioned in the Section 1.3, the proposed system aims fast to adapt features and weights in the recommender system to enable high-quality recommendation, albeit under little user-item interaction data. The proposed system has two components: a neural network f_θ that recommenders items to users with parameter θ, and a hypernetwork g which is another neural network that generates θ in f_θ. The hypernetwork g takes takes user interest embeddings as indicators to generate the appropriate parameters. From the perspective of representation learning, the representation of users are generated by g. The personalized θ in f and the user embedding simultaneously represent user preferences. The following sections introduce the details of the proposed system.

5.1 Proposed Method: HyperRS

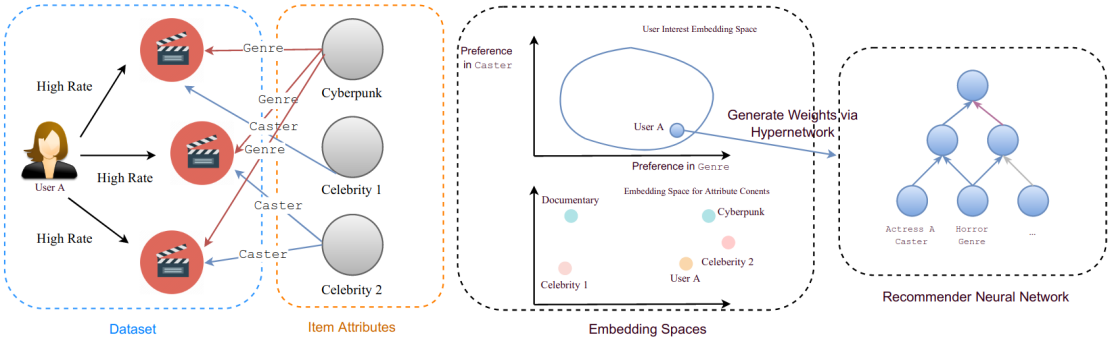


Figure 5.1: The example of the proposed method capturing the interest of User A on item attributes ("Genre" and "Caster") and contents in item attributes ("cyberpunk" in "Genre", "celebrity 1", "celebrity 2" in "Caster").

The intuition of the proposed method is to capture user interest in item attributes and the contents of item attributes. Take the domain "Movie" as an example. Suppose items in the "Movie" domain have two attributes: Caster and Genre. Figure 5.1 shows User A's item interactions and the structure of recommender network f and the hypernetwork that generates parameters in f. Every user in the proposed method has two embeddings: an embedding used for recommending items that match the user's interest; and another embedding that generates user-specific parameters in the recommender network f. In the feature space of recommender network f (shown in the middle of Figure 5.1), the user embedding of A is close to the attribute content embeddings that they has interacted with ("Celebrity 2" and "Cyberpunk ") after optimization. The parameters that project these embeddings to the next layer should be large (i.e., most neurons correspond to "Caster" and "Genre" should be activated) after generated by the user embedding for the hypernetwork.

The top-middle of Figure 5.1 shows the feature space of *user interest embeddings* in the hypernetwork. This feature space is a subspace spanned by the *user interest basis*. The user interest embedding is the input to the hypernetwork for generating parameters in the recommender network, and is a mixture of user interest basis. Following sections introduce the details of the proposed method. Because the proposed method depends on a hypernetwork to provide parameters in the recommender network, it is referred to as HyperRS, abbreviated for Hypernetwork-based Recommender System, in the

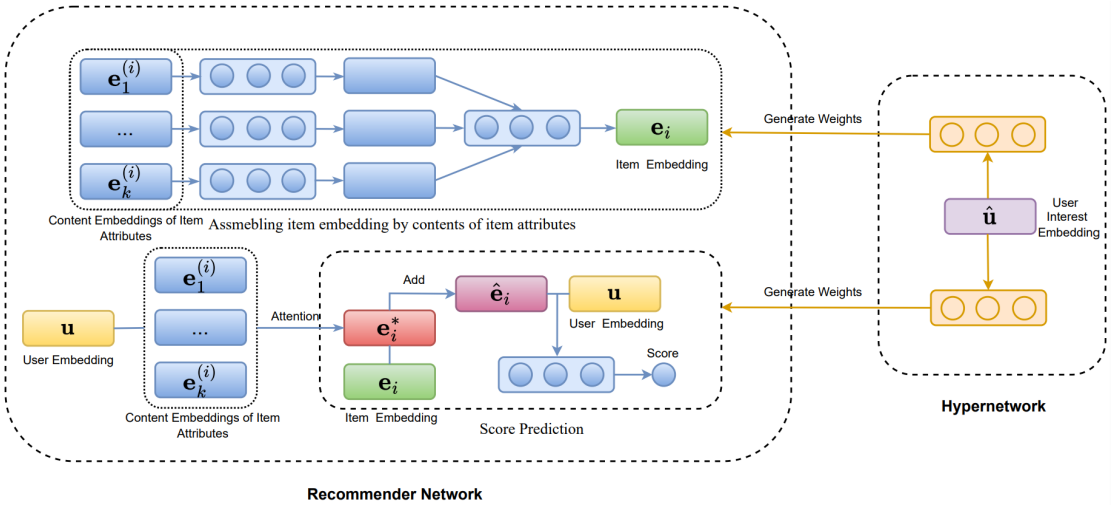


Figure 5.2: The structure of HyperRS.

following contexts.

Figure 5.2 presents the structure of HyperRS. The recommender network (left) constitutes item embedding by the item profile and takes item and user embeddings to predict the user's interest in the item. Hypernetwork (right) takes the user's interest embedding to generate all parameters in the recommender network. Following sections introduce all components in Figure 5.2.

5.1.1 Embeddings in the HyperRS

The user-interaction records are in the format $D = \{(u, i, s)|u \in U, i \in I, s \in S\}$. U and I are the set of users and items, respectively. S is the set of scores. In most scenarios, S is a finite set that defines the score values. If the score is in the range of non-negative real number, $S = \mathbb{R}^+ \cup \{0\}$.

Every user $u \in U$ has two embeddings: a user embedding $\mathbf{u}$ used in the recommender network and another user interest embedding $\hat{\mathbf{u}}$ used in the hypernetwork. The user embedding $\mathbf{u}$ is constructed by embedding-lookup for the user id, and $\hat{\mathbf{u}}$ is constituted by the mixture of user interest basis. The details of $\hat{\mathbf{u}}$ is introduced in Section 5.1.3.

Every item $i \in I$ contains a combination of attributes: $p(i) = (p_1^{(i)}, ..., p_k^{(i)})$. k is the number of attributes. The attribute contents are encoded as one-hot or multi-hot vectors $(\mathbf{p}_1^{(i)}, ..., \mathbf{p}_k^{(i)})$. The embedding of the item attribute content $\mathbf{p}_x^{(i)}$ is constituted as

$$\mathbf{e}_x^{(i)} = \mathbf{M}_x \mathbf{p}_x^{(i)}; \quad x = 1, 2, ..., k; \tag{5.1}$$

The column vectors in matrix $\mathbf{M}_x$ are embeddings of all contents in attribute x.

5.1.2 The Recommender Network Structure in HyperRS

Figure 5.2 presents the structure of the recommender network in HyperRS on the left side. The hypernetwork generates all parameters in the recommender network.

Item embeddings are constituted by item attribute contents embeddings in Equation 5.1. A hidden layer transforms the item content embeddings and merges them into the item embedding:

$$\hat{\mathbf{e}}_x^{(i)} = \phi(\mathbf{W}_x \mathbf{e}_x^{(i)} + \mathbf{b}_x); \qquad x = 1, 2, ..., k. \tag{5.2}$$

$$\mathbf{e}_i = \phi\big(\mathbf{W}_t[\hat{\mathbf{e}}_1^{(i)} \oplus ... \oplus \hat{\mathbf{e}}_k^{(i)}] + \mathbf{b}_t\big). \tag{5.3}$$

$\oplus$ is the operator of vector concentration. $\mathbf{W}_x, \mathbf{W}_t$ are weights and $\mathbf{b}_x, \mathbf{b}_t$ are bias terms. ϕ is the activation function.

The user embedding in the recommender network captures user interest in the item attribute contents. The recommender network uses a residual block [102] with attention to achieving this task.

$$\mathbf{e}_i^* = \text{attention}(\mathbf{u}, \mathbf{P}). \tag{5.4}$$

$$\hat{\mathbf{e}}_i = \mathbf{e}_i^* + \mathbf{e}_i. \tag{5.5}$$

Column vectors in the matrix $\mathbf{P}$ are item attribute content embeddings $\mathbf{e}_x^{(i)}$ in Equation 5.1. $\mathbf{u}$ is the user embedding. A four-layer neural network f_θ predicts the score with the user and item embedding:

$$\hat{s}^{(u,i)} = f_\theta(\hat{\mathbf{e}}_i \oplus \mathbf{u}). \tag{5.6}$$

The parameter in f_θ contains $\theta = \{\mathbf{W}_{\ell_1}, \mathbf{b}_{\ell_1}, \mathbf{W}_{\ell_2}, \mathbf{b}_{\ell_2}, \mathbf{W}_{\ell_3}, \mathbf{b}_{\ell_3}, \mathbf{W}_{\ell_4}\}$. $\mathbf{W}_{\ell_j}$ and $\mathbf{b}_{\ell_j}$ are weights and bias term in layer ℓ_j, respectively. The last hidden layer does not have bias

term.

The loss function in HyperRS is

$$\mathcal{L}(s^{(u,i)}, \hat{s}^{(u,i)}) = (s^{(u,i)} - \hat{s}^{(u,i)})^2. \tag{5.7}$$

$s^{(u,i)}$ is the score user u assigned to item i as the preference indicator. $\hat{s}^{(u,i)}$ is the predicted score by the recommender network.

All parameters in the recommender network are in the parameter set $\mathbf{\Theta} = \{\mathbf{W}_x, \mathbf{b}_x, \mathbf{W}_t, \mathbf{b}_t | x = 1, 2, ..., k\} \cup \theta$.

5.1.3 The Hypernetwork Structure in HyperRS

As mentioned in Section 5.1.2, the hypernetwork in HyperRS generates all parameters in $\mathbf{\Theta}$. The hypernetwork takes user interest embedding $\hat{\mathbf{u}}$ as input:

$$\hat{\mathbf{u}} = \mathbf{M}_u \mathbf{u}^*. \tag{5.8}$$

$\mathbf{M}_u \in \mathbb{R}^{p \times q}$ and $\mathbf{u}^* \in \mathbb{R}^q$. $q < p$. The column vectors in the matrix $\mathbf{M}_u$ are user interest basis mentioned in Section 5.1.1. $\mathbf{u}^*$ is constituted by embedding-lookup of the user id. Elements in the vector $\mathbf{u}^*$ represent weights of different user interest basis.

The motivation of Equation 5.8 is from latent embedding optimization (LEO) [103]. The intuition of Equation 5.8 is to learn the user interest basis in $\mathbf{M}_u$ from existing users to enable rapid adaption of $\hat{\mathbf{u}}$ for new users. Because $\mathbf{u}^*$ represent weights of user interest basis and the combination of user interest basis is optimized by back-propagation, the $\hat{\mathbf{u}}$ is a mixture of user interest basis (i.e., the column vectors in $\mathbf{M}_u$.) The optimized $\mathbf{M}_u$ should allow constituting $\hat{\mathbf{u}}$ for new users by only a few steps that change elements in $\mathbf{u}^*$.

Note that because the hypernetwork takes $\hat{\mathbf{u}}$ to generate all parameters in the recommender network, the fast adaption of $\hat{\mathbf{u}}$ by only changing combination weights in $\mathbf{u}^*$ ensures the parameters in the recommender network to be adequately generated *simultaneously*.

The hypernetwork is constituted by one-layer neural networks:

$$\theta^* := \phi(\mathbf{W}_{\theta^*} \hat{\mathbf{u}} + \mathbf{b}_{\theta^*}) \qquad \theta^* \in \mathbf{\Theta}. \tag{5.9}$$

The simple structure in Equation 5.9 prevents the number of parameters in the hypernetwork from blowing. The number of parameters in the hypernetwork increases vastly with the number of layers: suppose θ^* is a $i \times j$ matrix, because the number of elements in $\hat{\mathbf{u}}$ is p, $\mathbf{W}_{\theta^*}$ will have $i \times j \times p$ parameters. ϕ is the activation function. The parameter set $\Phi = \{\mathbf{W}_{\theta^*}, \mathbf{b}_{\theta^*}, \mathbf{M}_u | \theta^* \in \boldsymbol{\Theta}\}$ includes all hypernetwork parameters.

5.1.4 Optimization

Algorithm 5.1 Optimization Procedure

Data: Support Set $\mathcal{S}$ and Query Set $\mathcal{Q}$. $\mathcal{S}, \mathcal{Q} \subset \mathcal{D}$. Learning rates α, β.

Randomly initialize parameters $\boldsymbol{\Phi}$ in the hypernetwork;
for $i \leftarrow 1$ **to** N **do**
  for $u \in \mathcal{U}_{train}$ **do**
    Initialize parameters in $\boldsymbol{\Theta}$ by Equation 5.8 and 5.9;
    for $(u, i, s) \in \mathcal{S}_u$ **do**
      Predict $\hat{s}^{(u,i)}$ by the recommender network;
      $\forall \boldsymbol{\gamma} \in \boldsymbol{\Theta} : \boldsymbol{\gamma} := \boldsymbol{\gamma} - \alpha \nabla_{\boldsymbol{\gamma}} \mathcal{L}(s, \hat{s}^{(u,i)})$;
      $\mathbf{u}^* := \mathbf{u}^* - \alpha \nabla_{\mathbf{u}^*} \mathcal{L}(s, \hat{s}^{(u,i)})$
    for $(u, i, s) \in \mathcal{Q}_u$ **do**
      Predict $\hat{s}^{(u,i)}$ by the recommender network; $\forall \boldsymbol{\gamma} \in \boldsymbol{\Phi} : \boldsymbol{\gamma} := \boldsymbol{\gamma} - \beta \nabla_{\boldsymbol{\gamma}} \mathcal{L}(s, \hat{s}^{(u,i)})$;
      $\mathbf{M}_x = \mathbf{M}_\mathbf{x} - \beta \nabla_{M_x} \mathcal{L}(s, \hat{s}^{(u,i)})$;
      $\mathbf{u} = \mathbf{u} - \beta \nabla_{\mathbf{u}} \mathcal{L}(s, \hat{s}^{(u,i)})$;

The optimization procedure for HyperRS follows the setting in meta-learning [2, 103]. The dataset D is divided into three subsets: D_{train}, D_{valid}, and D_{test}. Users in these sets are not intersected. Every user u has two sets: the support set S_u and the query set Q_u. Items in S_u and Q_u are not intersected.

The hypernetwork takes user interest embedding $\hat{\mathbf{u}}$ to initialize parameters in the recommender network for every user u. The parameters in Θ are optimized by gradient back-propagation with data in S. The updated recommender network takes data in Q to update the parameters in Φ.

Algorithm 5.1 describes the optimization procedure for HyperRS and all baseline models in the experiment section. In addition, negative sampling is used for all models during optimization. The negative samples are items that are not interacted with by

the user. The scores for negative samples are 0.

5.2 Experiment Setting

This section introduces datasets used in experiments, a brief summary baseline models, the details of performance evaluation metrics, and hyperparameter settings. The experiments use three datasets with different user sizes to test the capability of HyperRS in different data-scale. The following sections contain the details of the experiments.

5.2.1 Datasets

The experiment uses three datasets: Movielens-10m [104], BookCrossing [105], and a TokyoTV dataset which contains one month of TV program watch logs by residents in Tokyo 23 wards (prepared by M Data Co., Ltd. and provided by MELCO.) The BookCrossing dataset is abbreviated as "Book" in the following contexts.

Item attributes of Movielens-10m are from WikiData to keep both datasets with the same attributes. Movies with incomplete features are dropped from the Movielens-10m dataset. The scores of Movielens-10m are from 0.5 to 5 with an interval of 0.5, whereas the scores of TokyoTV range from 1 to 3 with an interval of 1.

The data in Book dataset is a mixture of explicit and implicit scores. Items that are interacted by the user but not explicitly scored have a 0 score, whereas the explicit score is in 0 to 10 with an interval of 1. To make the dataset compatible with all baseline models, the scores in Book dataset are normalized as follows: all interacted items (regardless whether explicitly scored or not) are changed to be with the score 1 because the scores in the Book dataset are skewed. Most of explicitly scored items are with scores close to 10. Item attributes of Book dataset are extracted by the data dump of OpenLibrary [1]. The item attributes take authors as "Character," publishers as "Creator," and subjects as "Genre." Table 5.1 shows the statistics of these three datasets.

Every user in Movielens-10m and Tokyo TV datasets only keeps 20 records to simulate the scenario in which user-item interactions are rare. For the Book dataset, each user keeps only 10 records. In addition, the capability of HyperRS in different scales of the number of users is evaluated on these two datasets: Movielens-10m has

[1] https://openlibrary.org

Table 5.1: Statistics of preprocessed Movielens-10m, TokyoTV, and Book datasets.

	Data	User	Item	Creator	Character	Genre
Movielens-10m	6,601,340	58,711	6,178	3,571	28,252	273
TokyoTV	7,183	70	579	7	9,374	11
Book	580, 476	7,383	82,790	41,533	10,829	37,780

many users, whereas the TokyoTV dataset only has few users. The ratios of users in the training, validation, and test set are 60%, 20%, and 20%, respectively.

The criteria for choosing the item attributes in Table 5.1 are: (1) sharing a father node in the knowledge graph, or (2) semantically similar in the textual description. The director, broadcast TV station, and author are used as "Creator" for Movielens-10m, Tokyo TV, and Book datasets, respectively. The caster member is used as "Character" in Movielens-10m and Tokyo TV datasets, and it is an attribute existing for the Book dataset. "Genre" exists for all three datasets. The structure used for choosing these three attributes is from Wikidata, and the item contents are fulfilled with the OpenLibrary for the Book dataset.

5.2.2 Performance Measures

Two measures evaluates all models: (1) binary normalized discounted cumulative gain (NDCG) and (2) Recall@N. Both measures are in $[0, 1]$ and higher values indicate better performance. The binary NDCG@N is defined as

$$\text{DCG@}N(\mathbf{p}) = \sum_{i=1}^{N} \frac{2^{\mathbf{1}(p_i \in \mathbf{t})} - 1}{\log_2(i+1)}, \tag{5.10}$$

$$\text{NDCG@}N(\mathbf{p}, \mathbf{t}) = \frac{\text{DCG@}N(\mathbf{p})}{\text{DCG@}N(\mathbf{t})}. \tag{5.11}$$

$\mathbf{p}$ is a vector with the ids of items with N highest predicted scores to a user. $\mathbf{t}$ is a vector of the user-interacted item ids. $\mathbf{1}(s)$ is 1 if the statement s is true, otherwise $\mathbf{1}(s)$ is 0. The NDCG@N of a set D is divided by the number of users in D:

$$\text{NDCG@}N(D) = \frac{1}{|U(D)|} \sum_{u \in U(D)} \text{NDCG@}N(\mathbf{p}(u), \mathbf{t}(u)). \tag{5.12}$$

$U(D)$ is the set of users in D. $\mathbf{p}(u)$ is a vector of item ids with top-N predicted scores to user u. $\mathbf{t}(u)$ is the vector of user u's interacted item ids.

In validation and testing phrases, the local parameters in all models are optimized by data in the support set, and the optimized models are tested on the query set.

5.2.3 Baseline Models and Hyperparameter Settings

The baseline models in the experiments are listed as follows.

1. MeLU [38] is the vanilla model that uses the MAML framework. It uses the user and item profiles as features. The user profile is replaced by a user embedding in the experiment setting.

2. s^2Meta [75] uses gates in the local and global parameter updating to accelerate the adaption of local parameters. It utilizes REINFORCE [82] algorithm to learn gate parameters.

3. MetaTL [40] converts items in the support set as the user's representation. It follows a sequential recommendation setting that uses the previous k items in the sequence to predict the score of the $k+1$ item. The k items are used to generate the transition embedding from the k item to the $k+1$ item.

4. Mecos [106] attempts to learn the parameters in the recommender network that match the representation of the support set S_u to the query set Q_u's representation. The intuition of Mecos is akin to MetaTL: both models learn the representation of the support set to help the recommender network predict the user's preference in query set items. It uses an LSTM [107] cell to match support and query sets.

The activation functions in HyperRS are LeakyReLU and ReLU. The user and item content embeddings have 128 dimensions. The user interest basis in the hypernetwork has 256 dimensions, and the user interest embedding is 128 dimensions. For the Book dataset, the dimensions of user and item attribute content embeddings are 64, and the dimensions for the user interest embedding and user interest basis are 64 and 128, respectively. The optimizer for local steps is Adam [92]. Hyperparameters are optimized by the grid search and fine-tuning. The ranges of different hyperparameters in grid search are:

Table 5.2: Experiment results of NDCG@N measures on Book dataset.

	Book							
	NDCG@5		NDCG@10		NDCG@20		NDCG@50	
	C-W	C-C	C-W	C-C	C-W	C-C	C-W	C-C
MeLU	0.1524	0.034	0.1861	0.0356	0.2086	0.0388	0.2406	0.0446
s^2Meta	0.3748	0.1388	0.3829	0.1398	0.3874	0.1443	0.3972	0.1469
MetaTL	0.3691	0.1467	0.3892	0.1556	0.4039	0.1615	0.4203	0.1674
Mecos	**0.4196**	**0.3693**	**0.4285**	**0.3774**	**0.4386**	**0.3862**	**0.4485**	**0.3898**
HyperRS	0.2670	0.1727	0.3113	0.2156	0.3322	0.2189	0.3631	0.2235

1. Learning rate α, β: $\{1\times10^{-3}, 5\times10^{-3}, 1\times10^{-4}, 5\times10^{-4}, 5\times10^{-5}, 6\times10^{-5}, 7\times10^{-5}\}$.
2. Embedding Size: $\{32, 64, 128, 256\}$.
3. Epoch num N: $\{50, 20, 5, 1\}$.

The Recall@5 is used to choose the best hyperparameter on the validation set for testing.

5.3 Experiment Result and Discussion

This section contains experiment results of recommendation in the data-scarce scenario, the visualization of different components in the HyperRS model to show the capability of capturing user interests in item attributes, ablation experiment results for different support and query set sizes, and the ablation experiments for investigating the effects of different attributes.

5.3.1 Experiment Result of Recommendation in the Data-scarce Scenario

Table 5.2 to 5.7 present experiment results on Movielens-10m and TokyoTV datasets with NDCG@N and Recall@N, respectively. The best values in these tables are marked in bold font. The results on Book and Movielens-10m have two subgroups: C-C and C-W. C-C indicates the result is cold users with cold items, i.e., items in the validation

Table 5.3: Experiment results of Recall@N measures on Book dataset.

	Book							
	Recall@5		Recall@10		Recall@20		Recall@50	
	C-W	C-C	C-W	C-C	C-W	C-C	C-W	C-C
MeLU	0.0592	0.0276	0.2014	0.0314	0.2542	0.0381	0.3493	0.0552
s^2Meta	0.2873	0.1010	0.3021	0.1029	0.3127	0.1133	0.3415	0.1210
MetaTL	0.2479	0.0905	0.2796	0.1048	0.3092	0.1171	0.3514	0.1324
Mecos	0.2901	**0.2590**	0.3042	**0.2724**	0.3246	**0.2905**	0.3549	**0.3000**
HyperRS	**0.3077**	0.1752	**0.3852**	0.2514	**0.4345**	0.2590	**0.5261**	0.2724

Table 5.4: Experiment results of NDCG@N measures on Movielens-10m dataset.

	Movielens-10m							
	NDCG@5		NDCG@10		NDCG@20		NDCG@50	
	C-W	C-C	C-W	C-C	C-W	C-C	C-W	C-C
MeLU	0.2859	0.1540	0.3605	0.1615	0.4451	0.1811	0.5200	0.2232
s^2Meta	0.5884	0.4922	0.6782	0.5332	0.7056	0.5543	0.7389	0.5688
MetaTL	0.5657	0.4471	0.6507	0.4896	0.6618	0.4992	0.6749	0.5134
Mecos	0.4850	0.4633	0.6162	0.6377	0.6423	0.6493	0.6643	0.6571
HyperRS	**0.5893**	**0.4965**	**0.8834**	**0.7011**	**0.9026**	**0.7356**	**0.9108**	**0.7525**

and test sets do not appear in the training set. C-W indicates the results are cold users with warm items, i.e., items in the validation and test sets also appear in the training set. This setting examines the model capability on different item-side scenarios for cold users.

Experiment results in Table 5.2 and 5.3 indicate that the hypernetwork can generate appropriate parameters for the underlying recommender network in the cold users with warm items setting (C-W) on Book dataset. HyperRS has the best Recall@N performances in Table 5.3 with the C-W setting. It reflects that parameters in the recommender network capture user interests with adapted item attribute content embeddings. The lower NDCG@N (compared to Mecos) in Table 5.2 indicates that, although the items interacted by users appear in the top-N locations in the prediction, the positions of these items are not high. For example, 3, 4, or 5 in Recall@5, or 8, 9, or 10 in Recall@10, cf. Equation 5.10 and Equation 5.11. The performances on the C-C setting in Table 5.2 and Table 5.3 show that HyperRS outperforms all other baseline

Table 5.5: Experiment results of Recall@N measures on Movielens-10m dataset.

	Movielens-10m							
	Recall@5		Recall@10		Recall@20		Recall@50	
	C-W	C-C	C-W	C-C	C-W	C-C	C-W	C-C
MeLU	0.2002	0.0946	0.3074	0.1051	0.457	0.1404	0.6271	0.2382
s^2Meta	0.4366	0.3424	0.5588	0.3988	0.6082	0.4367	0.6847	0.4702
MetaTL	0.3844	0.2812	0.4923	0.3326	0.5109	0.3531	0.5389	0.3835
Mecos	0.3294	0.3224	0.5012	0.5529	0.5445	0.5723	0.5918	0.5890
HyperRS	**0.4533**	**0.3774**	**0.8699**	**0.6648**	**0.9037**	**0.7264**	**0.9223**	**0.7644**

models except for Mecos. The reason is that the number of item attribute contents in Book dataset is larger than other datasets, cf. Table 5.1, and the sizes of the support and query sets for Book dataset is smaller than other datasets (5 for the support and query sets for Book dataset whereas 10 for the support and query sets for Movielens-10m and Tokyo TV datasets.) Consequentially, the diverse attribute contents make it difficult for HyperRS to adapt item attribute contents that did not appear in the training set.

Results in Table 5.4 and Table 5.5 show that HyperRS outperforms all other baseline models, especially for cases that $N > 10$ in NDCG@N and Recall@N measures. The large NDCG@N measures for HyperRS indicate that the accurate recommended items (i.e., items in the user interests) are in front locations in the recommendations.

In contrast, MeLU's performance is not satisfying in all cases because of the lack of user profiles. s^2Meta and MetaTL that drop the user profile requirement perform better than MeLU.

In addition, all models are with deteriorated performances in the case of cold users with cold items because the adaption of the new item embeddings needs more optimization steps. Note that, for the cold users with cold items case, the HyperRS still outperforms other baselines. It indicates that the proposed method, HyperRS, has a fast item embedding adaption ability.

For the Tokyo TV dataset, because the number of users and items is not large enough, it does not support conducting the experiment with cold users and cold items. Therefore, the results in Table 5.6 and Table 5.7 are cold users with warm items.

The performances on Tokyo TV dataset are akin to those for the Movielens-10m dataset. In the case of small-scale users and items, Mecos has a better performance than

Table 5.6: Experiment results of NDCG@N measures on TokyoTV dataset.

	TokyoTV			
	NDCG@5	NDCG@10	NDCG@20	NDCG@50
MeLU	0.3371	0.4135	0.4582	0.5201
s^2Meta	0.4435	0.4804	0.5268	0.5406
MetaTL	0.6474	0.7880	0.8151	0.8151
Mecos	**0.6684**	0.9077	0.9260	**0.9412**
HyperRS	0.6285	**0.9158**	**0.9292**	0.9323

Table 5.7: Experiment results of Recall@N measures on TokyoTV dataset.

	TokyoTV			
	Recall@5	Recall@10	Recall@20	Recall@50
MeLU	0.2384	0.3461	0.4307	0.5692
s^2Meta	0.3076	0.3615	0.4461	0.4769
MetaTL	**0.5042**	0.7008	0.7521	0.7521
Mecos	0.4769	0.7923	0.8231	0.8538
HyperRS	0.4769	**0.8846**	**0.9077**	**0.9153**

HyperRS on NDCG@5 and NDCG@50 measures. MetaTL has a better performance than HyperRS on Recall@5.

The C-C results on the Movielens-10m and Book datasets indicate that HyperRS is more capable of capturing user interests with the parameters in the hypernetwork than adapting items with all new features. Because the item attribute contents in Book dataset are more diverse than those for the Movielens-10m dataset, the C-C setting contains more items with all new item attribute contents in the Book dataset. Note that the cold items are chosen by their item IDs rather than their attribute contents. The "Creator" and "Genre" attributes in Movielens-10m dataset are not as diverse as "Character." The contents in those two attributes can help the adaption of new Character contents by the residual block in the recommender network in HyperRS, cf. Equation 5.2 to Equation 5.5.

Additionally, the following research questions are investigated with more experiments.

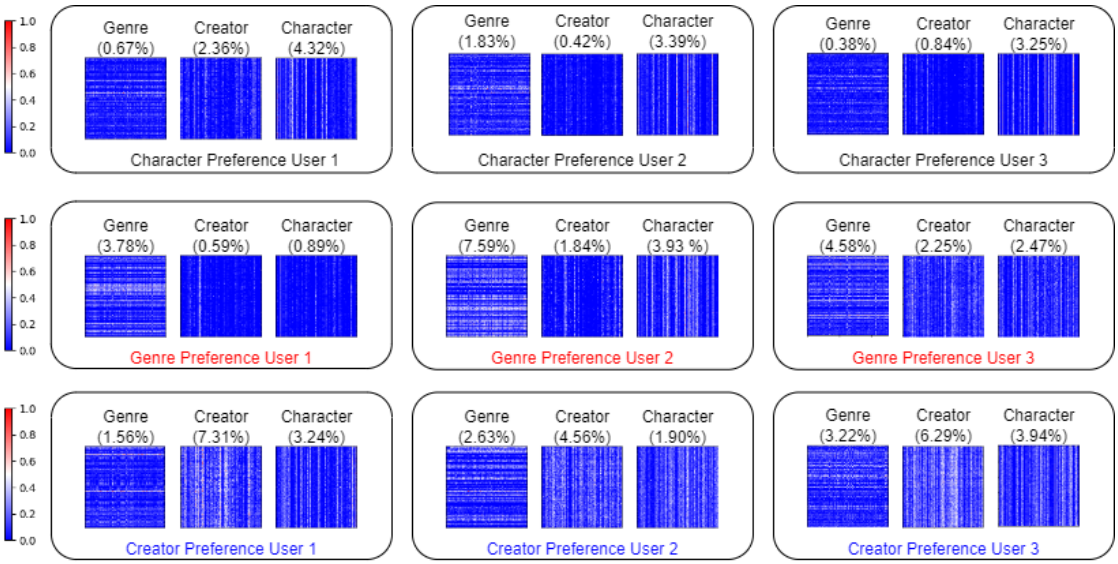


Figure 5.3: Heatmap of the weights that transforms item attribute content embeddings. The number shows the percent of elements that are larger than 0.5 after normalization. Different users are marked by colors and IDs.

5.3.2 Visualization of Components Capturing User Interests in Attributes

Research Question 1 - How does HyperRS capture user interest in item attributes? As mentioned at the beginning of Section 5.1, the user interest in item attributes is reflected by parameters in the recommender network. That is, *the parameters in the recommender network constitute the representations of user interest.* The matrix $\mathbf{W}_x$ in Equation 5.2 that transforms item attribute contents is visualized in Figure 5.3 to explain the effect.

There are three matrices that transform item attribute contents in the dataset: $\mathbf{W}_{\text{Genre}}$, $\mathbf{W}_{\text{Creator}}$, and $\mathbf{W}_{\text{Character}}$. The elements in these matrices are normalized to the range $[0, 1]$ before visualization. The numbers in Figure 5.3 are the ratio of elements larger than or equal to 0.5 in each matrix. Because the activation functions in the HyperRS are ReLU and LeakyReLU, more significant elements in these matrices indicate that the corresponding elements in $\mathbf{e}_x^{(i)}$ are amplified in $\hat{\mathbf{e}}_x^{(i)}$ (cf. Equation 5.2.) Therefore, more components in the item embeddings which are mixtures of item attribute embeddings are from the item attribute embeddings with the transformation

matrix. Consequently, users with $\mathbf{W}_{character}$ that have more prominent elements than the other two matrices can be categorized as character preference users. Similarly, the genre preference users and creator preference users follow the same categorization. Figure 5.3 presents three examples for each category (character preference, genre preference, and creator preference.)

5.3.3 Ablation Experiment Results for Different Support and Query Set Sizes

Research Question 2 - How does the sizes of the support and query sets affects model performance? The experiments with different sizes of support and query sets are conducted on the Movielens-10m dataset to investigate this question. Figure 5.4 presents the results under different sizes. Note that the hyperparameter setting is fine-tuned on the setting of $|S| = 10$ and $|Q| = 10$. Two different settings are $|S| = 5$, $|Q| = 15$ and $|S| = 15$ and $|Q| = 5$. All experiments are conducted by the cold users with warm items setting.

The increment of support size improves the performance measure NDCG@5 and Recall@5 for all models, but the other performance metrics decrease when the support set size increases. A probable reason is that the long sequential contains some irrelevant items in the support size to the items in the query size. Because of the optimization setting in meta-learning methods, the parameters in the recommender network are optimized over items in the support set, and the recommender network uses items in the query set for testing. These irrelevant items may mislead the recommender network. As a result, some items in I that are similar to the irrelevant items are ranked before the items in the query set in the lower positions of the prediction.

5.3.4 Ablation Experiment Results on the Effects of Attributes

In addition to the number of examples in the support and query sets, the ablation experiments are also conducted to investigate the effects of different attributes in HyperRS. The experiments use the best hyperparameter combinations in Section 5.3.1 and follow the cold user with the warm item setting.

The ablation experiment results are shown in Table 5.8, 5.9, 5.10, 5.11, 5.12, and 5.13

Figure 5.4: Results of all models with different sizes of support set and query set.

for the Movielens-10m, TokyoTV, and Book datasets, respectively. Results marked with "All" use all attributes and are from Table 5.4, 5.5, 5.6, 5.7, 5.2, and 5.3, "C-W" part for

Table 5.8: Ablation experiment NDCG@N results on Movielens-10m dataset. Improved measures are annotated by the bold font.

	NDCG@5	NDCG@10	NDCG@20	NDCG@50
All	0.5893	0.8834	0.9026	0.9108
Genre + Creator	0.5673	0.7986	0.8187	0.8335
Genre + Character	0.5720	0.8586	0.8797	0.8927
Creator + Character	**0.5983**	0.8464	0.8607	0.9058

Table 5.9: Ablation experiment Recall@N results on Movielens-10m dataset. Improved measures are annotated by the bold font.

	Recall@5	Recall@10	Recall@20	Recall@50
All	0.4553	0.8699	0.9037	0.9223
Genre + Creator	0.4461	0.7577	0.7939	0.8273
Genre + Character	0.4396	0.8458	0.8832	0.9120
Creator + Character	**0.4555**	0.8336	0.8891	**0.9295**

Table 5.10: Ablation experiment NDCG@N results on TokyoTV dataset.

	NDCG@5	NDCG@10	NDCG@20	NDCG@50
All	0.6285	0.9158	0.9292	0.9323
Genre + Creator	0.4831	0.6533	0.6855	0.7409
Genre + Character	0.5936	0.8838	0.8845	0.9183
Creator + Character	0.4873	0.6938	0.6959	0.7296

Table 5.11: Ablation experiment Recall@N results on TokyoTV dataset.

	Recall@5	Recall@10	Recall@20	Recall@50
All	0.4769	0.8846	0.9077	0.9153
Genre + Creator	0.3615	0.6000	0.6615	0.7846
Genre + Character	0.4462	0.8462	0.8538	0.8923
Creator + Character	0.3615	0.6462	0.6615	0.6923

each dataset, respectively.

Generally, the performances of the proposed model, HyperRS, deterioriate on Movielens-10m and TokyoTV datasets if it drops one attribute except for NDCG@5,

Table 5.12: Ablation experiment NDCG@N results on Book dataset. Improved measures are annotated by the bold font.

	NDCG@5	NDCG@10	NDCG@20	NDCG@50
All	0.2670	0.3113	0.3322	0.3631
Genre + Creator	**0.3171**	**0.3347**	**0.3438**	0.3495
Genre + Character	**0.3303**	**0.3543**	**0.3714**	**0.3956**
Creator + Character	**0.3159**	**0.3541**	**0.3817**	**0.4118**

Table 5.13: Ablation experiment Recall@N results on Book dataset.

	Recall@5	Recall@10	Recall@20	Recall@50
All	0.3077	0.3852	0.4345	0.5261
Genre + Creator	0.2915	0.3232	0.3444	0.3613
Genre + Character	0.2915	0.3345	0.3746	0.4458
Creator + Character	0.2768	0.3451	0.4099	0.5035

Recall@5 ,and Recall@50 (annotated with the bold font in Table 5.8) with the combination "Creator + Character" for Movielens-10m dataset, and the NDCG@N measures for the combination "Genre + Character" for Book dataset. Besides, the results for Movielens-10m dataset do not decay as severely as those for the TokyoTV dataset. For Book dataset, dropping attribute has a mixture effect - the NDCG@N measures are improved (except for NDCG@50 for the "Genre + Creator" combination) whereas the Recall@N measures decay.

One of the possible reasons is that, as shown in Table 5.1, Movielens-10m has more users than items compared to Book dataset, and it is with relatively diverse attribute contents compared to the TokyoTV dataset. These two factors help HyperRS avoid suffering severely from the insufficient number of attributes. In contrast, the combination of "Genre + Creator" decay badly on the TokyoTV dataset because the contents in these two attributes are too small to represent all user interests.

The Book dataset has fewer data in the support and query sets compared to the other two datasets (5 examples for the support and query sets whereas other datasets have 10) and it has more diverse attribute contents. As a result, it could be the case that the attribute content representations are not optimized sufficiently with the limited data. Therefore, dropping an item attribute helps improve the locations of

predicted items (reflected by the increasing NDCG@N) but fails to increase the number of correctly predicted items (reflected by the decreasing Recall@N.)

5.4 Conclusion

A hypernetwork-based recommender system for cold-start users, HyperRS, is proposed in this section. HyperRS has two neural networks as its components: a recommender network f_θ and another hypernetwork g_ϕ. The parameters θ in the recommender network are generated by the hypernetwork g, and these parameters are representations for a user's interest in the item attributes. f has item attribute content embedding transformation layers to capture the user's interest in item attribute contents, and the parameters in f capture the user's interest in item attirbutes themselves. Item attributes and contents are extracted from knowledge graphs. Different from MAML-based methods which learn a optimized initialization parameters for the underlying network, the proposed method generates all parameters in the underlying recommender network, and from the perspective of representation learning, the generated parameters for transferring item attribute contents constitute the representations of user interests in item attributes.

The hypernetwork takes a user interest embedding as a weight vector of the user interest basis (a group of vectors spanning the feature space for generating parameters in f.) and uses the weighted user interest basis with a parameter-specific, one-layer neural network to generate all parameters in f. The dynamically generated parameters in f ensure the fast adaption of these parameters for cold users.

The performance of the HyperRS is examined on three datasets, Movielens-10m, TokyoTV, and Book datasets. These three datasets represent different scales of the number of users and items. The baseline models contain a vanilla meta-learning recommender system that uses the MAML framework and other sequential meta-learning systems which do not require the user's profile. In HyperRS, users are represented by their embeddings that do not correspond to any demographic information. Therefore, it is as privacy-friendly as other sequential meta-learning systems for cold users.

The experiment results on both cold users with warm items and cold users with cold items scenarios show that the HyperRS outperforms all other baseline models.

These results indicate that the feature and parameter adaption is faster in the HyperRS. The experiment result on the TokyoTV dataset shows that the HyperRS performs better in most evaluation metrics compared to all baselines, even in the scenario where the number of users and items is small.

Two other experiments are conducted with the cold users and warm items setting on Movielens-10m to investigate the capability of representing user interest in item attributes by parameters in f and the effect of different sizes of support and query sets on the performance. In the first experiment, all matrices that transform item attribute content embeddings are normalized so that their elements are in $[0, 1]$. Then three examples are presented to explain the relation between user preference in item attributes and the scale of elements in these matrices. Because the activation functions in HyperRS are LeakyReLU and ReLU, those matrices that are with significant elements can amplify the corresponding elements in the item attribute content embeddings to the next layer. Therefore, they are indicators of user preference over item attributes. In the second experiment, the results show that HyperRS consistently outperforms other baseline models, although the support and query set size is changing.

6 Conclusion

This thesis investigates representation learning on knowledge graphs for two topics. The first topic is to learn type-aware representations in a unsupervised manner. There are two proposed methods for this topic. Both use type-agnostic knowledge graph completion models as their base model and calibrate entity and relation embeddings. Both methods use embeddings that represent certain locations in the feature space to encode the implicit type constraints in relations.

The first proposed method aims for the bilinear knowledge graph completion models. Its motivation is from the language model. It induces a loss function that calibrates entity embeddings by their global co-occurrences. Global co-occurrence is defined as entities that appear together at the same locations across all relations in the knowledge graph. The experiments on link prediction and entity clustering show that the proposed method is capable of capturing entity types and improves the link prediction performance for type-agnostic knowledge graph completion models.

The first method depends on entity co-occurrences, which is hard to count in large knowledge graphs. In addition, it is akin to many other implicit type-aware

approaches that use only a single embedding to represent the area where acceptable embeddings appear. The second proposed method uses multiple relation-specific embeddings to represent qualified entity types. As a result, it supports capturing multiple entity types within relations. Besides, the second proposed method (ProtoE) removes the dependency of global entity co-occurrence. This statistic is difficult to count for large knowledge graphs because it has to iterate over all triples to get the information. Comprehensive experiments were conducted on the same datasets used in the first proposed method on link prediction and entity clustering. Similar to the first proposed method, the ProtoE is capable of capturing diverse implicit entity types within relations and improves the link prediction performance. In ablation experiments, the effect of the number of relation-specific embeddings (the prototype embeddings) that represent acceptable entity types on link prediction performance is examined. The entity clustering results are visualized to explain the clusters constituted entities with different types and the prototype embeddings that play as representative points for them. Unlike the first proposed method, which only supports bilinear knowledge graph completion models, the ProtoE can also be adapted to translational knowledge graph completion models.

In the current setting, the number of prototype embeddings in ProtoE is a hyperparameter. The ablation experiment shows that this hyperparameter affects the link prediction performance. The effect is significant for the translational models. Therefore, it is preferable to adopt some stochastic processes, such as the Chinese restaurant process or the determinantal point process, to help automatically or semi-automatically determine this hyperparameter.

In the second topic, the research in this thesis focuses on the application of representation learning on meta-learning for the user-side cold start problem with the knowledge graph. The proposed method, HyperRS, depends on features extracted from knowledge graphs to capture user interest with little data. It is a hypernetwork-based method that takes several vectors to span the entire feature space of the parameters used in the recommender network. It is akin to the model-agnostic meta-learning (MAML)framework in that the underlying neural network uses user-specific parameters to capture user interests rapidly. Unlike the MAML framework-based methods, HyperRS generates the initial parameters for the recommender network by taking a user embedding that is a mixture of user interest basis used for parameter

generation. The experiments use the Movielens 10m dataset and the Tokyo TV dataset. The former contains many users and items, whereas the latter only includes a few users and items. Experiment results on a different scale of users and items show that HyperRS outperforms other MAML-based baselines. The ablation experiment changes the sizes of support and query sets to examine the effect on all models. It turns out that HyperRS has the best performance for all different size settings. In addition, the visualization of matrices that transform item attribute content embeddings shows that HyperRS can capture user interests in item attributes themselves.

Those two tasks constitute a pipeline as shown in Figure 1.1 at the beginning of Section 1.1. Task 1 concerns link prediction. The proposed methods learn implicit type constraints to improve the performance of existing type-agnostic KGC models on link prediction. These proposed methods can be used for adding new facts according to existing ones in a knowledge graph, and the knowledge graph can play the role of feature provider in task 2, the meta-learning for the user-side cold-start problem, by storing the facts enriched by the models proposed in task 1. Effective methods for adapting facts by learning their distributed representations for these two tasks are investigated in the thesis, and the experiment results show that all proposed methods can learn or adapt the facts in a knowledge graph in a distributed representation form for the given tasks.

Bibliography

[1] P. Jain, P. Kumar, S. Chakrabarti, *et al.*, "Type-sensitive knowledge base inference without explicit type supervision," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 75–80, 2018.

[2] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 1126–1135, 2017.

[3] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[4] Z. Liu, Y. Lin, and M. Sun, *Representation learning for natural language processing*. Springer Nature, 2020.

[5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, vol. 26, pp. 1–9, 2013.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 4171–4186, Association for Computational Linguistics, June 2019.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[8] H. Wang, F. Zhang, J. Wang, M. Zhao, W. Li, X. Xie, and M. Guo, "Ripplenet: Propagating user preferences on the knowledge graph for recommender systems," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, p. 417–426, Association for Computing Machinery, 2018.

[9] Y. Zhang, H. Dai, Z. Kozareva, A. J. Smola, and L. Song, "Variational reasoning for question answering with knowledge graph," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pp. 6069–6076, 2018.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[11] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," in *Proceedings of the International Conference on Learning Representations 2017*, 2017.

[12] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems*, p. 2787–2795, 2013.

[13] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pp. 687–696, 2015.

[14] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pp. 1112–1119, 2014.

[15] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, p. 2181–2187, 2015.

[16] S. Zhiqing, D. Zhi-Hong, and J. T. Jian-Yun, Nie and, "Rotate: Knowledge graph embedding by relational rotation in complex space," in *Proceedings of the International Conference on Learning Representations 2019*, 2019.

[17] Y. Bishan, Y. Wen-tau, H. Xiaodong, G. Jianfeng, and D. Li, "Embedding entities and relations for learning and inference in knowledge bases," in *Proceedings of the International Conference on Learning Representations 2015*.

[18] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, p. 809–816, 2011.

[19] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, p. 2071–2080, 2016.

[20] H. Liu, Y. Wu, and Y. Yang, "Analogical inference for multi-relational embeddings," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, p. 2168–2178, 2017.

[21] I. Balazevic, C. Allen, and T. Hospedales, "TuckER: Tensor factorization for knowledge graph completion," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pp. 5185–5194, 2019.

[22] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pp. 1811–1818, 2018.

[23] S. M. Kazemi and D. Poole, "Simple embedding for link prediction in knowledge graphs," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, p. 4289–4300, 2018.

[24] R. Xie, Z. Liu, and M. Sun, "Representation learning of knowledge graphs with hierarchical types," in *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, p. 2965–2971, 2016.

[25] Y. Zhao, A. Zhang, R. Xie, K. Liu, and X. Wang, "Connecting embeddings for knowledge graph entity typing," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6419–6428, 2020.

[26] Z. Cui, P. Kapanipathi, K. Talamadupula, T. Gao, and Q. Ji, "Type-augmented relation prediction in knowledge graphs," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pp. 7151–7159, 2021.

[27] C. Moon, P. Jones, and N. F. Samatova, "Learning entity type embeddings for knowledge graph completion," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, p. 2215–2218, 2017.

[28] D. Krompaß, S. Baier, and V. Tresp, "Type-constrained representation learning in knowledge graphs," in *Proceedings of the 14th International Semantic Web Conference*, p. 640–655, 2015.

[29] G. Niu, B. Li, Y. Zhang, S. Pu, and J. Li, "AutoETER: Automated entity type representation for knowledge graph embedding," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1172–1181, 2020.

[30] X. L. Dong, X. He, A. Kan, X. Li, Y. Liang, J. Ma, Y. E. Xu, C. Zhang, T. Zhao, G. Blanco Saldana, S. Deshpande, A. Michetti Manduca, J. Ren, S. P. Singh, F. Xiao, H.-S. Chang, G. Karamanolakis, Y. Mao, Y. Wang, C. Faloutsos, A. McCallum, and J. Han, "Autoknow: Self-driving knowledge collection for products of thousands of types," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, p. 2724–2734, Association for Computing Machinery, 2020.

[31] W. Zhang, C.-M. Wong, G. Ye, B. Wen, W. Zhang, and H. Chen, "Billion-scale pre-trained e-commerce product knowledge graph model," in *2021 IEEE 37th International Conference on Data Engineering*, pp. 2476–2487, 2021.

[32] Q. Ai, V. Azizi, X. Chen, and Y. Zhang, "Learning heterogeneous knowledge base embeddings for explainable recommendation," *Algorithms*, vol. 11, no. 9, pp. 1–17, 2018.

[33] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, "Kgat: Knowledge graph attention network for recommendation," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, p. 950–958, Association for Computing Machinery, 2019.

[34] M. Dong, F. Yuan, L. Yao, X. Xu, and L. Zhu, "Mamo: Memory-augmented meta-optimization for cold-start recommendation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 688–697, 2020.

[35] X. Feng, C. Chen, D. Li, M. Zhao, J. Hao, and J. Wang, "Cmml: Contextual modulation meta learning for cold-start recommendation," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 484–493, 2021.

[36] Y. Lu, Y. Fang, and C. Shi, "Meta-learning on heterogeneous information networks for cold-start recommendation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1563–1573, 2020.

[37] Y. Sun, K. Yin, H. Liu, S. Li, Y. Xu, and J. Guo, "Meta-learned specific scenario interest network for user preference prediction," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1970–1974, 2021.

[38] H. Lee, J. Im, S. Jang, H. Cho, and S. Chung, "Melu: Meta-learned user preference estimator for cold-start recommendation," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1073–1082, 2019.

[39] H. Sun, J. Xu, K. Zheng, P. Zhao, P. Chao, and X. Zhou, "Mfnp: A meta-optimized model for few-shot next poi recommendation," in *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pp. 3017–3023, 2021.

[40] J. Wang, K. Ding, and J. Caverlee, "Sequential recommendation for cold-start users with meta transitional learning," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1783–1787, 2021.

[41] R. Yu, Y. Gong, X. He, B. An, Y. Zhu, Q. Liu, and W. Ou, "Personalized adaptive meta learning for cold-start user preference prediction," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pp. 10772–10780, 2021.

[42] Y. Chen, X. Wang, M. Fan, J. Huang, S. Yang, and W. Zhu, "Curriculum meta-learning for next poi recommendation," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2692–2702, 2021.

[43] H. Bharadhwaj, "Meta-learning for user cold-start recommendation," in *Proceedings of the 2019 International Joint Conference on Neural Networks*, pp. 1–8, 2019.

[44] F. Tahmasebi, M. Meghdadi, S. Ahmadian, and K. Valiallahi, "A hybrid recommendation system based on profile expansion technique to alleviate cold start problem," *Multimedia Tools and Applications*, vol. 80, no. 2, pp. 2339–2354, 2021.

[45] S. Ahmadian, M. Afsharchi, and M. Meghdadi, "A novel approach based on multi-view reliability measures to alleviate data sparsity in recommender systems," *Multimedia tools and applications*, vol. 78, no. 13, pp. 17763–17798, 2019.

[46] L. Wang, B. Jin, Z. Huang, H. Zhao, D. Lian, Q. Liu, and E. Chen, "Preference-adaptive meta-learning for cold-start recommendation," in *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pp. 1607–1614, 2021.

[47] Y. Zhang, D. Z. Cheng, T. Yao, X. Yi, L. Hong, and E. H. Chi, "A model of two tales: Dual transfer learning framework for improved long-tail item recommendation," in *Proceedings of the Web Conference*, pp. 2220–2231, 2021.

[48] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals, "Rapid learning or feature reuse? towards understanding the effectiveness of maml," in *Proceedings of the International Conference on Learning Representations*, 2020.

[49] P. Savarese and M. Maire, "Learning implicitly recurrent cnns through parameter sharing," in *Proceedings of the International Conference on Learning Representations 2019*, 2019.

[50] A. Shamsian, A. Navon, E. Fetaya, and G. Chechik, "Personalized federated learning using hypernetworks," in *Proceedings of the 38th International Conference on Machine Learning*, pp. 9489–9502, 2021.

[51] R. Bensadoun, S. Gur, T. Galanti, and L. Wolf, "Meta internal learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20645–20656, 2021.

[52] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs.," in *Proceedings of the Semantics Conference*, pp. 1–2, 2016.

[53] G. A. Miller, "Wordnet: A lexical database for english," *ACM Communication*, p. 39–41, nov 1995.

[54] S. Zheng, J. Rao, Y. Song, J. Zhang, X. Xiao, E. F. Fang, Y. Yang, and Z. Niu, "PharmKG: a dedicated knowledge graph benchmark for bomedical data mining," *Briefings in Bioinformatics*, vol. 22, pp. 1–4, 12 2020.

[55] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, p. 1247–1250, 2008.

[56] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," in *Relational data mining*, pp. 307–335, Springer, 2001.

[57] B. W. Bader, R. A. Harshman, and T. G. Kolda, "Temporal analysis of semantic graphs using asalsan," in *Seventh IEEE international conference on data mining (ICDM 2007)*, pp. 33–42, IEEE, 2007.

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

[59] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, 1927.

[60] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, p. 3111–3119, 2013.

[61] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Empirical Methods in Natural Language Processing*, pp. 1532–1543, 2014.

[62] Z. Li, Q. Xu, Y. Jiang, X. Cao, and Q. Huang, "Quaternion-based knowledge graph network for recommendation," in *Proceedings of the 28th ACM International Conference on Multimedia*, p. 880–888, 2020.

[63] T. Ebisu and R. Ichise, "Generalized translation-based embedding of knowledge graph," *IEEE Transactions on Knowledge and Data Engineering*, pp. 941–951, 2020.

[64] M. Nickel, L. Rosasco, and T. Poggio, "Holographic embeddings of knowledge graphs," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, p. 1955–1961, 2016.

[65] K. Hayashi and M. Shimbo, "On the equivalence of holographic and complex embeddings for link prediction," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 554–559, 2017.

[66] F. Che, D. Zhang, J. Tao, M. Niu, and B. Zhao, "Parame: Regarding neural network parameters as relation embeddings for knowledge graph completion," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2774–2781, 2020.

[67] D. Brickley, R. V. Guha, and B. McBride, "Rdf schema 1.1," *W3C recommendation*, vol. 25, pp. 2004–2014, 2014.

[68] A. Zouaq and F. Martel, "What is the schema of your knowledge graph? leveraging knowledge graph embeddings and clustering for expressive taxonomy learning," in *Proceedings of The International Workshop on Semantic Big Data*, SBD '20, (New York, NY, USA), pp. 1–6, Association for Computing Machinery, 2020.

[69] X. Wang, D. Wang, C. Xu, X. He, Y. Cao, and T.-S. Chua, "Explainable reasoning over knowledge graphs for recommendation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 5329–5336, 2019.

[70] H. Wang, F. Zhang, J. Wang, M. Zhao, W. Li, X. Xie, and M. Guo, "Exploring high-order user preference on the knowledge graph for recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol. 37, no. 3, pp. 1–26, 2019.

[71] H. Wang, M. Zhao, X. Xie, W. Li, and M. Guo, "Knowledge graph convolutional networks for recommender systems," in *The world wide web conference*, pp. 3307–3313, 2019.

[72] W. Ma, M. Zhang, Y. Cao, W. Jin, C. Wang, Y. Liu, S. Ma, and X. Ren, "Jointly learning explainable rules for recommendation with knowledge graph," in *The world wide web conference*, pp. 1210–1221, 2019.

[73] X. Fu, J. Zhang, Z. Meng, and I. King, "Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding," in *Proceedings of The Web Conference 2020*, WWW '20, (New York, NY, USA), p. 2331–2341, Association for Computing Machinery, 2020.

[74] Y. Dong, Z. Hu, K. Wang, Y. Sun, and J. Tang, "Heterogeneous network representation learning," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20* (C. Bessiere, ed.), pp. 4861–4867, International Joint Conferences on Artificial Intelligence Organization, 7 2020. Survey track.

[75] Z. Du, X. Wang, H. Yang, J. Zhou, and J. Tang, "Sequential scenario-specific meta learner for online recommendation," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 2895–2904, 2019.

[76] Y. Bi, L. Song, M. Yao, Z. Wu, J. Wang, and J. Xiao, "Dcdir: A deep cross-domain recommendation system for cold start users in insurance domain," in *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pp. 1661–1664, 2020.

[77] A. Sankar, J. Wang, A. Krishnan, and H. Sundaram, "Protocf: Prototypical collaborative filtering for few-shot recommendation," in *Proceedings of the 15th ACM Conference on Recommender Systems*, pp. 166–175, 2021.

[78] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proceedings of the 33rd International Conference on Machine Learning*, pp. 1842–1850, 2016.

[79] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, pp. 2121–2159, 2011.

[80] C. J. Hoofnagle, B. van der Sloot, and F. Z. Borgesius, "The european union general data protection regulation: what it is and what it means," *Information & Communications Technology Law*, vol. 28, no. 1, pp. 65–98, 2019.

[81] Y. Zhu, R. Xie, F. Zhuang, K. Ge, Y. Sun, X. Zhang, L. Lin, and J. Cao, "Learning to warm up cold item embeddings for cold-start recommendation with meta scaling and shifting networks," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1167–1176, 2021.

[82] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Language*, vol. 8, p. 229–256, may 1992.

[83] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson, "Fast context adaptation via meta-learning," in *International Conference on Machine Learning*, pp. 7693–7702, PMLR, 2019.

[84] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, p. 4080–4090, 2017.

[85] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, "Meta-learning with latent embedding optimization," in *Proceedings of the International Conference on Learning Representations 2018*, 2018.

[86] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks," *Artificial Life*, vol. 15, pp. 185–212, 04 2009.

[87] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[88] D. Zhao, J. von Oswald, S. Kobayashi, J. Sacramento, and B. F. Grewe, "Meta-learning via hypernetworks," in *4th Workshop on Meta-Learning at NeurIPS 2020*, pp. 296–304, 2020.

[89] A. Lamb, S. Evgeny, L. Yingzhen, T. Sebastian, L. Camilla, W. Simon, H.-L. José Miguel, T. Richard E., C. Pashmina, and Z. Cheng, "Contextual hypernetworks for novel feature adaptation," in *4th Workshop on Meta-Learning at NeurIPS 2020*, pp. 616–627, 2020.

[90] Y. Lu and R. Ichise, "Unsupervised type constraint inference in bilinear knowledge graph completion models," in *2021 IEEE International Conference on Big Knowledge*, pp. 15–22, 2021.

[91] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proceedings of the 16th International Conference on World Wide Web*, p. 697–706, 2007.

[92] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations 2015*, 2015.

[93] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.

[94] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, pp. 2579–2605, 2008.

[95] Y. Lu and R. Ichise, "Protoe: Enhancing knowledge graph completion models with unsupervised type representation learning," *Information*, vol. 13, no. 8, pp. 354–379, 2022.

[96] T. Madushanka and R. Ichise, "Mdncaching: A strategy to generate quality negatives for knowledge graph embedding," in *Proceedings of International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 877–888, 2022.

[97] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[98] D. M. Blei, T. L. Griffiths, and M. I. Jordan, "The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies," *Journal of the ACM*, vol. 57, no. 2, pp. 1–30, 2010.

[99] T. Griffiths, M. Jordan, J. Tenenbaum, and D. Blei, "Hierarchical topic models and the nested chinese restaurant process," *Advances in neural information processing systems*, vol. 16, pp. 1–8, 2003.

[100] A. Kulesza and B. Taskar, "Structured determinantal point processes," *Advances in neural information processing systems*, vol. 23, pp. 1–9, 2010.

[101] A. Kulesza, B. Taskar, *et al.*, "Determinantal point processes for machine learning," *Foundations and Trends® in Machine Learning*, vol. 5, no. 2–3, pp. 123–286, 2012.

[102] S. Wu, S. Zhong, and Y. Liu, "Deep residual learning for image steganalysis," *Multimedia Tools Application.*, vol. 77, p. 10437–10453, may 2018.

[103] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, "Meta-learning with latent embedding optimization," in *Proceedings of the International Conference on Learning Representations 2019*, 2019.

[104] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transaction on Interactive Intelligence Systems*, vol. 5, pp. 1–19, dec 2015.

[105] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *Proceedings of the 14th international conference on World Wide Web*, pp. 22–32, 2005.

[106] Y. Zheng, S. Liu, Z. Li, and S. Wu, "Cold-start sequential recommendation via meta learner," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 4706–4713, 2021.

[107] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

A

Appendix

The case studies with entity clustering on the ProtoE's ability to capture implicit type constraints are listed as follows.

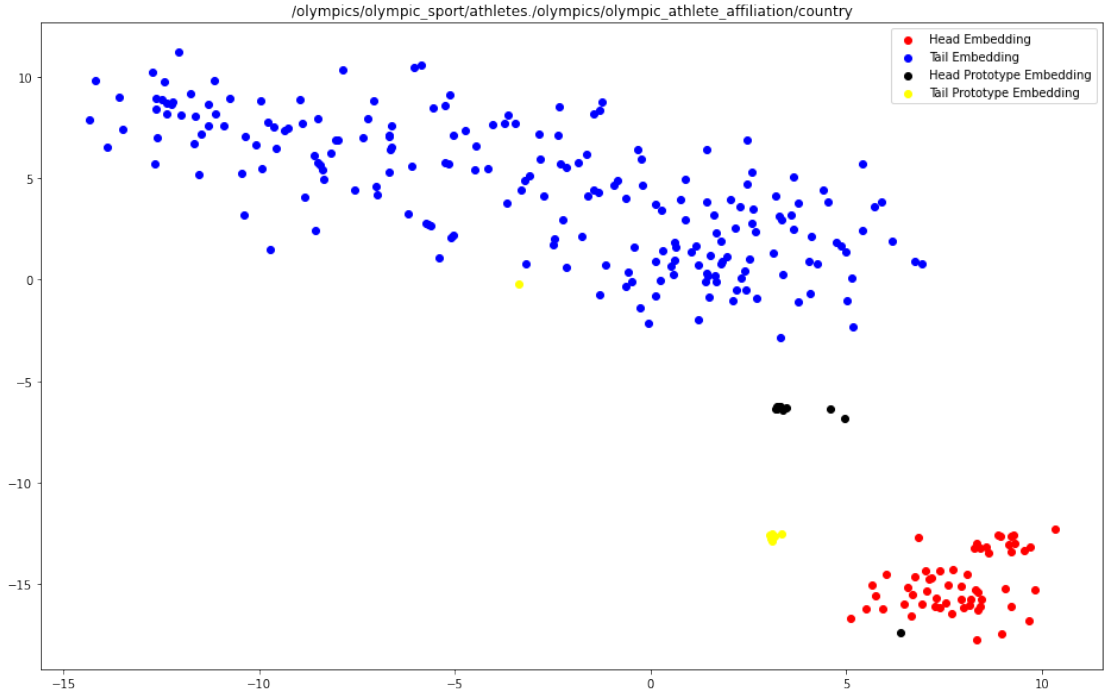


Figure A.1: A total of 57 different head entities, 209 different tail entities, and 154 records in the training data.

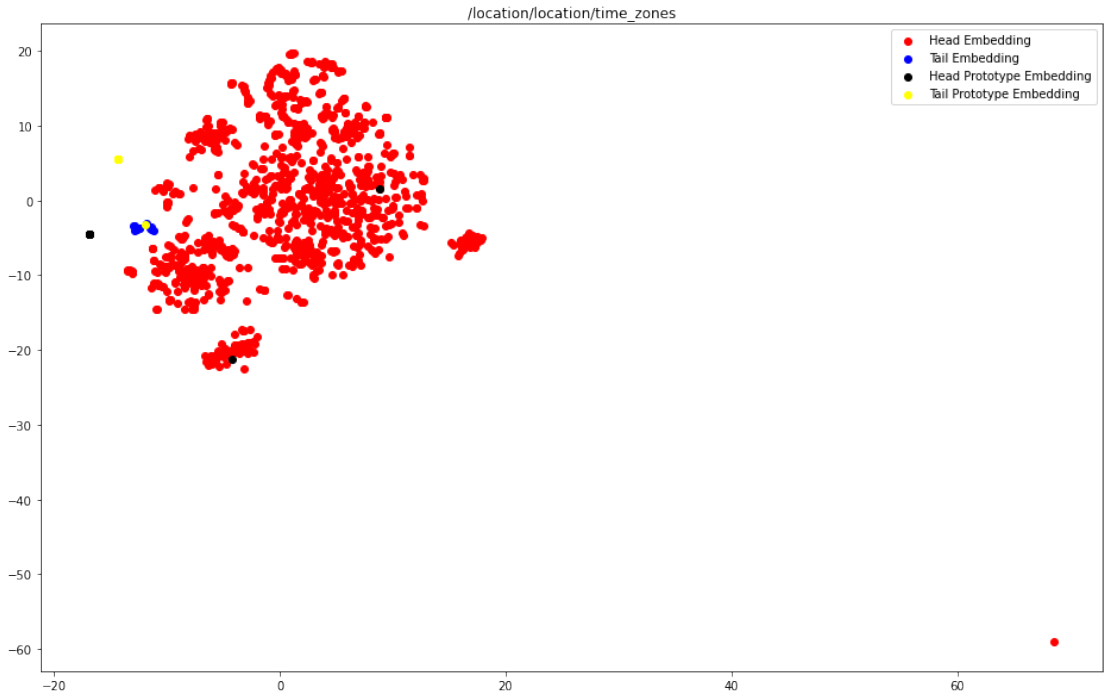


Figure A.2: A total of 1,127 different head entities, 13 different tail entities, and 1151 records in the training data.

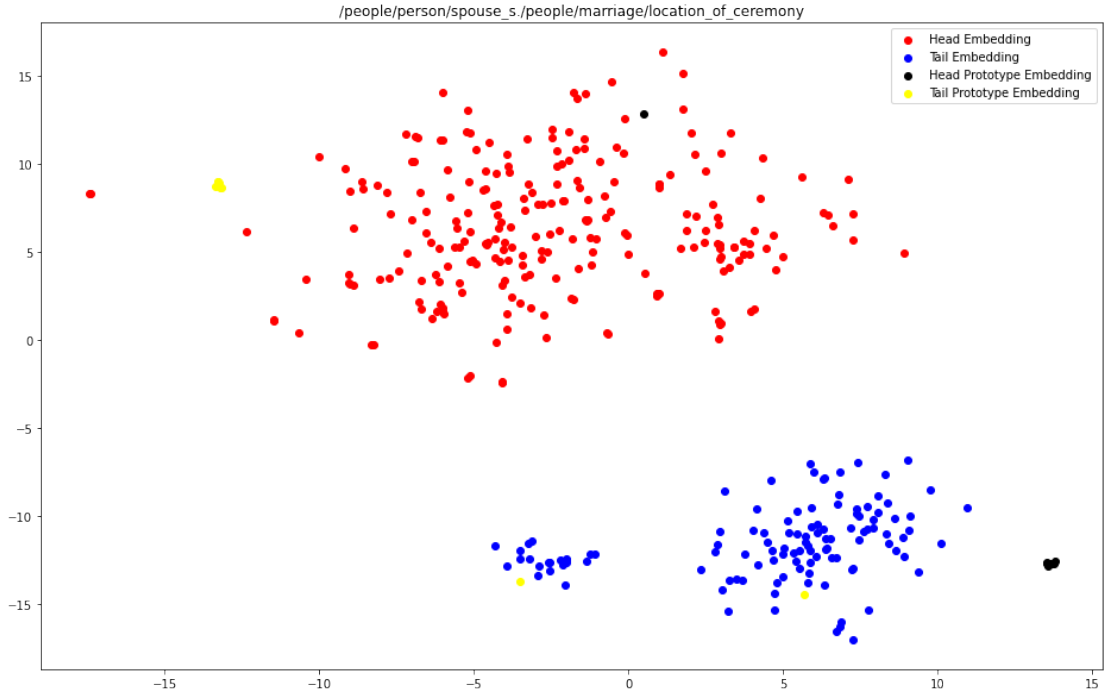


Figure A.3: A total of 235 different head entities, 116 different tail entities, and 251 records in the training data.

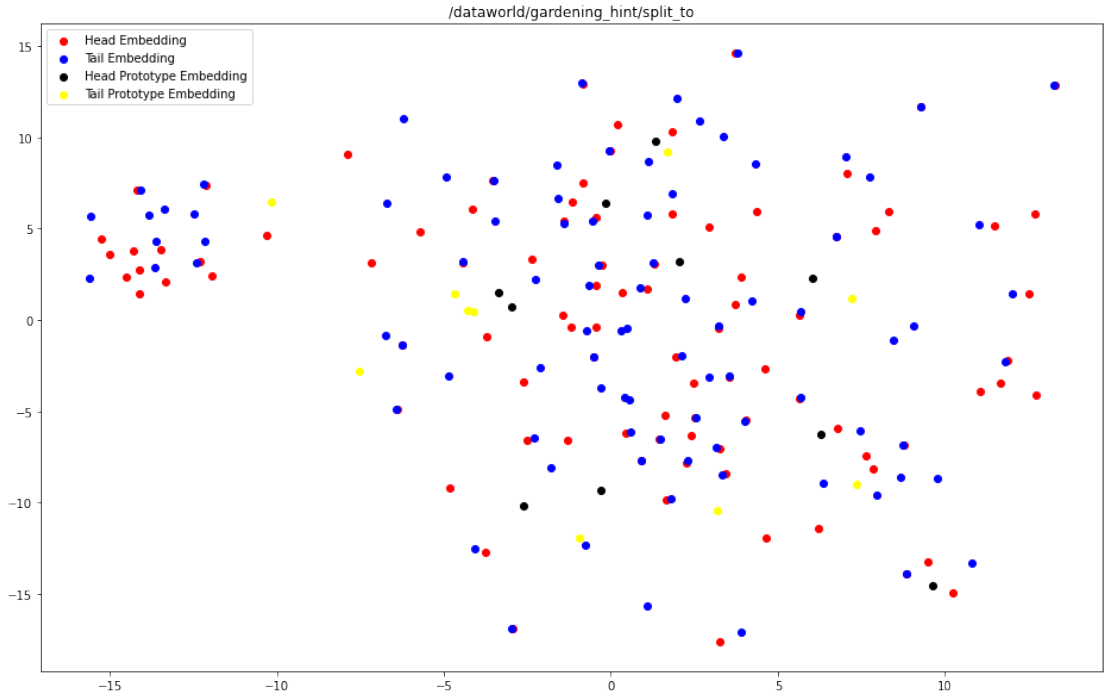


Figure A.4: A total of 93 different head entities, 90 different tail entities, and 99 records in the training data. Some of the head entities and tail entities overlap because the relation is symmetric.