Learning Latent Features using Stochastic Neural Networks on Graph Structured Data

Zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der

KIT-Fakultät für Wirtschaftswissenschaften des Karlsruher Instituts für Technologie (KIT)

genehmigte

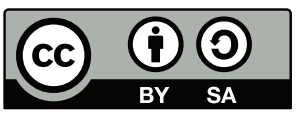
Dissertation

von

M.Sc. Tobias Weller

Referent:	Prof. Dr. York Sure-Vetter
Korreferentin:	Jun. Prof. Dr. Maribel Acosta Deibe

Tag der mündlichen Prüfung: 11. Februar 2021



This thesis is dedicated to my beloved parents
Edgar and Heiderose Weller

ABSTRACT

Graph structured data are ubiquitous data structures, used to model relationships between entities. Graphs have become an important foundation to represent interactions between users in social networks, items in recommender systems, and interactions between drugs in bioinformatics. The main research problems in these areas include node clustering, node classification and link prediction. Especially the link prediction task is in bioinformatics of special interest toward the identification and development of new uses of existing or abandoned drugs since drug development is currently very time consuming and expensive. In the context of knowledge graphs, link prediction is also of special interest to automatically complete missing information to derive further knowledge. Likewise, node classification is an important research focus in the context of knowledge graphs, e.g. to automatically classify new entities according to their class affiliation and to complete missing class affiliation for existing entities.

In recent years, network embeddings are often trained for encoding the entities of graph structured data into a low-dimensional space whilst preserving the graph structure. Based on the trained embeddings, machine learning techniques are applied to address the main machine learning tasks, such as link prediction and node classification. In most of the published methods, like e.g. RDF2Vec, DeepWalk, node2vec and LINE, random walks procedures are used to efficiently explore diverse neighbourhoods and compute embeddings based on them. However, these methods develop their full potential only when the input graph is connected, otherwise the random walks are not sufficient to gather enough information about nodes in the neighborhoods, as not all nodes in the graph can be reached.

In this work we address three types of problems: Link prediction on bipartite networks, link prediction on knowledge graphs and a semantic grouping of nodes and links in graphs. We use a stochastic factorization model to learn a target distribution over the graph structured data, allowing to predict unknown links and embed the nodes into a low-dimensional space whilst preserving the distribution of interactions within the graph. The embeddings are used in a following step to learn a function for predicting instance types and domain assertions using training data. Compared to the existing methods that use random walks, our approach is much more robust with respect to the connectivity of the graph structured data. Results show that the proposed method outperforms current state-of-the-art models in several studied graph structured data and sets a new baseline in link prediction on disconnected graph structured data and grouping of nodes and links.

CONTENTS

1

INTRODUCTION

Graph structured data are ubiquitous data structures, used to model relationships between entities. Graphs have become an important foundation to represent interactions between users in social networks [60, 98, 101], items in recommender systems [1, 57, 86], and interactions between drugs in bioinformatics [62, 121, 124]. Graphs in their simplest form consist of nodes, which represent entities like users in a social network and edges, which represent interactions between these users. Furthermore, there are graphs with special characteristics such as bipartite graphs, which assign nodes to two different types, whereby only edges between nodes of different types are valid and not between nodes of the same type. We define graphs in general as follows.

Definition 1: Graph

A graph G is an ordered pair (V, E), where V denotes a set of nodes and E denotes a set of edges.

Based on the concept of graphs, information can be encoded in graphs like knowledge in so-called knowledge graphs. Knowledge Graphs (KGs) are directional labelled graphs, used to model information. Consider Figure 1 in which knowledge about different entities – represented as nodes, e.g. *Angela Merkel* and their relations – represented as directional edges, e.g. *married_to* is encoded based on information from the real world.

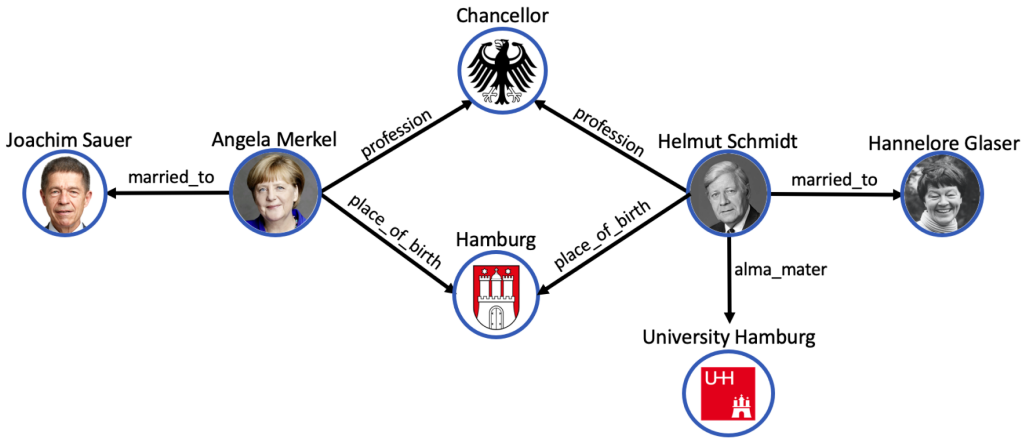


Figure 1: Exemplary knowledge graph in which facts from the real world are encoded.

Knowledge graphs, as shown for example in Figure 1, are used in many companies like Google[1] and Facebook[2] to enhance and improve the offered services. However, knowledge graphs are often not complete with regard to their encoded information. A complete knowledge graph or a graph with complete information is crucial for an effective performance of the services like question-answering as offered by Google [112]. In previous work, random walk procedures were often used to encode information from graph structured data to learn a function for completing missing information. The random walk methods are however sensitive regarding the topology of the graphs. In contrast, stochastic methods are robust regarding the topology of the graph structured data, since they do not depend on random walks and thus do not remain fixed in a local topology. For this reason, we propose an approach for learning latent features using a stochastic factorization method to complete missing information in graph structured data. To motivate this work in terms of learning latent features for completing the information encoded in a graph, we will discuss the motivation in more detail in the next section.

1.1 MOTIVATION

Consider again Figure 1, representing facts about different entities. It is conceivable that similar facts are also represented in the Google Knowledge Graph, which is used by Google to improve question answering and information retrieval services. Whenever a Google user enters a question into the search mask, the available information from the knowledge graph is used to find the answer and helpful links to this search query. When assuming a user enters the following query in Google *'What is the alma mater of Angela Merkel'*, then Google would try to answer this query based on the knowledge graph. Taking a closer look at the knowledge graph in Figure 1 we notice that the information at which university Angela Merkel graduated is missing and therefore Google cannot answer the query based on this knowledge graph. A complete knowledge graph, with information about the university at which Angela Merkel graduated, could answer this query. Using this example, we can see the value of a complete knowledge graph for providing services such as question answering or information retrieval. A complete knowledge graph supports these services and gives an advantage over competitors, which in turn leads to higher success in the market. Google, for example, has achieved a total revenue of $161,857 millions in 2019, which is in part due to the very successful services offered, using knowledge graphs for support.[3]

However, as we have seen in the above example, knowledge graphs are often not complete, either because they were created in an automatic or pay-as-you-go KG construction process. Therefore, information might be incomplete or missing for some entities and properties in the knowledge graph. Likewise, a large amount of new information is created, making it hard to keep up with the newly gained information from the real world and encode it in a knowledge

[1] https://blog.google/products/search/introducing-knowledge-graph-things-not/
[2] https://techcrunch.com/2013/01/15/facebook-announces-its-third-pillar-graph-search/
[3] https://abc.xyz/investor/static/pdf/2019Q4_alphabet_earnings_release.pdf

graph. For example, 215,800 new articles were created on Wikipedia(s) alone in 2019[4]. It is not possible to manually encode this huge amount of information in a knowledge graph. Therefore a process is needed, which determines incomplete facts or missing knowledge in a knowledge graph automatically. Besides, other information can be encoded in graph structured data as described above, e.g. drug-target interactions. The information encoded in these graphs can be used to predict whether existing drugs can be used to target new diseases. Currently drug development of a new drug usually takes up to 10 years and are very costly, making it highly desirable to reuse existing drugs. Reusing existing drugs for new targets allows for reducing the time until the market introduction (since clinical trials and regulatory requirements already exist for the existing drug). Similar to knowledge graphs, one is interested in determining missing interactions in the graph and to complete them if necessary.

1.2 PROBLEM STATEMENT

Based on the motivation and the example given in the previous section, we learned that complete information is crucial for providing effective services to gain a market advantage. For this reason we aim at completing information in graph structured data in this thesis. Furthermore we would like to restrict the existing problem for completing information in graph structured data even further and focus on the problem for completing the links. Often entities, which are usually represented by nodes in graphs, are not included in the knowledge graph, making it difficult to create missing links to them automatically. However, the knowledge of missing links might be sufficient to create advanced queries even without the information about the corresponding entity. Consider the example in Figure 1 and the query 'List all entities that graduated from a university'. Based on the original knowledge graph only *Helmut Schmidt* is part of the solution set of this query. However, as already described above, *Angela Merkel* studied at the University of Leipzig. This information is not included, however, if we could only infer that she has the relation *almaMater*, even without the information of the object, we could extend the solution set and return *Angela Merkel*, besides *Helmut Schmidt*, as part of the solution set.

The above example shows to what extent we want to predict missing links in knowledge graphs. In this work we do not want to focus exclusively on knowledge graphs, but develop methods that are generally applicable to graph structured data. Therefore, we define the overall problem of this thesis as follows.

↯ Problem 1

Given is a Graph $G = (V, E)$, and two nodes $u, v \in V$, predict whether u and v should be connected in the graph G, i.e.,$(u, v) \in E$.

[4] https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

1.3 CHALLENGES

The challenges of the formulated problem for predicting links in graph structured data can be grouped into different categories, which we will discuss in detail.

Topology. Graphs are abstract structures that represent a set of entities along with the connections existing between these entities. The abstract definition of graphs allows different specifications to represent different features. There are so-called directed graphs without multi-edges in which the set of edges E consists of a subset of the cartesian product of the nodes, i.e. $E \subseteq V \times V$ and are visualized by a directed edge. Besides, directed graphs with multiple edges allow for modeling different directed relations between nodes. This form of graphs is especially used for knowledge graphs as shown in Figure 1. A node can have different labeled edges to nodes, encoding different information of the entities, which are represented by nodes in the graph. Another very common type of graphs are undirected graphs without multi-edges, which are often used to represent network relationships. These are used, for example, to represent friendship relationships in social networks or to represent untyped interactions of drug-target interactions in biomedical networks, since the type of relationship is always the same and the relationship is symmetrical, i.e. both an edge $(u, v) \in E$ and $(v, u) \in E$ exist. The different graph-based structures mentioned above are only a small fraction of the possible variations, but demonstrate the multitude of possible variations, which makes it difficult to develop a method that can be applied to all the different variations of graphs.

Another important aspect in graphs is the connectivity, or the related number of available connections. The term density is often used in this context, which defines the number of connections in relation to the number of nodes. The denser a graph, the more relationships exist in it and the more information is available. However, there are also graphs in which only very few edges or relations between the nodes exist, which are called sparse graphs. The difficulty in predicting links in sparse graphs is to gather enough information from the existing links in the graph to predict new links. Furthermore, the distribution of the connections in the graph may not be equally distributed, but focused on a few nodes, causing the degree of the nodes to vary greatly. Degree is the number of outgoing or incoming connections. This can vary significantly for different nodes, resulting in a large number of incoming or outgoing links for some nodes, while others have few incoming or outgoing links. Predicting new links for nodes with a very low degree of edges can be very difficult, because there is only little information about their relations available to predict new links.

Another aspect related to topology is that the topology can change frequently, depending on the graph. Graphs such as the representation of drug-target relationships rarely change, so there is no need to focus on a method to address frequent changes in the graph. In contrast, social networks change very often, with new relationships being added or removed, or new nodes being added to the network. An inductive method for the extraction of latent features, which even efficiently extracts latent features for previously unseen data is preferable.

In this way, the model would not have to be recomputed when the graph changes, but could efficiently take frequent changes of the graph into account.

Semantics. As described above, there are both directed and undirected graphs, i.e. different types of edges between the nodes. This different semantics of edges has to be taken into account in a method, since it contains essential information for learning latent features. Furthermore, depending on the knowledge graph, further semantics can be stored in the graph, e.g. the domain and range of edges, i.e. the types of nodes from which these directed edges generally go and where they point to. In addition, schema-knowledge can be encoded for the links, e.g. *inRelationshipWith subPropertyOf married_to*, so that the conclusion can be drawn that a node also has the edge *inRelationshipWith* to the same node to which this node has the edge *married_to*. Using scheme knowledge to enrich the graph with additional information about the semantics of the relations allows the use of additional information that can be exploited in computing latent feature for predicting links to nodes.

Knowledge Representation. The last major challenge in using graph structured data is the assumption of a closed or open world assumption. The choice of one of the two assumptions has a major impact on the methods and interpretation of the results. While with a Closed World Assumption (CWA) the knowledge is considered complete and therefore everything that is not modeled is considered false, with an Open World Assumption (OWA) it is assumed that the modeled knowledge in the graphs is possibly incomplete. Accordingly, it cannot be directly concluded that knowledge is false if it is not available in the knowledge graph. Therefore, the methods and evaluation techniques used must also take these assumptions into account. In practice, a Closed World Assumption is more often assumed as it is easier to evaluate, in contrast to a Open World Assumption. However, the modeling of a complete graph, e.g. in the context of interpersonal relationships, is very unlikely, since it cannot be guaranteed that the entire knowledge is represented in a graph.

1.4 HYPOTHESES AND RESEARCH QUESTIONS

In previous work on predicting links between nodes in graph structured data, random walk approaches were commonly used [42, 102, 95]. An advantage of this approach is its easy implementation, as well as its efficient runtime. However, this approach requires an increased memory usage, as the different random walks have to be stored. Furthermore, additional hyperparameters such as the length and number of random walks have to be configured for each graph in order to achieve an effective performance for predicting links in graphs [3]. In contrast, factorization methods allow the decomposition of the graph into several non-trivial factors to provide a latent representation [83]. Factorization methods do not depend on the structure of the graph, while random walk approaches cannot reach every node if the graph is not coherent. Thus, we would expect that factorization methods for predicting links in graph structured data would provide more robust results regarding the topology of the graph. In addition, the assumption of a coherent graph is a major prerequisite that cannot always be fulfilled in practice. For this reason, we want to focus on factorization methods

to learn latent representations of graph structured data. Using these latent representations we will predict missing links in graphs. The underlying hypothesis is that factorization methods have an advantage in encoding features in graphs compared to classical random walk approaches, in particular for disconnected graphs.

We use a stochastic factorization method due to more robust results of stochastic procedures [130]. Based on the stochastic factorization method, we want to compute the distribution of the used links of nodes. We will then use the distribution of the edges to predict missing links. Besides the assumption of the better performance regarding the topology of the graph, we expect that this method is superior to classical random walk approaches, especially if an Open World Assumption (OWA) is assumed. We therefore formulate the first hypothesis as follows.

□ **Hypothesis 1**

Link distribution learning is suitable for predicting missing properties in knowledge graphs which are represented under the Open World Assumption.

In the first hypothesis, we consider incomplete knowledge graphs, belonging to the category of directional multi-graphs, that are represented under the Open World Assumption. Thus, we assume that the encoded knowledge in the graphs is incomplete and therefore not all real world information is represented in the knowledge graph. We will focus on predicting missing properties and show that models that learn a distribution function over the usage of properties are suitable for predicting missing properties in knowledge graphs. This distribution function encodes latent features of the individual nodes. We will study the impact of the Open World Assumption on the effectiveness of the considered methods, as well as characteristics of knowledge graphs, e.g. with respect to topology, which can be used to learn an effective distribution function for predicting missing properties.

As mentioned above, classical methods such as node2vec [42] or Attention Walk [3] use random walk approaches to learn a latent representation of nodes that can be used to predict missing links. These methods are applicable to graph structured data. In contrast to these methods, we use a stochastic factorization model to learn the distribution of the used links of nodes for predicting missing links. Using a stochastic factorization model, our introduced approach becomes independent of the topology of the graph. Therefore, we assume that this approach performs more effective regarding the prediction of missing links, especially on disconnected graphs than previous approaches using random walk approaches. In order to study the effectiveness of the method on bipartite graphs, we will apply the introduced method to specific graph structured data, namely bipartite networks. We formulate the hypothesis as follows.

□ **Hypothesis 2**

A stochastic factorization model is able to learn the distribution of links, even in disconnected bipartite networks.

This hypothesis restricts the learning of a distribution function to stochastic factorization models, thus a very specific model is considered here. Moreover, bipartite networks are considered rather than knowledge graphs as in Hypothesis 1. We expect that stochastic factorization models can learn the distribution of these networks efficiently, and are especially suitable for disconnected networks, compared to existing methods which are often based on random walk approaches. In this context, we will also study the characteristics of the networks in relation to the performance of the considered methods in order to gain more insights into the effectiveness of the methods.

On the basis of the preceding hypothesis, it can be assumed that if the stochastic factorization method can be used to effectively learn latent features for predicting interactions in bipartite networks, we can apply this method in a similar way to knowledge graphs, i.e. for the classification of nodes and edges. We therefore hypothesize that latent features can be encoded using sub-symbolic representation for node classification. We formulate the hypothesis as follows.

□ **Hypothesis 3**

The sub-symbolic KG representation learned with our model encode latent groups of nodes and links in knowledge graphs.

The stochastic factorization model can be used to learn latent features of nodes and edges which are especially suitable for the classification of nodes and edges. We want to verify this hypothesis by demonstrating the effectiveness of the learned sub-symbolic KG representations by node and edge classification.

1.5 CONTRIBUTIONS

Considering existing works and studies dealing with similar topics, it can be seen that they are often based on random walk approaches to learn latent features from the graph structured data in order to perform link prediction, node classification or related tasks. We want to distinguish ourselves from existing work and use among others stochastic neural networks to learn latent features. We expect an advantage over existing work especially regarding disconnected graphs, since not all components of disconnected graphs can be reached by random walk approaches.

Thereby this might also has a positive effect on knowledge graphs, since they do not necessarily have to be connected. In addition, we would like to study our proposed methods with regard to an Open World Assumption in comparison to existing work. We therefore formulate the scientific contribution regarding the first hypothesis as follows.

☞ **Contribution for Hypothesis 1**

Approach and model for predicting missing properties of knowledge graphs which are represented under the Open World Assumption.

Based on the second hypothesis, we will develop a model to learn latent features especially in bipartite networks, even if these networks are not connected.

Likewise to the first contribution, this model will also use a stochastic neural network that will help us to compute the distribution of the usage of interactions of each node of the graph structured data in order to predict missing interactions in the networks. We formulate the second contribution as follows.

> ☞ **Contribution for Hypothesis 2**
>
> Model for capturing distributions of nodes for predicting missing links, even in disconnected bipartite networks.

The third contribution is based on the third hypothesis and uses the model we developed in the second contribution to learn a representation of nodes and edges with a modified variant that is specifically tailored to knowledge graphs. As a result of this contribution we present a sub-symbolic KG representation, encoding latent features on the class membership of entities and relations. Similar to the previous hypotheses, a large number of sub-symbolic KG representations use random walk approaches and relational learning approaches to learn sub-symbolic KG representation. Using stochastic factorization, we expect that our introduced method is much more robust regarding the topology of the knowledge graph. Therefore, we formulate the scientific contribution based on the third hypothesis as follows.

> ☞ **Contribution for Hypothesis 3**
>
> A sub-symbolic KG representation that captures relevant classes for entities and properties in a KG.

1.6 OUTLINE

Each contribution is based on its own hypothesis. In the next section we will introduce the fundamentals that are needed to comprehend this thesis. Afterwards, we will dedicate individual sections to each hypothesis in which we will divide it into further research questions and conduct appropriate experiments in order to address these questions. The order of the chapters is based on the hypotheses and contributions. In the following, Chapter 3, we will study methods for predicting missing properties in knowledge graphs under the OWA and introduce two methods that address this issue. In the further course, in Chapter 4, we will study the problem of predicting missing links in bipartite networks. In particular, we will consider connected and disconnected bipartite networks. In Chapter 5 we will predict instance types, as well as domains of properties in knowledge graphs. This problem is closely related to the first hypothesis, since links in knowledge graphs are predicted in this context as well, but these represent very specific information, so-called schema knowledge. In Chapter 6 we will conclude the whole work, as well as give an outlook on future research activities in the context of the findings presented here.

2

FOUNDATIONS

2.1 GRAPHS

A graph is a mathematical structure, which can be used to model relations between objects. The first paper in which graphs were used and thus are considered the beginning of graph theory was published in 1736 by Leonhard Euler in which he described and solved the problem of the Seven Bridges of Königsberg [30]. A graph, denoted G, comprises a set of nodes, denoted V, and a set of edges, E. In the simplest case, the set of edges is an unordered pairs of nodes and is called undirected simple graph. The definition of an undirected simple graph permitting loop is as follows.

Definition 2: Undirected Simple Graph

A graph G is an ordered pair $G = (V, E)$, where V is a set of nodes and $E \subseteq \{\{x, y\} | x, y \in V\}$.

As defined, the edges of the graph, i.e. the connections between the nodes, are undirected. The edges are interpreted in that way, that one edge $(x, y) \in E$ is from node x to y and vice versa. Graphically, the nodes are usually represented as dots in a graph and the edges as links between the dots. In the case of an undirected simple graph, the edges are visually represented as undirected lines between the dots. Figure 2 shows a visual representation of an example of an undirected simple graph.

Beside undirected simple graphs there is a variation in which the edges of the graph have an orientation. This type of graph is called directed graph or digraph. Directed graphs permitting loops are defined as follows.

Definition 3: Directed Graph

A Graph G is an ordered pair $G = (V, E)$, where V is a set of nodes and $E \subseteq \{(x, y) | (x, y) \in V^2\}$.

When comparing the two definitions between undirected simple graphs (Definition 2) and directed graphs (Definition 3), it is noticeable that only the type of edges has changed. Instead of undirected simple graphs in which the set of edges consists of unordered pairs, the set of edges in directed graphs is a set of

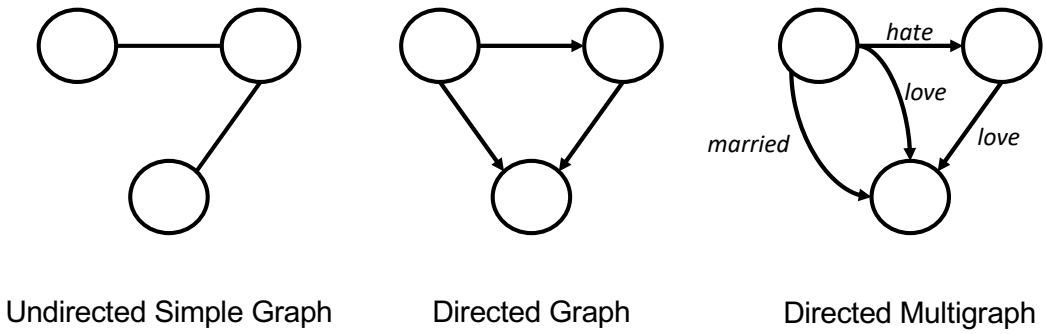


Figure 2: Visual representation of three different types of graphs.

ordered pairs. Thus we give the edges its orientations, which is the main difference compared to undirected simple graphs. Thereby the nodes in the context of an edge from which the relation originate, which are named as x in the Definition 3 above, are often named as source or head and the nodes to which the edges lead, denoted as y in Definition 3, are named as sink, target or tail. Graphically the nodes of a directed graph are represented as dots similar to undirected simple graphs. Unlike undirected simple graphs the edges are shown as a directed relation in which the arrow indicates to which node the edge points to. Figure 2 shows an example of a visualization of a directed graph.

As can be seen in the definition of directed graphs, there is no distinction between the edges. Each edge has the same meaning. In order to assign several edges between the nodes and thus different meanings to them, there is an extension to directed graphs, called directed multigraph. In the following we give the definition of directed multigraphs permitting loops.

Definition 4: Directed Multigraphs

A Graph G is an ordered pair $G = (V, E, \phi)$, where V is a set of nodes and E a set of directed edges. ϕ is an incidence function $\phi : E \to \{(x, y) \mid (x, y) \in V^2\}$.

By introducing an incidence function, different edges can be assigned to an ordered pair of nodes. Thus, different edges can be assigned to the same ordered pair of nodes. Directed multigraphs are visualized similar to directed graphs. Nodes are represented as dots and edges as a directed relation between nodes. The difference is, that for the same directed node pairs now different directed edges are allowed. To distinguish them visually the edges are usually labeled according to the set E to distinguish them. Figure 2 provides an example of a visual representation of a directed multigraph. Later we will introduce knowledge graphs, which represent a more advanced directed multigraph.

As another type of graphs we introduce bipartite graphs in the following. Bipartite graphs are undirected simple graphs, where the nodes of the bipartite graph can be divided into two disjoint sets, V_1 and V_2. Compared to undirected simple graphs, edges are only allowed between nodes of these two different sets. Bipartite graphs are defined as follows.

Definition 5: Bipartite Graph

A bipartite graph $G = ((V_1, V_2), E)$ is a graph $G = (V, E)$, whereby, $V_1 \cap V_2 = \emptyset$, such that every edge $(v_1, v_2) \in E$ connects a node $v_1 \in V_1$ with a node in $v_2 \in V_2$.

Bipartite graphs are often used to model affiliations in social networks, e.g. to model information about the affiliation of users to social groups. Visually, bipartite graphs are presented similar to undirected simple graphs. Nodes are represented as dots and edges as connections between the corresponding dots. The nodes are grouped according to their affiliation to one of the two sets and, if necessary, visually marked with different colors or with a label. An example of a visual representation of a bipartite graph is given in Figure 3.

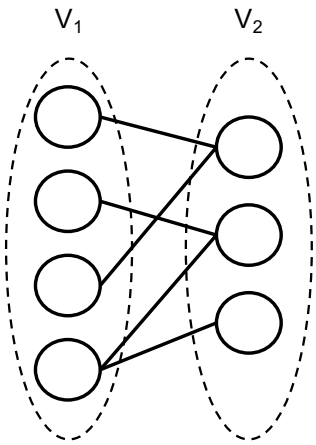


Figure 3: Example of a bipartite graph.

Often the term network is used in the context of graphs, sometimes as a synonym or as a special indication that the edges of a graph represent the interactions between nodes. Accordingly, edges in networks, compared to graphs, denote interactions between nodes and in graphs in general relationships between nodes.

Besides the graphs presented above, there are many other different types of graphs, such as hypergraphs, which are not relevant for this work, therefore we will not introduce them in the following. Further insights and application examples concerning graphs can be found in textbooks [71].

Following the introduction of different graphs and their visual representation, we would like to take a closer look at the adjacency matrix as a data structure for the representation of finite graphs. Adjacency matrix is a square matrix where its elements indicate whether an edge exists between two nodes or not. In the following we will only consider binary adjacency matrices, but they can also consist of real numbers.

Definition 6: Adjacency Matrix

Given an undirected simple graph G in which the set of edges is represented by an unordered set tuples (i, j), whereby i and j represents a natural number of the start and end nodes of the edges. Then the elements of the adjacency matrix A for the graph G is represented by

$$a_{ij} = a_{ji} = \begin{cases} 1 \text{ if } (i,j) \in E, \\ 0 \text{ otherwise.} \end{cases}$$

Depending on the graphs considered, there are always slightly different definitions for adjacency matrices, the above Definition 6 refers to undirected simple graphs. In general, adjacency matrices are squared in which each row i and each column j represents a node. Based on the undirected simple graph in Figure 2 the following adjacency matrix results.

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Using this data structure, graphs of many different types can be stored and edited. The advantage of this data structure is the fast access time for changing entries, which is in $O(1)$. In contrast to this, there is the increased memory consumption, which is in $O(n^2)$, where n corresponds to the number of nodes, i.e. $|V|$.

2.2 KNOWLEDGE GRAPHS

Knowledge Graphs (KGs) can be considered as a model in which information is stored. In contrast to relational databases, this information is not structured, but semi-structured as a graph. Due to the links between the information in the graph, knowledge is generated, thus making it a knowledge graph. The term knowledge graph was first used by Google in 2012, after they had acquired Metaweb and Freebase, i.e. two datasets with a variety of information[5]. The knowledge graph enabled Google to integrate the information into the search engine to further enhance the service. However, the foundation of today's knowledge graphs is based on semantic networks that were developed in the 1960s [106].

As mentioned above, knowledge graphs allow to represent information in a semi-structured way. There are different serializations, i.e. forms for representing the information of the knowledge graph. The information can be represented as a list of triples in the form (h, r, t). Thereby h denotes the head, r the relation and t the tail. Furthermore there is the distinction between entities, classes and literals. Entities are objects which are described by a unique identifier, classes are concepts using a unique identifier for identification. Unique identifiers are

[5] https://blog.google/products/search/introducing-knowledge-graph-things-not/

for example an IRI. Literals are represented using strings. Relations themselves can be described with information. We denote a Knowledge graph as following.

Definition 7: Knowledge Graph

A knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{L}, \mathcal{C})$, is a tuple of pair-wise disjoint sets $\mathcal{E}$, $\mathcal{R}$, $\mathcal{L}$, and $\mathcal{C}$ correspond to the set of entities, relations, literals, and types or classes, respectively. A statement in $\mathcal{G}$ is modelled as a triple (h, r, t), with $h \in \mathcal{E} \cup \mathcal{R} \cup \mathcal{C}$, $r \in \mathcal{R}$, and $t \in \mathcal{E} \cup \mathcal{R} \cup \mathcal{L} \cup \mathcal{C}$.

An entity e is a real-world or abstract object, which can be identified by its URI. Entities are for example real-world objects like *Angela Merkel* or abstract objects like e.g. *Mickey Mouse*. A relation r describe entities and can be identified by its URI as well. Relations can be any relation like e.g. *birthdate*, *place_of_birth* and *populationTotal*. Classes are a set of entities that logically group entities which share similar characteristics. Classes are e.g. *Human*, *Animated characters* and *Place*. Because of the abstract concept of a knowledge graph all kinds of knowledge can be stored in it. Often there is a distinction between ABox and TBox statements. While ABox statements describe concrete facts such as *'Angela Merkel is Chancellor'*, TBox Facts describe the scheme of knowledge, i.e. classes and domains such as *'All Chancellors are humans'*. This example already demonstrates that concepts such as chancellors themselves are included in the knowledge graphs and can be described in more detail. By describing the concepts, semantic knowledge can be stored which can be used to conclude further knowledge. The modeling of the information in the knowledge graph is often done using RDF. RDFS is an extension of RDF to model schema knowledge in the knowledge graph.

For visual representations of knowledge graphs, directed multigraphs are used (see Definition 4). Each subject and object is represented as a node whereas the relation between them is represented by a labelled edge. In the representation of nodes, the label is often added to the node for improved readability. Literals are represented as rectangles for better distinction them from entities. Figure 4 illustrates an example of a knowledge graph in which knowledge about the entitites *Angela Merkel* and *Hamburg* is encoded.

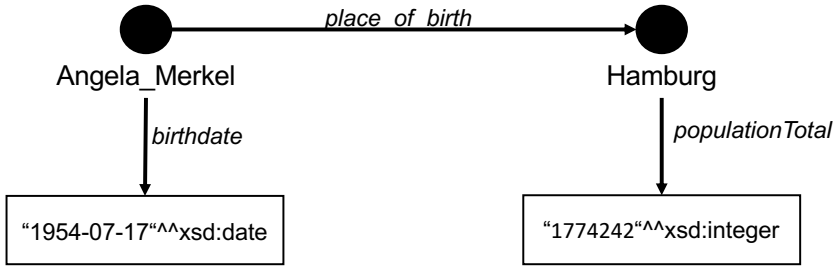


Figure 4: Example of a knowledge graph.

Knowledge graphs can contain several million different entities such as DBpedia[6] and Wikidata[7]. DBpedia is a knowledge graph that extracts structured

[6] https://wiki.dbpedia.org/about/facts-figures
[7] https://www.wikidata.org/wiki/Wikidata:Statistics

information from various Wikimedia projects and stores it in the knowledge graph [9]. Thus, information from e.g. Wikipedia is available in a knowledge graph, enriched with semantic relations. The semantics, i.e. the description of relations, can include domain and range assertions. The domain of relations describes from which types of entities the relation starts. An example would be the relation *place_of_birth*. This relation usually originates from humans, so that the domain of this relation can be specified using *Human*. The range of a relation describes which type of object it should be. These can be different classes, but also data types like dates. In the case of the relation *place_of_birth* the range would usually be a location, so that the class *Place* could be used as range to specify the range of this relation. By describing the relation with such information it gets a certain meaning, therefore it is called semantics. Thus, if we consider the fact *(Angela_Merkel,place_of_birth,Hamburg)* in addition to the schematic knowledge above, we can conclude that *Angela_Merkel* is of type human, since the domain of *place_of_birth* is *Human* and Hamburg must be a place, because the range of the relation is *place_of_birth* is *Place*.

As already seen in the example above, a large amount of implicit information can be inferred from very few facts, making it interesting to add further facts automatically, which leads to an increased possibility of deriving further knowledge from the knowledge graph.

2.3 NEURAL NETWORKS

Neural networks are a group of machine learning methods, which are inspired by biological neural networks of living beings. Warren McCulloch and Walter Pitts established the foundations in 1943 by developing the model of neural networks [69]. Rosenblatt created the first perceptron in 1958 [109]. Over the years a variety of different types of neural networks have been developed. In this section we will focus on the simplest form, the feed-forward networks. Further types of neural networks can be looked up in literature [41].

Neural networks can be represented as a directed graph. Nodes are called neurons and the edges represent the weights between the neurons. Neurons are arranged in layers, whereas in feed-forward networks only directional connections exist between neurons of the next layer. A neural network always consists of at least one input layer and one output layer, optionally there can be any number of layers between these two layers. These optional layers are called hidden layers. Figure 5 illustrates a 4-layer neural network. In case of fully-connected feed-forward neural networks, each neuron is connected to every other neuron of the next layer. The idea of neural networks is that neurons receive information from the neurons of the previous layer, process it and forward it to the neurons of the next layer.

In the following $x \in \mathbb{R}^n$ denotes the input of the neural network and $\hat{y} \in \mathbb{R}^c$ the output, whereby c denotes the number of classes. Furthermore, the indexing of the vector x describes the concrete elements of the vector, i.e. $x = (x_1, x_2, \ldots, x_n)$. The set of all samples, x, are stacked in a matrix to obtain X, where $X \in \mathbb{R}^{n \times m}$. Further we denote the set of neurons of the layers as Z and the activation values as A. Lower case letters denote individual neurons, whereas up-

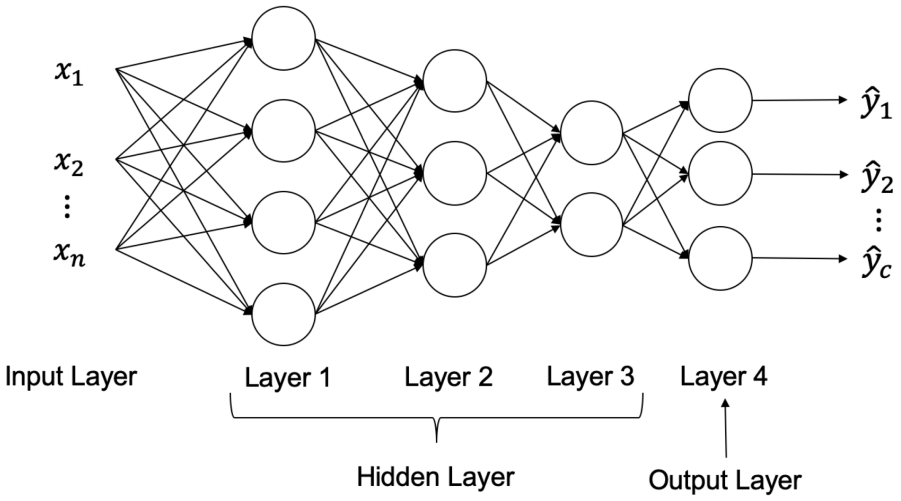


Figure 5: Illustration of a 4-layer neural network using x as input and $\hat{y}$ as output.

per case letters denote all neurons of a layer. To understand which layer is meant, we introduce the variable l, indicating which layer we are currently considering. The number of layers is specified by L.

Considering Figure 5, $L = 4$, because four layers exist in the neural network. We denote with $a_1^{[1]}$ the activation value of the first neuron in the first layer. In comparison, $a_2^{[3]}$ describes the activation value of the second neuron in the third layer.

In order to calculate the output of a neural network, the input is passed through the neural network to the output layer, by multiplying the input with the weights between the layers and applying an activation function. First the input features of the neural network are multiplied with the weights W of the first layer and adding a bias value b. The formula is as follows:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

Afterwards an activation function is applied to the individual neurons of the calculated values. This activation function is referred to as σ in the following.

$$A^{[1]} = \sigma(Z^{[1]})$$

After calculating the activation function, these values are forwarded to the next layer. Thus they serve as input for the next layer. In general these two formulas can be represented as follows.

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = \sigma(Z^{[l]})$$

The last layer, $A^{[L]}$, represents the output and is usually denoted as $\hat{y}$.

The activation functions used have a great impact on the decision function calculated within the neural network and on the performance regarding the accuracy of the neural network. In the following we will present only a part of possible activation functions. The goal of the activation function is to introduce non-linearity into the neural network. For each layer a different activation function can be specified. The activation functions presented in the following are shown in Figure 6.

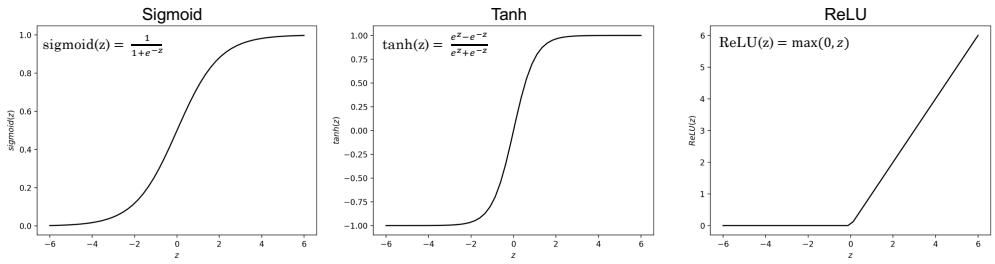


Figure 6: Illustration of different activation functions for neural networks.

Sigmoid. The sigmoid activation function is defined as follows.

$$\mathrm{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

The sigmoid activation function yields values between 0 and 1, thus $\mathrm{sigmoid}(z) \in (0,1)$. This allows for a simple and useful interpretation of the output values of the sigmoid activation function as probability of how much a neuron fires or how much the downstream neurons of the next layer are activated. The disadvantage of this activation function is the gradient, i.e. the slope of the activation function for very large and very small values of z, as these values are very small, almost zero. Only for values $z \in [-2,2]$ the slope is sufficient high compared to other values, which is why the neurons converge faster in this range. The slope is important because backpropagation is used for learning neural networks, using the gradient descent method [55] and taking the slope of the function into account when adjusting the weights. For small gradients, only a small adjustment of the weights results, causing a low speed of convergence of the neural network. For this reason usually activation functions with a higher slopes are used. Nevertheless, the output of the sigmoid activation function is often used as probabilities because of its easy interpretability and thus this activation function is usually used in the output layer.

Tanh. The Tanh activation function is defined as follows.

$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

This function yields values in the range -1 and 1, i.e., $\tanh(z) \in (-1,1)$ and thus has a larger scope than the sigmoid activation function. Apart from that, it shares many characteristics with the sigmoid activation function, in which the slope is also sufficiently high only for values $z \in [-2,2]$ to allow a fast convergence of the neural network. In practice this activation function is usually preferred over the sigmoid activation function, as it has an higher slope. However, the Tanh function is mostly used only in the hidden layers, because the output values of the Tanh activation function cannot be interpreted as probabilities.

ReLU. The ReLU activation function has become very popular especially in the last few years. It is defined as follows.

$$\mathrm{ReLU}(z) = \max(0, z)$$

Due to its constantly high gradient for values greater than 0, i.e. $z > 0$, which is one, the ReLU activation function generally converges faster than the previous two activation functions. According to published studies, neural networks using the ReLU activation function converge six times faster than neural networks compared to a Tanh activation function [53]. However, when using a ReLU activation function, there is a risk of irreversible dying of neurons during training, especially when using a too high learning rate. A high negative update of the weights can cause the ouput of the ReLU activation function to always be 0, and the gradient, which is 0, given a negative input, causes the adjustment of the weights to be 0. However, this happens rarely if the learning rate is adjusted appropriately.

Neural networks use backpropagation to learn the weights of the networks. Backpropagation uses the gradient descent method in conjunction with the chain rule to learn the weights of the neural network. If we denote the cost function of the network as J and the learning rate as α, then the formula for adjusting the weights of the neural network is defined as follows.

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial}{\partial W^{[l]}} J$$

We will not discuss the details of the derivative of the activation functions and implementation of the backpropagation technique, as this would go beyond the scope of this section. More details about learning neural networks are sufficiently described in literature [41]. Essential for this thesis is an understanding of the structure of neural networks and the mathematical concepts for calculating the output.

3

LINK PREDICTION ON KNOWLEDGE GRAPHS

3.1 INTRODUCTION

KGs have become an important foundation to represent knowledge exploited in, e.g., Question Answering [66], Entity Linking [94], and recommender systems [88]. A complete knowledge graph, or a knowledge graph containing a large amount of information, offers a considerable advantage in question answering systems and information retrieval. Search providers commonly use knowledge graphs to improve their services, as does Google to improve its information retrieval and services (Google Knowledge Graph[8]). The Google search provides, when searching for entities such as *Angela Merkel*, in addition to the hits on websites, likewise useful information about this entity such as place of birth, relationship status and profession in an infobox. Thereby a first overview of an entity can already be given and the most important facts can be made available without visiting further pages. This service is an important medium to provide users of the service a quick and clear overview of information about entities. Besides using a knowledge graph for information retrieval, a knowledge graph may also be used for question answering. Google, like other search providers, answer simple questions using a knowledge graph. Thus, the question *'What is the alma mater of Angela Merkel?'* in Google already returns the correct answer. Just like the infobox, the Question Answering service is an essential service to quickly and directly answer questions from users. In both cases a knowledge graph is used to provide the information. A complete knowledge graph is thereby essential for both services to ensure a high coverage of information and thus to have an advantage against competitors, which leads to a higher market success[9].

There are already a number of very large public knowledge graphs such as Freebase [19], DBpedia [9] and Wikidata [117] that consist of millions of facts and can be used for such services. However, even though they consist of millions of facts, they still suffer from incompleteness which may hinder the effectiveness of the applications where they are consumed. The problem of KG completeness has been studied in different contexts: (i) predicting a missing component in a head-relation-triple [11, 20, 122], (ii) predicting types of entities [128], and iii) predicting missing values in query answers [4, 5, 24]. In most of these methods embeddings are trained, in which the nodes and edges of the knowledge graph are encoded into low-dimensional spaces while keeping the graph structure. Based

[8] https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html
[9] Google made in 2019 a revenue of 45 billion

on these embeddings, the structure is analysed using machine learning methods and missing head entities, relations and tail entities are predicted. This method has proven to be very effective in predicting entities and relations on certain knowledge graphs, but most of these methods assume a closed world assumption, meaning that facts that are not encoded in the knowledge graph are considered to be false. This assumption is very useful for an evaluation of the method, as it makes the method very easy to evaluate, but it is too restrictive when considering a real-world system. Furthermore, the existing methods are often evaluated on very few knowledge graphs, resulting in no well-founded evaluation of the methods on different characteristics of knowledge graphs. Knowledge graphs can have very different characteristics, e.g. different topologies regarding the degree of nodes and the connectivity of a graph. Likewise, there are different definitions of knowledge graphs and therefore different types of knowledge graphs. In very sophisticated knowledge graphs, semantic information such as domain and range information of relations can be encoded. In our opinion it is important to study the methods for different characteristics of the knowledge graph in order to select the appropriate method, given the characteristics of a knowledge graph.

We want to address these existing limitations and create a method that predicts missing relations in knowledge graphs, even if they are represented under the Open World Assumption (OWA) and with different level of semantics. Instead of relying on embedding methods, as existing work does, we want to develop two methods and compare the effectiveness of them. We will develop a method based on rule mining and a stochastic factorization model. We thereby investigate the problem of KG completeness from an entity-centric view. Our goal is to predict missing relations, even under Open World Assumption, and thus contribute to the completion of knowledge graphs to support information retrieval and question answering. To get a common understanding of a knowledge graph, we define the knowledge graph as follows.

Definition 8: Knowledge Graph

We denote a knowledge graph $KG = (H \cup T, R)$, where H denotes the set of head entities, T the set of tail entities, and R the set of labelled relations. The information in the knowledge graph KG is modeled as triples (h, r, t), with $h \in H$ denotes the head entity which has a relation $r \in R$ to a tail entity, denoted as $t \in T$.

The above Definition 8 enables a common understanding about knowledge graphs. There are several different definitions of knowledge graphs, some of which are much more restrictive [89]. We have decided to use a very broad definition of a knowledge graphs, which is already sufficient for our purposes. The use of this broad definition of knowledge graphs demonstrates the broad use of our methods, which we will introduce later. The methods are therefore applicable to different specific knowledge graphs.

In general, a Knowledge Graph KG consists of facts, which are modelled using triples. The set of these facts creates the total available knowledge. As described

above, real-world knowledge graphs may be incomplete and, although a relation such as *alma_mater* should exist for an head entity such as *Angela_Merkel*, it is not represented in the knowledge graph. Therefore, the knowledge graph is incomplete, since not all facts are available in the knowledge graph. In order to define a complete knowledge graph, we first define a function which contains the set of all relations for a given head entity.

Definition 9: Set of Relations

We denote $R_h(KG)$ as a function which is the set of relations where the entity h appears in the head of a statement in KG, i.e., $R_h(KG) = \{r \mid \exists t \in T, (h, r, t) \in KG, r \in R\}$.

Based on the above Definition 9 we can now very easily define the ideal knowledge graph KG^* which, based on KG, is complete for all head entities regarding the available relations.

Definition 10: Complete Knowledge Graph

Given a knowledge graph KG, consider KG^* the ideal graph that contains all the statements that should be in KG, i.e., $KG \subseteq KG^*$ and therefore including all relations, i.e. $R_h(KG) \subseteq R_h(KG^*)$.

Our goal is to develop two models to predict missing relations based only on a head entity. One of the methods is based on rule mining and the other on a stochastic factorization model. By using different learning paradigms, a direct comparison of the two paradigms can take place. We want to evaluate the effectiveness of both methods on different knowledge graphs. Our hypothesis is that a stochastic factorization model can learn the distribution of the usage of relations for head entities in a knowledge graph which are represented under the Open World Assumption. Therefore, we want to analyse the effectiveness of a factorization model for the prediction of missing relations, given an head entity, in knowledge graphs under the Open World Assumption and compare them against a rule mining method. In particular, we want to study the characteristics of knowledge graphs, for which our methods for predicting missing relations are very effective. The goal is to be able to indicate whether a high performance can be expected before executing our methods. In addition, the findings found can be reflected back into our methods for refining them to improve the performance. Furthermore, we would like to study the impact of the Open World Assumption on the effectiveness of the methods. We would like to distinguish ourselves from existing work and investigate the methods with respect to OWA in order to take a step towards loosening a restrictive Closed World Assumption (CWA). The hypothesis we make, including the research questions based on it, is formulated in Research Question 1.

Research Question 1

Hypothesis: Link distribution learning is suitable for predicting missing properties in knowledge graphs which are represented under the Open World Assumption.

1.1 What is the effectiveness on learning KG features for link prediction of stochastic factorization models, in contrast to rule mining methods.

1.2 What are the characteristics of knowledge graphs that allow learning models to effectively learn the distribution of links?

1.3 What is the impact of following the Open World Assumption on the effectiveness of the studied methods?

3.1.1 *Structure of the Chapter*

In this part of the thesis we will focus on the problem of predicting missing relations in knowledge graphs. We will introduce two different methods, which are based on different learning paradigms. For better understanding of the work and the problem we will give an example of motivation in Section 3.1.2. The motivating example is furthermore used to illustrate the procedures in the following sections with a practical example thus making the procedures more comprehensible. In the following section, Section 3.2, related work in the context of knowledge graph completion, with special focus on predicting missing relations will be presented. We will discuss the existing work and distinguish them from the methods presented. In Section 3.3 we present two approaches which are based on different learning paradigms, but pursuing similar goals, namely predicting missing relations in knowledge graphs. The motivating example from Section 3.1.2 is used to illustrate the two methods and thus make them more comprehensible. Details about the individual aspects of our methods are given, such as the input to our models, parameter learning, and computing the predictions, based on our model. In the following section we will conduct experimental studies. These studies are designed to support us in answering the research questions. We will discuss the effectiveness of our methods and compare them to baseline methods. Furthermore, we will explain which characteristics knowledge graphs need to have in order to guarantee a high effectiveness of our methods. Based on these experiments we can then answer the research questions posed in the introduction in Section 3.5. We thereby will summarize the presented work and, based on the findings from the experiments, give an outlook for future work, which, in our opinion, is worthwhile to continue.

3.1.2 *Motivating Example*

In the following, we introduce a simple scenario, which on the one hand illustrates the problem in more detail, and on the other hand serves as a running example to explain the methods in more detail in the later sections and illustrate them with a practical example. We use a knowledge graph based on Definition 8,

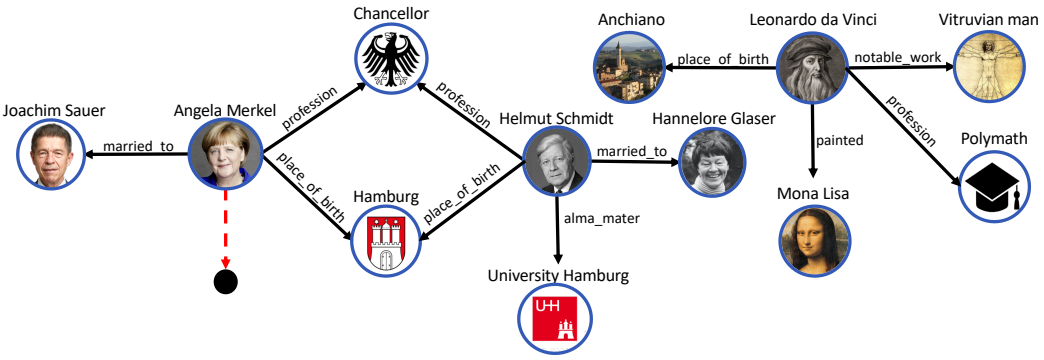


Figure 7: Running example for predicting missing properties in Knowledge Graphs. We are interested in predicting missing properties for entities, such as for *Angela_Merkel*.

which is a very general definition and models the facts using triples. The knowledge graph can be visualized as a graph where head and tail are nodes and the relation is a directional labelled edge between the head and tail. In Figure 7, a subgraph of the knowledge graph DBpedia [9] is visualized, in which different facts of persons like *Angela_Merkel* and *Helmut_Schmidt* are represented. This knowledge graph corresponds to a knowledge graph according to Definition 8 and can be used for improving information retrieval and question answering, among other things. For example, this knowledge graph could be used to answer the question about the birthplace of *Angela_Merkel*. But considering the knowledge graph in detail, it is noticeable that there are much more relations available to describe an entity than currently used by the entity *Angela_Merkel*. For example, the entity *Helmut_Schmidt* uses the property *alma_mater* to describe at which university the entity studied. This property is not used to describe the University of *Angela_Merkel*, although both entities are very similar in terms of the relations used, as well as the tail entities. It is questionable if *alma_mater* is a missing outgoing relation of *Angela_Merkel* and if there are other relations missing to describe *Angela_Merkel*. Identifying these potentially missing relations manually for a small knowledge graph as shown in Figure 7 is possible with little effort. However, it is impossible to identify them manually in a large knowledge graph with several million entities and thousands of relations, like in DBPedia and Wikidata. For this reason, an automatic procedure is needed to identify missing relations, given a head entity. Adding this information to the knowledge graph allows the procedures, i.e. question answering systems, that access the knowledge graph and use the information encoded in the knowledge graph to work even more efficiently since more information is available. Likewise, services can be expanded and work can be done even more precisely to answer questions such as *'What is the alma mater of Angela Merkel?'* This question cannot be answered according to the information as shown in the knowledge graph of Figure 7. Moreover, as described at the beginning, a complete knowledge graph represents a significant market advantage for the provision of such services. In this chapter we will therefore focus on developing methods for the automatic identification of missing relations. Therefore we define our motivation for this chapter as follows.

Motivation : Predicting missing relations in knowledge graph

Based on a knowledge graph $KG = (H \cup T, R)$, a function $f(h, r)$ with $h \in H$ and $r \in R$ will be learned, which indicates the likelihood of h having a relation r to a tail entity t in KG. This function is intended to predict missing relations in the knowledge graph KG. Hence we can identify the set of missing relations of h, i.e., the set of relations such that $R_h(KG^*) \setminus R_h(KG)$.

Based on the above motivation, we are interested in a function $f(h, r)$ which, given a head entity h and a relation r, indicates how likely this relation will be used to describe the head entity. In many related papers the function is based on a triple, such that based on a head entity h and a tail entity t, a function indicates how likely the relation r describes the relationship between the head entity h and the tail entity t. However, we will focus and use only the head entity h to compute the likelihood of the relation r as a missing relation for describing the head entity h. Since, unlike in the existing work, we assume an Open World Assumption and assume that not every entity is encoded in the knowledge graph KG. The related work fails if, for example, the University of Leipzig is the alma mater of Angela Merkel, since this tail entity is not included in the knowledge graph KG. In the following section we will go into this in more detail and present related works and distinguish ourselves from them to emphasize the added value of our motivation.

3.2 RELATED WORK

A very prominent method, which has received a lot of momentum for predicting missing relations in knowledge graphs, is TransE [20]. TransE represents relation, head and tail entities as vectors, so that the sum of head entity and relation vector equals the tail vector. An advantage of using TransE is its simplicity, since it requires very few parameters and is efficient in training. However, this simplicity comes at the cost of dealing with 1-to-N, N-to-1 and N-to-N relations. The flaws in modelling power comes from the used scoring function. TransE is only suitable for 1-to-1 relations. In order to overcome this drawback, multiple extensions to TransE were introduced like e.g. TransH [122] and TransR [59]. TransH introduces distributed representations for entities when involved in different relations in the same semantic space. TransH thus overcomes the simplicity of TransE of dealing many-to-many relations. TransE and TransH represent the entity and relation vectors in the same semantic space. However, entities and relations are different objects, therefore they should not be represented in the same semantic space. In contrast to this, TransR represents entities and relations in distinct semantic spaces.

ComplEx [115] is based on the same concept as TransE, but uses complex-valued vectors for predicting relations in knowledge graphs. Both methods are transductive learning algorithms, making it possible to predict missing parts of triples, given that the individual entities and relations are known to the model in advance. In contrast to these methods, EDMAR [114] and RDF Shape Induction [73] are inductive methods that learn a general model from examples

and are therefore applicable to all triples, even if entities and relations were not known in advance while learning the model. Another very promising method for Link Prediction is HolE [82]. This method exploits circular correlation in relations for creating representations. RESCAL [83] is a tensor factorization approach. This approach models the KG as a tensor and uses factorization to predict relations between entities. DistMult [125] learns relation embeddings and utilizes the embeddings to mine logical rules. The logical rules are used to predict relations between entities. ConvE [25] and HypER [12] use both convolutional layers for processing and predicting missing relations. TuckER [11] generalizes the previous works of RESCAL, DistMult, and ComplEx. TuckER is a linear model, based on the TuckER decomposition. The facts are modeled in a binary tensor. ANALOGY [61] exploits the analogical structures of the KG, similar to semantic analogy of words in methods like e.g. Word2Vec [75] and GloVe [93].

In the context of KG completion, there are approaches that rely on the symbolic representation of KGs. HARE [5] is an engine that detects missing values in a KG based on the Local-Closed World Assumption. It crowdsources the missing values to complete the KG and allows for answering SPARQL queries. Other work specifies the number of missing relations and thus measures the completeness of the KG [92]. The information about missing relations can be used to learn rules [38] for KG completion. For more details on KG completion, please refer to survey paper [14]. All these methods predict relations between two given entity nodes in the KG, while our approaches only consider the head entity as input and predicts the missing relation.

Frequent Itemsets. Highly-correlated itemsets are based on the concept of frequent itemsets [6]. This technique claims to find the items that occur very often in the customer's transactions. High-utility Itemsets [34, 64] is an extension to Highly-correlated itemset mining in which the most frequent itemsets are to be found, which yield the highest profit. The utility of the transactions is the most important criteria. A number of procedures use Transaction-Weighted Utilization to prune the search space [108, 65]. These procedures first generate all possible candidates and then filter out any candidates whose utility is too low. The disadvantage, however, is that the procedure generates a large number of unnecessary candidates. This results in a long runtime. Therefore, algorithms with pruning mechanisms have been proposed to reduce the number of itemsets candidates and thus reduce the computational complexity [34, 64]. Faster High-Utility Itemset Mining (FHM) is a very fast High-Utility Itemset Mining algorithm [34], which reduces the number of join operations and thus improves the runtime. However, utilizing utility results in many itemsets which yield a high utility but correlate only very weakly. Therefore, FHM was extended to guarantee that the itemsets correlate strongly, besides yielding a high utility [34].

Community Detection. The identification of communities is particularly prominent in the area of social network analysis [101]. Community detection, however, is not exclusively applicable to social networks. Network analyses in the field of co-authorship are also conceivable [78]. However, an existing problem in this research area is that there is no common definition of a community. This has already been mentioned in previous works [33]. This leads to the effect that there are different conceptions for the term community. In general,

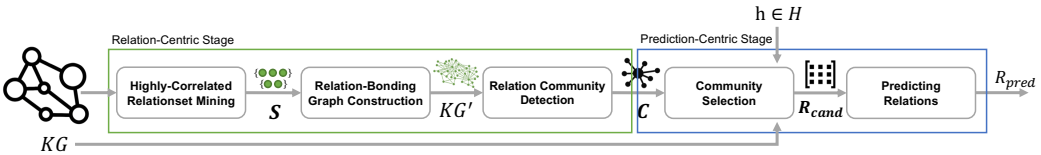


Figure 8: Proposed approach for the predictions of missing relations for head entities, based on a knowledge graph KG. The relation-centric stage captures latent knowledge between the relations. The prediction-centric stage predicts missing relations based on the communities detected in the previous stage and the KG G for a given head entity h.

a community is a dense subgraph. Detecting them is computationally expensive. For this reason, methods have been developed which, for example, use random walks to speed up the computations [97] or use grouping methods to simplify computations [99]. However, a trade-off will arise here between the quality of the results and runtime for large networks. Nevertheless, there are also approaches that have returned reliable results on very large graphs while exhibiting a satisfying runtime [22]. Some community methods focus on detecting non-overlapping communities [22], others like e.g. Community-Affiliation Graph Model (AGM) [126, 127] allows for detecting overlapping, non-overlapping and hierarchically nested communities. Analyses and comparisons about existing community structure algorithms allow to explain the advantages and disadvantages of the individual algorithms more precisely and show to which extent these algorithms can be improved [26, 33, 80].

3.3 LEARNING LATENT FEATURES FOR PREDICTING MISSING RELATIONS

In the following we will present two methods that allow for predicting missing relations under the Open World Assumption for head entities, based on a knowledge graph. The first method presented, *Community-based Relation Prediction* (CRP) is based on itemset mining and a clustering method on graph-based data. This method is based on the idea that related relations occur in common and can thereby be clustered into latent groups. The second method, *Link Distribution Learning* (LDL), was developed on the basis of the collected insights of CRP, in which it was found that similar entities have a similar distribution regarding the use of relations.

3.3.1 *Community-based Relation Prediction*

Community-based Relation Prediction (CRP) exploits latent features of properties for predicting missing relations of a head entity in a knowledge graph. In our proposed solution, we distinguish two main stages: the *relation-centric stage* and the *prediction stage*. The *relation-centric stage* captures the latent features of the relations encoded in the KG. The outcome of this stage is then used in the *prediction stage* to predict missing relations of head entities. The proposed approach is illustrated in Figure 8.

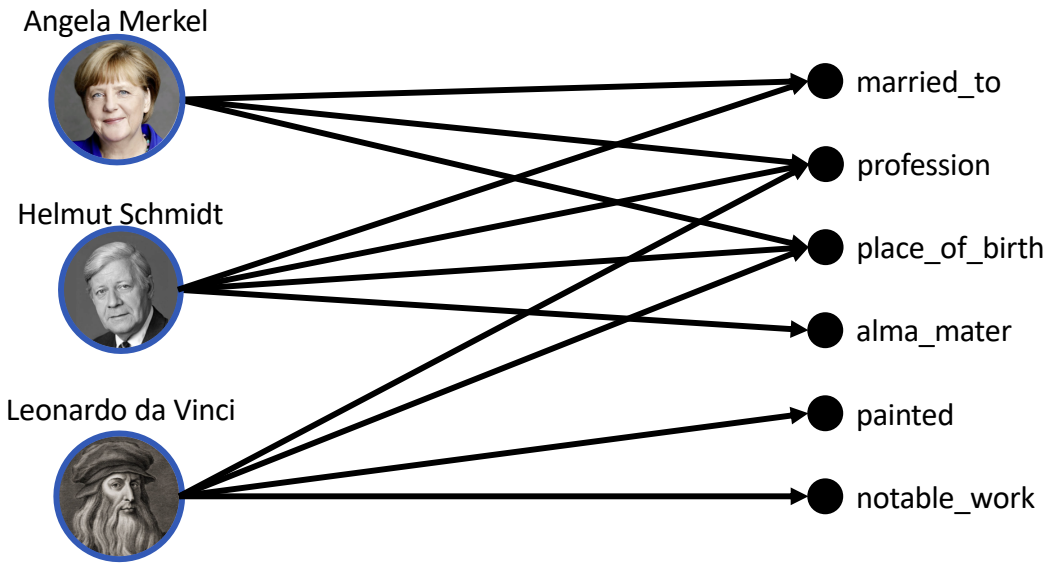


Figure 9: Head-Relation Graph of the running example.

Relation-Centric Stage

The objective of this stage is to identify groups of relations that are related based on the implicit knowledge encoded in KG. The output of this stage is a set of communities where each community represents relations that are highly associated according to KG. The input of the relation-centric stage is a knowledge graph KG. To get a better view on the co-occurrence of relations we transform KG into a bipartite graph. In this bipartite graph, nodes represent head entities and relations, while edges encode the head-relation interactions. We denote this graph the *Head-Relation Graph* and is formally defined as follows:

Definition 11: Head-Relation Graph

Assume a knowledge graph $KG = (H \cup T, R)$. A head-relation graph is a bipartite graph $I = (V, E)$, where $V = H \cup R$. H corresponds to the set of head entities, and R corresponds to the set of relations in KG. An edge $(h, r) \in E$ denotes that head node $h \in H$ interacts with relation $r \in R$ in KG.

Based on Definition 11, the bipartite nature of the head-relation graph I, $H \cap R = \emptyset$ obviously holds true. To illustrate the concept of a head-relation graph, consider the running example from Figure 7. We converted this subgraph into a head-relation graph, shown in Figure 9. The head-relation graph corresponds to head entities and their existing outgoing relations in the knowledge graph KG. The head entities represent nodes $H = \{$ *Angela_Merkel, Helmuth_Schmidt, Leonardo_da_Vinci* $\}$ and the relation nodes represent the relations of the knowledge graph, defined as $R = \{$ *married_to, profession, place_of_birth, alma_mater, painted, notable_work* $\}$.

We will identify highly-correlated relations, based on the head-relation graph I. To this end, we propose the application of frequent itemsets mining. Frequent itemsets approaches rely on transactions to identify items that highly co-occur. In our approach, the set of all relations of one head entity represents one transac-

tion. Therefore, all transactions can be determined by the union over the transaction of the individual head entities. A good side effect of using frequent itemsets, is the removal of noise in the data and filter out relations that occur very rarely. One important aspect to consider when applying frequent pattern mining is that many frequent patterns are not interesting and items cannot appear more than once in a transaction. This is for the usage of itemset mining in KGs not useful, since a head entity can have the same relation multiple times to different tail entities, e.g. *profession* and this might effect the computation of itemsets. At the same time some relations which occur very infrequent but are of high interest could be higher weighted than others that occur very frequently in a KG but are at the same time only of limited interest. Using the Apriori algorithm [6] would identify frequent itemsets, but could not overcome those limitations. To overcome those two limitations, the Fast Correlated High-Utility Itemset Miner (FCHM) [35] efficiently finds highly correlated itemsets, based on transaction data. FCHM prunes the space of candidates based on a bond measure, meaning that all itemsets that do not fulfill a minimum number of utility (*minutil*) and correlation (*minbond*) are not pursued further for future computations, resulting in an adequate runtime, even for very large graphs. The bond measure indicates how the items in a frequent itemset correlate and thus expresses the relative importance of a relationset [35, 85]. This method enables us to identify relations that correlate and therefore occur very frequently with each other. In addition, we can identify and remove relations that occur only very rarely in the KG. These low occurring relations are noise in the KG and due to their low occurrence provide only very little information for the prediction of relations.

The input for FCHM is a set of transactions. One transaction is the list of existing relations for a head entity h, i.e., R_h. Considering the head-relation graph I of the motivating example in Figure 9, the transaction for head entity *Angela_Merkel* is the following (*married_to*, *profession*, *place_of_birth*). Likewise, the transactions for the other head entities describe the existing relations outgoing from the head entity.

Having these transactions, we reduce noisy relations using FCHM by computing highly-correlated relationsets. The computation of highly-correlated relationsets, using a head-relation graph is formally defined as follows.

Definition 12: Highly-Correlated Relationsets

Let $I = (H \cup R, E)$ be a head-relation graph, $minbond \in \{x \in \mathbb{R} \mid 0 \leqslant x \leqslant 1\}$ be a minimum bond threshold and $minutility \in \mathbb{R}^+$ be a minimum utility threshold. The set of interactions is $D = \{T_1, T_2, \ldots, T_q\}$ where each element is a tuple $T_x := \big(X = \{r \in R \mid \exists h_x \in H, (h_x, r) \in E\}, |X|\big)$ containing the relations of head entity $h_x \in H$ and the number of the relations as utility.

FCHM receives $minbond$, $minutility$ and D as input parameters and returns the set $\mathcal{S}$. The set $\mathcal{S} = \{S_0, S_1, \ldots, S_m\}$ is a set of highly-correlated relationsets where $S_k \subseteq R$ and $B(S_k) \geqslant minbond$, for each $S_k \in \mathcal{S}$. $B(S_k)$ denotes the bond measure of the highly-correlated relationset S_k and is defined as follows:

$$B(S_k) = \frac{support(S_k)}{dissup(S_k)}$$

where

$$support(S_k) = |\{h \in H \mid \forall r \in S_k : (h, r) \in E\}|,$$

$$dissup(S_k) = \sum\nolimits_{r \in S_k} |\{h \in H \mid (h, r) \in E\}|.$$

The outcome of FCHM are sets of highly correlated relations. We call them relationsets. The returned relationsets in $\mathcal{S}$ are different in size and strongly overlapping. Thus, a relation $r \in R$ can occur in different relationsets. Due to the overlap and the differences in the sizes of these sets, the information from the relationsets will be grouped. To extract information from the relationsets, we will model the corresponding relations as nodes in an undirected, weighted graph, which we denote relation-bonding graph.

Definition 13: Relation-Bonding Graph

Let $\mathcal{S} = \{S_0, S_1, \ldots, S_m\}$ be a set of highly-correlated relationsets. An Relation-Bonding Graph is an undirected weighted graph $KG' = (R', E', w)$, where $R' = \bigcup_{S_k \in \mathcal{S}} S_k$, and for each $S_k \in \mathcal{S}$, $r_x, r_y \in S_k \Rightarrow (r_x, r_y) \in E'$. The weights w are defined as a function $w : R' \times R' \to \mathbb{R}$ and computed as the sum of the corresponding bond measure, i.e.:

$$w(r_x, r_y) = \sum_{r_x, r_y \in S_k} B(S_k).$$

The graph KG' contains the relations from the relationsets as nodes. The Relation-Bonding Graph for the running example is represented in Figure 10. The weights of the graph are the summed bond measure and the links represent the co-occurence in same frequent itemsets.

It should be noted that due to the computation of the highly-correlated relationsets, $R' \subseteq R$ applies, which means that not every item $r \in R$ of the original graph KG must also be represented in KG'. The highly-correlated relationsets are used to remove relations with only minor relative importance in the graph KG. Thereby the noise in the data is reduced. The edges of KG' represent the

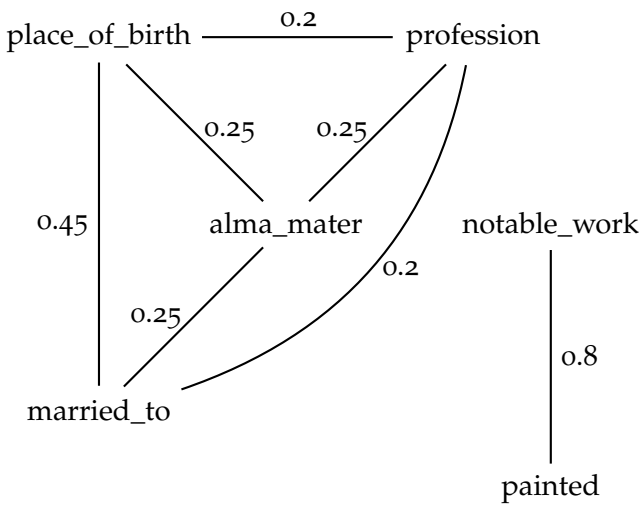


Figure 10: Relation-Bonding Graph for the running example representing the strength of the relationships among the relations.

common occurrence of relations in the same relationset. The weight of the edge between the relations is, as given in the above definition, the sum of all bond values of the relationsets in which both relations occur. The weight of the edge thus expresses the strength of its tie across all relationsets.

As last step to determine the latent features from the relations, we will use the information represented in the relation-bonding graph KG' to identify communities. A community is a set of nodes in a graph such that each node of the set is densely connected to each other node in the set. The identification of communities in KG' is used to group relations that are strongly related. In the case of KGs, relations of related information that often occur in common should be identified. There are many community detection algorithms, using different methods like e.g. minimum cut method or modularity maximization. Modularity maximization is a very prominent method to detect communities within graphs. Modularity is a unit of measurement of a network. The modularity describes the strength of a network by dividing it into communities. We chose Fastgreedy algorithm [22, 80] for detecting communities. This algorithm optimizes the metric modularity when discovering communities. A benefit is that there is no need to predefine the number of communities. This algorithm detects the best number of communities by itself. Fastgreedy is a non-overlapping community detection algorithm, which means that nodes in the graph are exactly assigned to one community. This method starts by assigning each node to its own community. It then computes the expected improvement of modularity for each pair of communities. The communities with the highest expected improvement of modularity are merged. This procedure is repeated until the merge of communities no longer leads to an increase in modularity. This procedure allows Fastgreedy to come up with results in a reasonable amount of time. This algorithm is therefore recommended for very large networks and thus used in the following.

By using the Fastgreedy algorithm, communities will be detected in the relation-bonding graph KG'. These communities represent a set of relations from KG'

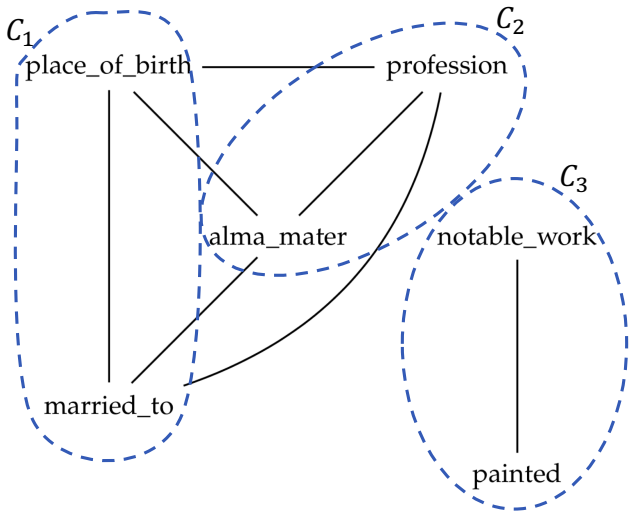


Figure 11: Detected communities using Fastgreedy. In total three communities, representing latent groups of highly correlated relations.

that have a high density, with only a few connections to the other communities. The communities are called relation communities and are defined as follows:

Definition 14: Relation Community Set

Let KG' be a relation-bonding graph. A relation community set is denoted $\mathcal{C} = \{C_1, C_2, \ldots, C_p\}$, where $C_j \in \mathcal{C}$ is a relation community defined as a dense sub-graph of KG', and $C_i \cap C_j = \emptyset$, for each $C_i, C_j \in \mathcal{C}$.

The communities are clusters of relations which are strongly related to each other. The communities represent latent features, which we mined from the KG. Frequently co-occurring relations are grouped into the same communities. We will see later in the experiments an excerpt for the clustered relations from a KG. Given the running example, Figure 11 represents the detected communities in the head-relation graph. In the running example, three latent groups are detected using Fastgreedy, showing meaningful latent associations. With the detection of the relation communities, the first stage, the relation-centric-stage, of the presented approach, is completed. In the second stage, the prediction stage, the focus is on using the information obtained to predict missing relations for head entities in the knowledge graph KG.

Prediction-Centric Stage

We use the information from the knowledge graph KG and the information we have mined from it and encoded in the relation community sets $\mathcal{C}$, to predict missing relations of head entities. In general, the number of possible relation candidates for predicting is, depending on the number of relations in the KG, usually very high. Therefore, in the following, we reduce the number of possible relations for predicting. For this, we use the information from the community sets $\mathcal{C}$. We compute for a head entity h the relative number of its existing relations in the knowledge graph KG to each community set. We sort the results in

Table 1: For a given head entity, the selection of suitable relations for predictions is based on relative number of existing relations to each community set.

Community	**Relative Frequency**
Community C_1	$\frac{2}{2}$
Community C_2	$\frac{1}{2}$
Community C_3	0

descending order. An exemplary sorted table for a given head-entity is given in Table 1.

We select the first community set with the highest relative frequency, unless the relative frequency is one. A relative frequency of one for a community means that the head entity h is already complete with respect to the relations from this community. Possible candidates for missing relations of a head entity h are now all relations in this community set that the entity h does not already have. In the case of Table 1, there are three relations missing in C_1. In mathematical terms, this means that, starting from a fixed h and C_i, we check the following relations as possible candidates for prediction: $R_{cand} = \{r \mid r \in R : r \in C_i \wedge \neg\exists t \in T : (h, r, t) \in G\}$. For each of these candidates we compute a confidence of prediction. A confidence gives the user the opportunity to assess the reliability of the prediction. It also serves to rank the predictions. For the computation of confidence we have tried different metrics. We first computed the entropy of the relations. In conducted experiments, it turned out to be an insufficient metric. One metric that has proven to be very robust is the usage of probable occurrence of a relation. The confidence for predicting a relation $r \in R_{cand}$ for head entity $h \in H$ is computed as follows:

$$conf(h, r) = \frac{|\{h_j \mid h_j \in E \wedge \exists t \in T:(h_j,r,t) \in G \wedge \exists r_k \in R, r_k \neq r \exists s,t \in T:(h_j,r_k,s) \wedge (h,r_k,t)\}|}{|\{h_j \mid h_j \in H \wedge \exists t \in T:(h_j,r,t) \in G\}|}$$

The above formula for computing the confidence divides the number of head entities that have relation r and share at least one relation with h by the number of entities that have relation r. We compute for each relation in R_{cand} its confidence and use the top-k relations as predictions.

3.3.2 *Link Distribution Learning*

Based on the insights of the previous section, more precisely on the finding that a few relations often occur in common, i.e. they correlate, and can thus be grouped into latent communities, we have taken a closer look at the distribution of the use of relations of the head entities. When looking at the usage of relations of head entities, it becomes apparent that similar head entities have a similar distribution of property usage. In order to illustrate this in more detail, consider the following multilabel encodings, which describe head entities in terms of their usage of relations. Next to the first vector, *Angela_Merkel*, the corresponding relations for this entry are listed.

$$o_{\text{Angela\_Merkel}} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \textit{married\_to} \\ \textit{profession} \\ \textit{place\_of\_birth} \\ \textit{alma\_mater} \\ \textit{painted} \\ \textit{notable\_work} \end{matrix} \quad o_{\text{Helmut\_Schmidt}} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$o_{\text{Leonardo\_da\_Vinci}} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Figure 12: Binary Vectors representing entities of the running example.

The above example indicates that similar head entities such as *Angela_Merkel* and *Helmut_Schmidt* also have a similar distribution in the usage of relations. The assumption is that the relation *alma_mater* is missing in *Angela_Merkel*, since this head entity uses similar relations as *Helmut_Schmidt*, but the relation *alma_mater* is missing. In the case of *Leonardo_da_Vinci* there is a very different distribution in the usage of the relations, though this head entity is significantly different from the two above. Thus, in the case of *Leonardo_da_Vinci* it is not suspected that the relation *alma_mater* is missing for this entity. In the following we would like to use this characteristic that similar head entities have a similar distribution of the used relations for our method Link Distribution Learning (LDL) for predicting missing relations of head entities.

For this, a Restricted Boltzmann Machine (RBM) is used as a basis for determining the target distribution of the usage of relations of the head entities. RBMs are energy-based models representing a stochastic neural network with a visible layer and a hidden layer. Unlike feed-forward networks, RBMs are undirected neural networks, which attempt to restore an input after compression. A further difference to regular neural networks is the use of stochastic units in the hidden layer, in which a distribution function is used in addition to an activation function. When applying RBMs, the reconstructed input, which corresponds to the output of the network, should follow the same distribution as the input. Since the goal of the network is not to reconstruct the input exactly, but to determine the distribution of the usage of relations, this is not a supervised problem, but an unsupervised problem. Thus there are no labels available for computing the error. We will later go into more detail about the description of the model and the reconstruction of the input, as well as about learning the parameters of the neural network. But first we have to define a suitable representation of the input features for the stochastic neural network.

Input

As in any neural network, the input for the RBM must be numerical. We define the input as $\mathbf{v}$ an index it with zero ($\mathbf{v}_0$) indicating that this is the initial input. Gibbs Sampling, which is an iterative method for determining a distribution, is later used when learning the parameters. Therefore, $\mathbf{v}_0$ is used to indicate the initial input to the model. The goal is to use the model for predicting missing relations for a head entity. A head entity can have the same relation only to different tail entities such as for the relation *profession*. Therefore, a head entity such as *Angela_Merkel* may have two professions with different tail entities according to their professions. However, we are not interested in predicting the amount of missing relations to tail entities for a particular head entity and relation, but rather new unknown relations that are not yet represented in the knowledge graph. As in our running example we want to predict if *alma_mater* is a missing relation of *Angela_Merkel*, since exactly this relation is not available for the head entity *Angela_Merkel* in the knowledge graph. Consequently, we are not interested in finding out whether *Angela_Merkel* has other profession relations to other tail entities, since we already know from the knowledge graph that this entity has at least one *profession* relation, but finding out new unknown relations. For the representation of a head entity it is therefore necessary to specify which relation is used and which is not. For this reason, we use multi-label encoding for the representation of relations of a head entity. We define the representation of a head entity as follows.

Definition 15: Head Entity Representation

Assume a knowledge graph $KG = (H \cup T, R)$. A head entity $h \in H$ is represented as a binary vector, denoted as o_h. The length of the vector is defined by the number of relations in KG: $|o_h| = |R|$. For each relation, the binary vector o_h indicates the presence of a relation for the head entity h.

$$o_{h[r]} = \begin{cases} 1, & \text{if } r \in R_h \text{ for head entity } h \\ 0, & \text{otherwise} \end{cases}$$

Following the above Definition 15, we can represent any head entity as a vector of its existing relations. Thus we get the binary vectors for the running example, represented in Figure 12.

Using these vectors as input for the machine learning model, the distribution function of the used relations can be identified to determine missing relations. As already described in Definition 15, the length of the vectors is determined by the number of relations in the knowledge graph. Depending on the number of existing relations in the knowledge graph the number of dimensions of the head entities can become very large, which leads to the conclusion that there might be an increased memory consumption[10]. However, these are binary vectors, meaning that each entry in the vector can be represented by a single bit. Overall, the memory consumption is therefore relatively low, even with many

[10]The knowledge graph DBPedia_2016-10 has about 100,000 relations

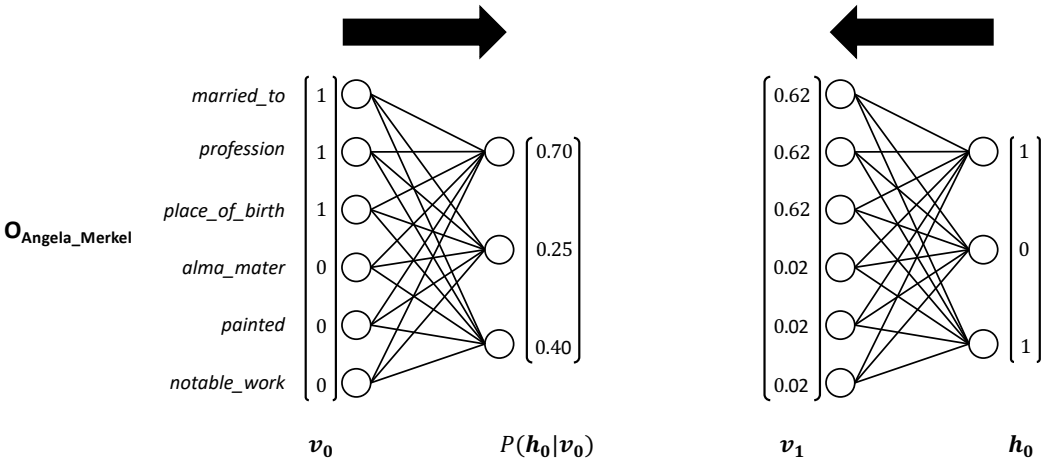


Figure 13: Left is the input for the head entity *Angela_Merkel* of the running example. Based on this input the probability $P(\mathbf{h}_0|\mathbf{v}_0)$ to activate the hidden state is computed. On the right side the hidden state was determined based on $P(\mathbf{h}_0|\mathbf{v}_0)$ and the Bernoulli distribution. Two neurons have been activated, representing two latent features which are fed back for reconstructing the input. The goal is to make the distribution of the reconstruction equal to that of the original input.

available relations in knowledge graphs, and can be processed efficiently with modern computers.

We have defined a meaningful and efficient representation of the head entities, which is used as input for the machine learning model. In the following we will introduce the model that is used to determine the target distribution based on the input in order to predict missing relations.

Model

As described above, a Restricted Boltzmann Machine (RBM) is used as basis for determining the target distribution. An RBM is an unsupervised machine learning method, consisting of an input layer and a hidden layer. The number of neurons in the hidden layer is a positive integer, greater than zero, and can be arbitrarily defined apart from this restriction. The number of input neurons is defined by the number of dimensions of the input vector. Unlike common feed-forward networks, an RBM is not a feed-forward network, but an undirected, generative stochastic artificial neural network. The two main differences are that it tries to restore the input, using the same weights to compute the hidden layer, and using a stochastic distribution function in the hidden layer. The model is shown in Figure 13 and illustrates the steps in the computations. In the following, the individual steps are described in more detail.

Restricted Boltzmann Machines (RBMs) are Energy Based Models which consist out of one visible layer and one hidden layer. As described above the input is called $\mathbf{v}_0$ and the hidden state is called $\mathbf{h}_0$. The indexing indicates in which step the data is currently in the process of reconstruction. The indexing is essential as the Gibbs Sampling method is used later for learning the parameters, which is an iterative process to determine a distribution. Hence, based on the notation, $\mathbf{v}_0$ is the initial input and $\mathbf{v}_1$ is the input restored from $\mathbf{v}_0$. Within Gibbs sampling,

$\mathbf{v}_1$ itself can be used as an input to further train the model. However, we will only perform one Gibbs sampling step, since related work suggest that one Gibbs sampling step is already sufficient for training a model [32]. Later, as we describe the learning of the parameters, we will discuss the Gibbs sampling technique and the reasons for using a single Gibbs sampling step. The model does not have an explicit output layer, since it tries to restore the distribution of the data in an unsupervised manner.

Energy based probabilistic models use a probability distribution through an energy function to measure the quality, similar to cost functions of machine learning models. Having the hidden layer as latent variables to increase the expressive power of the model, we get the following energy-based probabilistic function (Gibbs distribution) which denotes that a certain state $\mathbf{v}$ can be observed:

$$P(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{1}$$

where Z is the sum from all possible states and called the normalizing factor:

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{2}$$

Having a low energy $E(\mathbf{v}, \mathbf{h})$ leads to a high probability, whereas a high energy translates to a low probability. The goal of the model is to increase the probability of the energy-based probabilistic function $P(\mathbf{v})$, leading to the goal of minimizing the energy function $E(\mathbf{v}, \mathbf{h})$ to obtain a high probability $P(\mathbf{v})$. The energy function $E(\mathbf{v}, \mathbf{h})$ for an RBM with its input $\mathbf{v}$ and hidden state $\mathbf{h}$ is the following:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} \tag{3}$$

$\mathbf{W}$ represents the weight matrix between visible layer and hidden layer. $\mathbf{a}$ and $\mathbf{b}$ each represent the bias of the respective layer.

In the first step the multi-label representation of a head entity is used as input to determine the hidden state $\mathbf{h}_0$. Using the input $\mathbf{v}_0$ and the parameter $\mathbf{W}$, the probability of activation of the hidden state $\mathbf{h}_0$ is computed as follows, using k as index for representing the current gibbs sampling step:

$$P(\mathbf{h}_k|\mathbf{v}_k) = \sigma(\mathbf{W}\mathbf{v}_k + \mathbf{b}) \tag{4}$$

In this context, σ represents the sigmoid activation function, which is defined as follows: $\sigma(x) = \frac{1}{1+e^{-x}}$. The sigmoid activation function has the advantage that its output can be interpreted as probability as the sigmoid activation function returns results in the range $(0, 1)$. Thus, the output of the sigmoid function, $P(\mathbf{h}_0|\mathbf{v}_0)$, can be interpreted as the probability of a hidden state being activated based on the input $\mathbf{v}_0$. The disadvantage of using the sigmoid activation function is that the derivation for very large and very small input values is very small, meaning that when applying the gradient descent method, very small parameter updates may occur. If the input values become very large, there is even the danger of dying neurons, as the parameters are hardly adjusted to adapt the

output of the sigmoid function according to the desired direction given an input. If the input values are in the range of $[-1, 1]$ this is a negligible problem, as in this case the sigmoid function has a suitable gradient to adjust the parameters quickly using the gradient descent method, thus the gradient descent approach converges quickly and does not run into the problem of dying neurons.

Using the running example as illustration, Figure 13 shows in the left panel the computed probability to activate the hidden states $P(\mathbf{h}_0|\mathbf{v}_0)$, based on the input $\mathbf{v}_0$. The computation is performed as defined in Equation 4. Given the running example, the model has three neurons in the hidden layer, which obtain the following probability $P(\mathbf{h}_0|\mathbf{v}_0)$ $(0.70, 0.25, 0.40)^T$. Thus the probability that the first hidden state is activated is 70%, the second hidden state is 25% and the last hidden state is 40%. Based on the computed probabilities $P(\mathbf{h}|\mathbf{v})$, we sample the values $\mathbf{h}_0$ based on a Bernoulli distribution function. In general, the choice of the distribution function in the hidden layer is arbitrary, but has a great impact on the result of the model. However, the model presented in this chapter, and thus used for predicting relations in knowledge graphs, uses only the Bernoulli distribution. This will be relaxed in the next chapter, considering further distribution functions for the hidden layer in the model. Using the Bernoulli distribution function leads to Bernoulli sampling, meaning that values are sampled randomly based on the Bernoulli distribution. The advantage of using the Bernoulli distribution function is that each sample has the same probability. We define $Bernoulli_X(P(\mathbf{h}_0|\mathbf{v}_0))$ as Bernoulli sampling, which samples the hidden state $\mathbf{h}_0$ based on the Bernoulli distribution, the stochastic variable X and the probabilities $P(\mathbf{h}_0|\mathbf{v}_0)$.

$$\text{Bernoulli}_X(P(\mathbf{h}_0|\mathbf{v}_0)) = \begin{cases} 0 & \text{if } P(\mathbf{h}_0|\mathbf{v}_0) < X \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

Introducing a stochastic distribution function extends the neurons to stochastic neurons. The Bernoulli distribution only has the values 0 or 1. The binary values of the neurons for the hidden states $\mathbf{h}_0$ are obtained by sampling from a Bernoulli distribution $Bernoulli_X$ using the probability $P(\mathbf{h}_0|\mathbf{v}_0)$. A high $P(\mathbf{h}_0|\mathbf{v}_0)$ leads to a high probability of having a positive hidden state $\mathbf{h}_0$, whereby a low probability leads to a zero output. Thus, as described above, the output of the sigmoid function $P(\mathbf{h}_0|\mathbf{v}_0)$ can be interpreted as probabilities of activating the hidden state, since the binary vector of the hidden state can either be activated or not activated, depending on $P(\mathbf{h}_0|\mathbf{v}_0)$. Each neuron can be considered as a latent feature, which is activated once the head entity has a certain combination of relations. If the combinations of relations of a head entity are not sufficient to result in a sufficiently high probability of $P(\mathbf{h}_0|\mathbf{v}_0)$, then the corresponding neuron in the hidden state is not activated. The hidden state can, therefore, be interpreted as an activation of the latent features, based on the input $\mathbf{v}_0$, of the hidden layer. This is illustrated in the right part of the Figure 13 for the running example. It illustrates the sampling, which determines the hidden states based on the Bernoulli distribution. A random variable $X \in [0, 1]$ is generated for each neuron. If the probability of the hidden state $P(\mathbf{h}_0|\mathbf{v}_0)$ is less than the random variable X, the neuron is not activated by setting it to the value 0, otherwise it is activated

by setting it to the value 1. Using this example it is obvious that the output of the sigmoid function $P(\mathbf{h}_0|\mathbf{v}_0)$ can easily be interpreted as the probability of the neuron being activated. Based on the vector $(0.70, 0.25, 0.40)^T$ and the Bernoulli distribution the hidden state $h_0 = (1, 0, 1)^T$ was sampled.

The hidden state $\mathbf{h}_0$ describes which latent features of an input $\mathbf{v}_0$ are activated. Based on these latent features the input is reconstructed. When reconstructing the input, the same weight matrix $\mathbf{W}$ is used, but transposed to ensure the computation due to the dimensions. Following this, the hidden state is multiplied by the transposed weight matrix $\mathbf{W}$ and added up with a bias $\mathbf{b}$. Afterwards an activation function is applied, which is a sigmoid activation function in the presented model. Since the input is a binary vector, and our goal is to reconstruct the vector, applying a sigmoid function is useful, since its output is in the range $(0, 1)$ and can be interpreted as probabilities. The output in the visible layer can be seen as reconstruction or approximation of the original input. The formula for reconstructing $P(\mathbf{v}_1|\mathbf{h}_0)$ is the following:

$$P(\mathbf{v}_1|\mathbf{h}_0) = \sigma(\mathbf{W}^T\mathbf{h}_0 + \mathbf{b}) \tag{6}$$

The reconstruction of the input $\mathbf{v}_0$ is denoted as $\mathbf{v}_1$ and corresponds to the computation of $P(\mathbf{v}_0|\mathbf{h}_0)$ in Equation 6. The reconstruction $\mathbf{v}_1$ describes the probability that a relation for a head entity should be present. This is determined by the relations described in the input vector of the head entity $\mathbf{v}_0$ and the latent features in the hidden layer. The target distribution is specified in the model using the above equations. In Figure 13 the reconstruction $\mathbf{v}_1$ in the right part is exemplarily indicated with $(0.62, 0.62, 0.62, 0.02, 0.02, 0.02)^T$. According to the interpretation of the output as probabilities there is a 62% likelihood that the entity *Angela_Merkel* has the relation *married_to*. Considering its actual relations, we see that the entity *Angela_Merkel* already has the relation *married_to*. Thus, the model is not yet fully fitted, since this probability should be much higher. Therefore, the parameters need to be adjusted in order to achieve a better reconstruction and thus a better distribution. Based on the reconstruction $\mathbf{v}_1$ of the input $\mathbf{v}_0$ the error is now computed and the parameters are adjusted to reduce the error. We will explain this learning in more detail below.

Learning

In supervised learning procedures, a loss function is used to determine the error between the prediction and the actual value, and the parameters of the model are adjusted on the basis of this error. The challenge in the present model is that it is an unsupervised learning problem, i.e. no labels are available for computing the error and in particular the input values $\mathbf{v}_0$ should not be reconstructed exactly, but the goal is to learn a target distribution in order to be able to reproduce the distribution of the input data. For this reason, we will not use a loss function such as categorical crossentropy based on the above energy function $E(\mathbf{v}_0, \mathbf{h}_0)$, but rather, due to our unsupervised problem, we will use another loss function to minimize the error. We are interested in determining the error in the distribution between the actual and reconstructed input values. The idea is that we create the reconstructed values using samples from the model distribution.

In stochastics, Markov chains [37] are used for an infinite time to ensure stationarity and thus convergence to the actual distribution. However, this is very time consuming due to infinite time and therefore not suitable in practice. Counteracting this there is the possibility of contrastive divergence (CD-k), in which instead of sampling from the RBM distribution, a Gibbs chain for only k steps is run. Contrastive divergence is an approximation of the log-likelihood gradient that has been found to be a successful update rule for training RBMs.

The use of Contrastive Divergence (CD) to approximate the gradients has become a standard for training RBMs [31]. CD is based on Gibbs chain in which k denotes the length of a single sample of the Gibbs chain. It has been shown in the literature and later in our experiments that Gibbs chain is sufficient to achieve very good results at $k = 1$. We will therefore also apply k-step contrastive divergence (CD-k) with $k = 1$ for learning the parameters of the presented Model. Using the indexing of the input v and hidden states h introduced above, we can identify which Gibbs step the model is in. After one iteration of the Gibbs chain, the error between the distribution of the reconstructed data and the actual distribution of the input is determined. Based on this error, the parameters of the model are adjusted to reduce the error. The time complexity of CD is $O(n^2)$, where n is the number of neurons in the hidden layer. This time complexity is due to the vector multiplication which requires $O(n^2)$ and at first appearst to be a poor complexity, but it is by far better than related work.

As described above, the input to our model is described by $\mathbf{v}_0$, the reconstruction using CD-k with $k = 1$ and thus the output is denoted as $\mathbf{v}_1$ and is computed by $P(\mathbf{v}|\mathbf{h})$. As loss function we use the deviation of the energy function $E(\mathbf{v}, \mathbf{h})$ between the input and its reconstruction. To simplify this expression even further we use the log-likelihood. The gradient w.r.t. log-likelihood for one training sample $\mathbf{v}_0$ is then approximated by the following formula [16].

$$CD(W, \mathbf{v}_0) = -\sum_{\mathbf{h}} P(\mathbf{h}_0|\mathbf{v}_0)\frac{\partial E(\mathbf{v}_0, \mathbf{h}_0)}{\partial W} + \sum_{\mathbf{h}} P(\mathbf{h}_0|\mathbf{v}_1)\frac{\partial E(\mathbf{v}_1, \mathbf{h}_0)}{\partial W} \tag{7}$$

The left part of the equation denotes the actual distribution of the data while the right part denotes the distribution of the reconstructed input $\mathbf{v}_1$, computed by our Machine Learning model. The difference between these two distributions should be minimized and is our loss function for our model. Having this loss function, we can apply learning algorithms, i.e. Gradient Descent and its extension like e.g. Adam Optimizer [51], for adapting the parameters in the model. The basic equation for adapting a parameter $\mathbf{W}$ using the Gradient Descent Approach is the following:

$$\mathbf{W} = \mathbf{W} - \alpha\Delta\mathbf{W} \tag{8}$$

$$\Leftrightarrow \mathbf{W} = \mathbf{W} - \alpha\frac{\partial CD(W, \mathbf{v}_0)}{\partial \mathbf{W}} \tag{9}$$

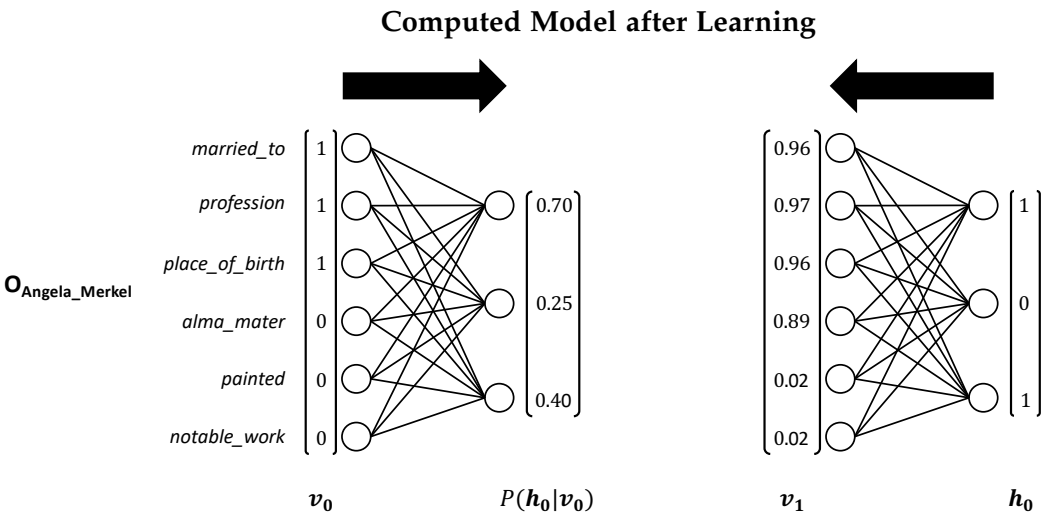


Figure 14: Left is the input for a the head entity *Angela_Merkel* of our running example. Based on this, the probability $P(\mathbf{h}_0|\mathbf{v}_0)$ to activate the hidden state is computed. On the right side the hidden state was determined based on $P(\mathbf{h}_0|\mathbf{v}_0)$ and a Bernoulli distribution. The hidden state is fed back to reconstruct the input. The reconstruction $\mathbf{v}_1$ corresponds to the distribution of relations based on the input and is used to predict missing relations.

Based on the Gradient Descent Approach and the Equation 8 we get the following updates of the parameters:

$$\Delta \mathbf{W} = P(\mathbf{h}_0 = 1|\mathbf{v}_0) \cdot \mathbf{v}_0 - P(\mathbf{h}_0 = 1|\mathbf{v}_1) \cdot \mathbf{v}_1 \tag{10}$$

$$\Delta \mathbf{a} = \mathbf{v}_0 - \mathbf{v}_1 \tag{11}$$

$$\Delta \mathbf{b} = P(\mathbf{h}_0 = 1|\mathbf{v}_0) - P(\mathbf{h}_0 = 1|\mathbf{v}_1) \tag{12}$$

We use mini-batch with Adam Optimizer [51] to update the parameters. The parameters converge so that the distribution of the reconstructions $\mathbf{v}_1$ corresponds to the distribution of the input $\mathbf{v}_0$.

Repeating the learning for a large number of iterations minimizes the error. Based on the learning method, the parameters of the model are adjusted in such a way that the distribution of the reconstructed input converges to the distribution of the input. Figure 14 shows the trained model and the reconstruction of the input. Based on many head entities and their distribution on using the relations and several iterations, the reconstructed input from the head entity *Angela_Merkel* is as follows: $(0.96, 0.97, 0.96, 0.89, 0.02, 0.14)^T$. The distribution of this reconstruction corresponds to the distribution of the head entities, which use similar relations as the head entity *Angela_Merkel*. Based on this reconstruction, we will show below how we predict missing relations in the knowledge graph for a given head entity.

Predictions
The reconstruction of the input is used to predict missing relations of a head entity. Since the reconstruction $\mathbf{v}_1$ follows the distribution of the use of relations in a knowledge graph the assumption is that variations from it are based on missing relations. We therefore use for a given head entity its reconstruction $\mathbf{v}_1$ for predicting missing relations of the head entity. Therefore, the input $\mathbf{v}_0$ is processed according to the equations described above to obtain the reconstruction $\mathbf{v}_1$. As a result of using the sigmoid function in the visible layer, the reconstruction, $\mathbf{v}_1$, can be interpreted as probabilities for the presence of relations. Based on the reconstruction of a head entity and the reconstruction of its used relations, we exclude all already known relations in the prediction, since we are only interested in the unknown relations. The remaining relations can then be sorted according to the probability $P(\mathbf{v}_1|\mathbf{h}_0)$ to obtain, for example, the relation with the highest probability. According to the example in Figure 14 we get a probability of 89% that the relation *alma_mater* for the head entity *Angela_Merkel* is missing.

3.4 EXPERIMENTAL STUDY

In the following we will evaluate our two introduced approaches with respect to their effectiveness in predicting missing properties on twelve knowledge graphs. The evaluation is based on well-known metrics, which are commonly used in related work. By using known metrics of related work, we enable the comparability of our method. After the presentation of the experimental setup we will present the results. Based on the results, we will then critically evaluate the two introduced methods in a discussion.

3.4.1 *Experimental Setup*

Datasets. We use twelve knowledge graphs for the evaluation of our methods, containing information from different domains. Among them are FB15k which is a subgraph of Freebase [19] and contains cross-domain knowledge, and WN18 which is a knowledge graph based on WordNet [76]. Both were introduced in 2013 by Bordes et al. [20]. However, in both of these datasets a large test leakage was found due to inverse relations in the datasets. Thereby, a large part of the triples of the test set can be generated correctly by inverting the triples from the training dataset. Therefore Toutanova et al. introduced the dataset FB15k-237 [113], representing a subgraph of FB15k in which inverse relations have been removed. Similar to Toutanova et al, Dettmers et al. introduced the graph WN18RR, which contains 11 relations, no pair of which is reciprocal. These four datasets are frequently used in related work to evaluate link prediction methods, and are therefore also used in this study in order to enable greater comparability with related work. In recent years the non-bias datasets FB15k-237 and WN18RR have been used in particular. In addition to these records, we have extracted subgraphs from DBpedia (DBp) [9] and Wikidata (WD) [117] which do not contain cross-domain knowledge, but information on specific topics. These subgraphs contain facts about entities in the classes Person (Pers), Company (Comp), Movie (Mov) and Songs. For each knowledge graph, the splitting into

Table 2: Overview of the knowledge graphs and the experimental configurations. At the top of the table is a summary of the characteristics and at the bottom of the table are the parameters used to compute the communities for the CRP approach.

Metric	FB15k	FB15k-237	WN18	WN18RR	Pers(DBp)	Pers(WD)	Comp(DBp)	Comp(WD)	Mov(DBp)	Mov(WD)	Songs(DBp)	Songs(WD)
#Entities	14,951	14,541	40,943	40,943	229,613	190,419	63,545	10,925	231,637	287,775	39,619	126,606
#Relations	1,345	237	18	11	2,239	1,509	1,189	304	959	382	332	321
#Train	483,142	272,115	141,442	86,835	313,296	229,059	142,887	12,103	396,834	390,295	95,833	184,542
#Valid	50,000		17,535 5,000	3,034	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000
#Test	59,071	20,466	5,000	3,134	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000
minbond	0.1	0.1	0	0	0.1	0.1	0.1	0.1	0.1	0.3	0.1	0.1
minutility	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

training, validation and test data is already available. By the variation of different topics like cross-domain knowledge but also specific facts of individual classes we get a good insight into our methods how they perform on different knowledge graphs. An overview of the knowledge graph is given in Table 2.

Metrics. Following related link prediction models on knowledge graphs [20, 115], we use Hits@k (with $k = 1, 3$, and 10) as evaluation metric. Hits@k measures the proportion of correct relations in top-k ranked relations.

Silver Standards. We call the test dataset silver standard, as the test sets may suffer from incompleteness originated in the KG, thus, creating spurious false positives. In other words, a prediction may be correct but the relations might be missing in the KG and hence in the test set. This is a common problem in link prediction methods under the OWA. Therefore, evaluating a link prediction method using a silver standard is very common under OWA. We will discuss in detail the predictions of missing relations for individual knowledge graphs in the experiments and explain the reasons for which some predictions make sense but are classified as false due to the silver standard and, therefore, reduce the performance with respect to the evaluation metric.

Baselines. We compare our approaches *Community-based Property Prediction* (CRP) and *Link Distribution Learning* (LDL) with well-known state-of-the-art models in Link Prediction on Knowledge Graphs. We chose TransE [20], as it was one of the first methods to transform entities and relations into low-dimensional vector spaces and use them to predict missing entities and relations. Additionally, this method scales very well to large datasets. Besides TransE we use ComplEx [115] as another baseline method. This method extends TransE by using the complex number and can therefore represent symmetry/antisymmetry relations better than TransE. In both methods subject as well as object entity information are used to predict missing relations. The comparison with the presented methods is therefore not quite possible, but we still want to compare them to see how well our methods perform compared to methods that have full information. For the reason of insufficient comparability, we have also decided not to introduce any more recently published work as a baseline.

Preprocessing. For the DBpedia and Wikidata subgraphs, we removed regularly appearing relations for all head entities, e.g., *wikiPageID*, *wikiPageRevisionID*, and

P31.[11] We removed them for making predictions more challenging by deleting regularly occurring relations. Otherwise, our approach would have and increase in performance, because if such regularly appearing relations are missing, then it would be an easy guess.

Implementation. We implemented our method in Python3. We performed hyperparameter tuning on the validation dataset. The hyperparameter settings for CRP can be found in Table 2. For LDL, we chose for each dataset a learning rate $\alpha = 0.01$ and $1,000$ iterations. We used for all datasets 50 hidden units for LDL, except for WN18 and WN18RR in which the number of hidden units was set to 2. The experiments were performed on a server with Intel(R) Xeon(R) Gold 6142 CPU@ 2.60GHz, 32 physical cores and 188GB RAM.

3.4.2 *Performance Analysis*

We will first discuss the performance of our methods compared to the baseline models using the metric Hits@k. The results of the methods are shown in Table 3. While comparing the performance of the different methods, the first thing to notice is the lack of a definitive winner. Depending on the dataset and metrics, different methods perform better or worse. Both CRP and LDL on the cross-domain knowledge graphs FB15k-237 and FB15k do not perform as well as the baseline methods on Hits@1 and Hits@3. The reason for the low performance of CRP is the large impact of community size on the Hits@k metric. We will look at this impact in more detail later. Considering Hits@10, LDL performs much better on these two datasets than CRP. Considering the knowledge graph WN18RR, we see that LDL performs better than the baseline methods, but only from Hits@3 onwards, while at Hits@1 the performance of both models, LDL and CRP, does not match the performance of ComplEx. We will discuss this later in detail and show that the predictions at Hits@1 are very good, but are classified as false due to the silver standard, causing the Hits@k metric to suffer, which is particularly noticeable for the metric Hits@1. However, it should be noted that our models only use the head information to predict missing relations while the baseline models use both head entity and tail entity information to predict missing relations. Considering this fact, the achieved results are remarkable. Considering Hits@3, we see an improved performance, especially of LDL, compared to Hits@1 across all datasets. In general, one can see a significant performance increase of both methods between Hits@1 and Hits@3. This performance increase is much stronger for our methods than for the baseline methods. This increase in performance is particularly noticeable between Hits@3 and Hits@10, and especially noticeable for LDL, so that LDL performs better on most knowledge graphs with respect to the metric Hits@10 than the other methods. This is especially remarkable when considering that this method only uses information about the head entity, but no tail information as with TransE and ComplEx. Both of our methods perform very well on several DBpedia and Wikidata subgraphs, such as Pers(DBp), and can exceed the baseline methods, but cannot exceed the baseline models on the

[11] These are DBpedia- and Wikipedia-specific relations to denote information about the Wikipedia page and the class of an entity, respectively.

Table 3: Comparison of our approach with state-of-the-art algorithms. Our approaches (LDL and CRP) uses the head entity to predict missing relations. The compared methods uses head and tail entity to predict missing relations.

	Hits@1				Hits@3				Hits@10			
KG	LDL	CRP	TransE	ComplEx	LDL	CRP	TransE	ComplEx	LDL	CRP	TransE	ComplEx
FB15k-237	0.268	0.237	0.666	**0.699**	0.577	0.231	0.764	**0.878**	0.827	0.285	0.841	**0.947**
FB15k	0.181	0.389	**0.667**	0.519	0.427	0.473	**0.885**	0.800	0.772	0.698	**0.974**	0.940
WN18RR	0.655	0.484	0.220	**0.716**	**0.838**	0.518	0.447	0.833	**1.000**	0.678	0.966	0.979
WN18	0.645	0.561	0.924	**0.945**	0.812	0.650	0.974	**0.986**	0.946	0.900	**0.997**	0.995
Pers(DBp)	**0.475**	0.438	0.085	0.085	**0.622**	0.490	0.149	0.233	**0.782**	0.655	0.246	0.292
Pers(WD)	0.270	0.253	**0.328**	0.273	**0.468**	0.254	0.431	**0.468**	**0.663**	0.367	0.517	0.618
Comp(DBp)	0.309	0.275	0.185	**0.319**	0.595	0.345	0.326	**0.699**	**0.832**	0.580	0.452	0.780
Comp(WD)	0.415	**0.635**	0.483	0.008	0.600	**0.647**	0.603	0.017	**0.776**	0.674	0.692	0.058
Mov(DBp)	0.244	0.393	**0.453**	0.106	0.494	**0.615**	0.515	0.222	0.847	**0.900**	0.582	0.347
Mov(WD)	0.373	**0.471**	0.383	0.205	**0.662**	0.567	0.450	0.424	**0.879**	0.833	0.527	0.553
Songs(DBp)	0.349	0.398	**0.444**	0.409	0.696	0.498	**0.898**	0.736	0.924	0.811	**0.980**	0.887
Songs(WD)	0.296	0.488	**0.788**	0.203	0.542	0.654	**0.941**	0.359	0.819	0.825	**0.986**	0.452

Table 4: Overview of the structure of determined communities for the studied KGs.

Metric	FB15k	FB15k-237	WN18	WN18RR	Pers(DBp)	Pers(WD)	Comp(DBp)	Comp(WD)	Mov(DBp)	Mov(WD)	Songs(DBp)	Songs(WD)
Relations	86.47%	86.08%	100%	100%	72.53%	84.84%	78.47%	71.38%	51.02%	37.07%	77.27%	44.59%
Communities	110	24	3	2	244	76	77	28	104	31	29	31
mean Com.	10.57	8.5	6	5.5	6.52	16.58	12.12	7.75	4.55	4.48	8.21	4.52
std. Com.	15.49	9.165	4	0.707	16.9	44.83	39.53	10.30	4.32	8.60	10.28	4.68
max Com.	71	30	10	6	213	313	313	47	40	44	39	24
min Com.	2	2	2	5	2	2	2	2	2	2	2	2

cross-domain knowledge graph FB15k-237 and FB15k. However, having in mind the lack of information they use, their performance with respect to the metric Hits@k is remarkable.

Comparing LDL and CRP directly, it is noticeable that for Hits@3 and Hits@10 the performance of LDL is generally better than CRP. LDL can generally predict missing relations of head entities with respect to the metrics Hits@3 and Hits@10 very precisely. Especially in the non-biased knowledge graph FB15k-237, LDL performs much better than CRP and has a significant advantage in cross-domain knowledge by computing the distribution function. Considering the performance of both methods with respect to the metric Hits@1, it is remarkable that CRP often performs better than LDL among the different knowledge graphs. Thus, the predictions are much more precise than those of LDL when using only one relation. While LDL performs better at Hits@3 and Hits@10 in most knowledge graphs, CRP manages to outperform LDL when considering the metric Hits@1. The reason for the different efficiency at Hits@k depends on two factors. First, the efficiency depends on the determined communities of CRP, and second, on the impact of the Open World Assumption. We will first look individually at the influences of those factors regarding the effectiveness of the methods separately and then combine the gained insights.

In the following we will first focus on CRP, which predicts missing relations on the basis of communities. In order to analyze the impact of the communities on the performance, Table 4 shows aggregated information about the determined communities of the individual knowledge graphs.

Table 5: Examples of some communities of the FB15k and the relations they contain. The exemplary communities illustrate the latent associations among the KG relations.

Community	Relations
Com 18	nominated_for, honored_for, award_nominee
Com 23	writing_system, language_family
Com 25	symptom_of, diseases, causes, risk_factors
Com 31	writer, producer, artists, distributor, location
Com 39	current_club, players, gender, position
Com 55	institution, major_field_of_study, specialization
Com 79	noble_person, noble_rank
Com 108	rank, military_person
Com 109	producer, program

We first consider the cross-domain knowledge graph FB15k. CRP performs better than LDL but cannot keep up with the performance of the baseline methods. A total of 110 communities were determined on this knowledge graph and the determined communities are certainly reasonable. In Table 5 an extract of the determined communities is shown to get a better impression of them.

3.4.3 *Impact of the Community Structure*

As can be seen, the grouping is certainly reasonable, meaning that similar or related relations have been grouped into the same community. The coverage of relations is very high, 86.47%, as shown in Table 4. However, CRP does not perform as well as the baseline methods. One reason for this is the size of the determined communities, which at an average of 10.57 relations per community, is too high to allow an efficient prediction of missing relations when considering Hits@k. The number of possible relations per community is too high to make an efficient prediction. Considering the metrics in Table 4 again, we also see that the standard deviation of the communities is very high at 15.49, resulting in a large variance in the size of the communities. Thus, there are some communities that contain many relations in the present knowledge graph, and some communities with only very few relations. This large discrepancy in the different sizes of the communities is due to the cross-domain knowledge in the knowledge graph, causing this high standard deviation. This effect is due to the fact that some topics in the knowledge graph are covered with more relations than other topics. As the communities represent the latent subject areas in the knowledge graph, a high standard deviation occurs. Following the CRP approach, once a community has been selected for predicting missing relations, the high average size of the communities results in a large number of possible relations, leading to a poor performance with respect to Hits@k. The communities should therefore contain fewer relations in order to make a more precise prediction and thereby improve

the performance of CRP with respect to Hits@k. Considering another knowledge graph with a high average size of communities like Pers(WD), the hypothesis is confirmed that a large average community size has a negative impact on the performance of CRP with respect to Hits@k. Pers(WD) has an average number of relations per community of 16.58, and has a much higher standard deviation than FB15k, namely 44.83, making the performance of CRP with Hits@1 of 0.253 not satisfactory either. Even with increasing k there is only an insignificant increase in performance with respect to Hits@k, meaning that CRP cannot keep up with the other methods. Considering instead a knowledge graph on which CRP has determined communities with only a small average number of relations, we see that the performance with respect to Hits@k is improved to the previuos considered knowledge graphs. Regarding the knowledge graph Mov(WD), CRP has computed an average number of relations per community of 4.48 and with a standard deviation of 8.60. Considering the performance with respect to Hits@k, CRP was always better than the baseline methods, only being outperformed in Hits@3 and Hits@10 by LDL. Likewise with Mov(DBp), the positive impact of a small average size of the communities and standard deviation on the performance of Hits@k can be seen. Here, however, CRP can only outperform the baseline methods with increasing k and perform better with respect to Hits@k. Likewise when considering the knowledge graph Songs(WD) the average number of relations per community and the standard deviation of the communities is low, however, the performance is not better than expected and not better than the baseline methods. Nevertheless, we see in Table 4 that the coverage of relations is very low at 44.59% and therefore relations in the test dataset are not present in the prediction. The reason for this low coverage is due to the pre-processing of the data in the Highly-Correlated Relationsets step (cf. Definition 12) in which itemsets are calculated. A large number of itemsets were determined, resulting in excessive memory consumption. Therefore we increased the minbond value, although this leads to many missing relations in the test set. However, if sufficient resources are available, it is recommended to reduce the minbond value, so that there is a higher coverage of relations in this knowledge graph. The low coverage of relations with only 44.57% has a negative impact on the performance of predicting missing relations, thus TransE could not be outperformed. Considering the low coverage of relations, the performance with respect to Hits@k is still satisfactory and better than ComplEx. We can state that the processing steps, such as the computation of itemsets and communities, have a strong impact on the performance of CRP with respect to Hits@k. A small average size of communities and a low standard deviation has a positive impact on the efficiency of learning the model. Whereas a large average size of the communities and a high standard deviation has a negative impact on the efficiency of CRP, as does a low coverage of the relations. Thus, CRP generally has a worse initial situation on cross-domain knowledge graphs, as different topics are usually represented with a varying number of relations. This results in CRP communities with a high standard deviation regarding the number of relations per community, which leads to a worse performance with respect to Hits@k. Therefore, much more effort must be put into the computation of the communities to keep the number of relations and standard deviation of the communities low. We will discuss future work to

improve the method in more detail in the section 3.5. One of the possible tasks is to put more effort into the computation of the communities and to keep them small, even when applying CRP on cross-domain knowledge graphs.

3.4.4 *Impact of the Knowledge Graph Topology*

Comparing the results of LDL among the different knowledge graphs, a large discrepancy in the achieved performance can be seen. LDL achieved only 0.772 on the knowledge graph FB15k with respect to metric Hits@10 and thus has a lower performance than the baseline methods, but achieved a Hits@10 of 0.879 on the knowledge graph Mov(WD) and thus a much better performance than the baseline methods. In the following we want to discuss the difference in performance across the knowledge graphs and explain the reasons for the different efficiency of LDL on the knowledge graphs.

There are two factors affecting the performance of LDL, which is on the one hand the density of the usage of the relations for which LDL determines a target distribution and on the other hand the impact of the Open World Assumption on the predictions. We will have a closer look at the latter of the two in the next section and focus now on the density of the usage of the relations. As explained in Section 3.3, LDL generates a head-relation graph encoding the use of relations for each head entity. The density of this graph reflects the actual usage of relations compared to the potential usage of relations. The higher the density, the more relations the head entities use. The density of the Head-Relation Graph is an indicator of LDL's performance, as when many interactions of relations exist, the more LDL can use them to compute a target distribution. Thereby no complete graph, i.e. a Head-Relation Graph where every head entity uses every relation, but only a critical number of interactions between a head entity and relations has to be reached, thus making it sufficient to learn a target distribution. Therefore the density for each knowledge graph is shown in Table 6. When considering the knowledge graph Song(WD) in Table 6, the knowledge graph has a density of $0.222e^{-3}$, which is much lower than the other knowledge graphs. Having this low density in mind and considering the achieved result with respect to Hits@10 on this knowledge graph, we see that the performance with 0.819 is lower than the performance on other knowledge graphs. The performance of LDL on the knowledge graph Songs(WD) is particularly worse than the performance of CRP, which is surprising, as LDL is usually better than CRP with increasing k in Hits@k. Nevertheless, LDL is not able to determine a meaningful target distribution on the knowledge graph to achieve a high performance with respect to Hits@k. The reason is the low density of the head-relation graph, indicating that for some head entities only very few relations are used, thus not enough information is available to determine a meaningful target distribution. A similar result is obtained on the knowledge graph Mov(DBp), which also has a low density. Although LDL is able to outperform the baseline methods, it is not able to determine a meaningful target distribution due to the low density, leading to a Hits@10 of 0.847 and thus a worse performance than CRP. In contrast, the density of the knowledge graph Comp(DBp) in Table 6 is with a density of $2.641e-3$ very high compared to the other knowledge graphs. Its performance

Table 6: Density of the Head-Relation Graphs for the knowledge graphs considered.

Metric	FB15k	FB15k-237	WN18	WN18RR	Pers(DBp)	Pers(WD)	Comp(DBp)	Comp(WD)	Mov(DBp)	Mov(WD)	Songs(DBp)	Songs(WD)
Density	1.112e-3	0.946e-3	0.104e-3	0.079e-3	0.146e-3	4.884e-3	2.641e-3	0.634e-3	0.163e-3	9.493e-3	4.986e-3	0.222e-3

is, compared to the other methods with a Hits@10 of 0.832, much better than the other methods. However, when considering this knowledge graph and the different Hits@k metrics, it is apparent that LDL does not perform as well as the baseline models with respect to the Hits@1 and Hits@3 metrics, despite the higher density. Similarly, the knowledge graph Songs(DBp), which has a density of $4,986e-3$, is much higher than the other knowledge graphs. The achieved result of LDL with a Hits@10 of 0.924 on Songs(DBp) is as well very good and confirms the hypothesis that a high density generally leads to a better result, but LDL is outperformed on this knowledge graph by TransE. Nevertheless, the performance is remarkable, considering that only head-relation information was used to predict missing relations. However, even on this knowledge graph LDL does not perform as well as the baseline models with respect to the metrics Hits@1 and Hits@3. For completeness we want to mention the knowledge graph Mov(WD) with a density of $9.493e-3$, which has a high density as well, but LDL does not perform as well as on other knowledge graphs with respect to the metrics Hits@1 and Hits@3. The performance of LDL on Mov(WD) is better than the other methods with respect to Hits@10, but with 0.879 less than expected, although there is a high density for this knowledge graph. We can thus state that the density of the head-relation graph has an impact on the effectiveness of learning a target-distribution of relations and thus on the performance of LDL with respect to Hits@k, but there are some knowledge graphs where this conclusion is not entirely valid. The performance of LDL with respect to Hits@k is generally worse than the other methods with a small k, and only with increasing k does LDL achieve better performance. The question remains why LDL performs worse than expected with respect to Hits@k with a small k and why, despite a high density of the head-relation graph, the performance of LDL sometimes deviates from the expectation, as in the case of Pers(WD), which only achieved a Hits@10 of 0.663 despite a high density of $4.884e-3$. Thus, there must be another factor affecting the effectiveness of LDL. Based on the conducted experiments it turned out that this influencing factor does not only affect LDL, but also CRP. This influencing factor is due to the Open World Assumption and the resulting limitation of the evaluation by means of a silver standard. This will be discussed in more detail in the following.

3.4.5 *Impact of the Open World Assumption*

A significant impact on the efficiency of the two introduced methods is the Open World Assumption and the resulting silver standard, which may be incomplete. Thus, relations may be predicted to be missing, but due to the incomplete facts of the silver standard, these may be evaluated to be wrong, although they may still be correct. We refer to those cases in which a predicted relation for a given head entity is mistakenly considered to be false as spurious false positives. We

Table 7: Samples of spurious false positives across all considered knowledge graphs.

Dataset	Head Entity	Actual Relation	LDL	CRP
FB15k-237	The Matrix Reloaded	nominated for	film festivals	award winner
FB15k	Marshall	time zones	county	time zones
WN18RR	Fugaciousness	synset domain topic of	member meronym	derivationally related form
WN18	Fugaciousness	synset domain topic of	member meronym	derivationally related form
Pers(DBp)	Deven Marrero	bats	throws	throws
Pers(WD)	Jiajing Emperor of Ming	cell line	military rank	given name
Comp(DBp)	Zubaan Books	distribution	headquarters	foundation
Comp(WD)	Google China	headquarters location	named after	headquarters location
Mov(DBp)	The Naked Gun	language	basedOn	basedOn
Mov(WD)	Harry Potter and the Chamber of Secrets	executive producer	screenwriter	screenwriter
Songs(DBp)	All Summer Long (The Beach Boys song)	recorded	released	released
Songs(WD)	Calling You	language of work or name	lyrics by	follows

can manually identify these cases and include them in our metrics accordingly, however, this is an extremely time-consuming process that would not be appropriate given the large number of knowledge graphs. We will therefore present a few examples below to indicate that the actual performance of CRP and LDL is much better than previously presented, but cannot be presented due to the Open World Assumption and the lack of a suitable evaluation option. This aspect of the incompleteness of the KGs must be considered while evaluating the results. Consider the movie *The_Naked_Gun* in the Mov(DBp) dataset. We predicted with CRP among others *basedOn* as missing relation for this head entity. According to our silver standard, this prediction is considered to be a false positive because this head entity does not contain a *basedOn* relation in the KG, therefore, it is not part of the Silver Standard. However, assuming complete knowledge about the film, the prediction of our approach would be correct, because the film is based on the american television comedy *Police Squad!*. Similar cases of spurious false positives are encountered in other KGs, e.g. Pers(DBp). For example, the head entity *Deven_Marrero* describes an american professional baseball player. Both of our approaches predicted *throws* as missing relation. Again, this prediction was considered a false positive, as there is no *throws* relation for the head entity *Deven_Marrero* in the original KG. However, we can confirm that this information is indeed missing, because this baseball player actually throws baseballs with his right hand. Likewise, relations for other entities e.g. *Dave_Dictor* is missing. Our approaches predicted *instrument* as missing relation, while it is known that this entity is an american musician which plays the guitar.

Cases of spurious false positives occur across all knowledge graphs, not only on the above mentioned ones, but also, for example, on the cross-domain knowledge graphs FB15k and FB15k-237. In the case of the knowledge graph FB15k-237, the relation *film_festivals* was predicted for the head entity *The_Matrix_Reloaded* by LDL as missing relation. However, the entity does not use the relation *film_festivals* in the test data set, although the film was premiered at the Cannes International Film Festival and, therefore, this relation would have been reasonable. Table 7 shows samples for each knowledge graph of spurious false positives for both methods. The above examples illustrate the problems involved in evaluating KG

completeness. Although in some cases the predictions are correct, the evaluation classifies them as wrong because the information is not available in the KG. Due to the incompleteness of the KGs, the actual effectiveness of the predictions cannot be assessed with absolute certainty. Assuming that all information is available to us, the performance will increase, because more information tends to improve the results. Consider Hits@10 for a head entity. More information in the test dataset cannot lead to changing the predictions made for the head entity. A correct prediction will be a correct prediction since $KG \subseteq KG^*$ (with KG^*, the complete version of KG). However, a previously false prediction may turn into a correct prediction, as more information may cause this information not to be available in KG.

The above examples indicate that Open World Assumption has a huge impact on the measured effectiveness of both LDL and CRP methods. We have shown that the predictions of both methods are useful in many cases, but due to the limitations of the silver standard it is difficult to measure the actual effectiveness. A more detailed analysis of the effectiveness of the methods, which reduces the limitations of the Open World Assumption, would go beyond the scope of this work, hence we decided against it. Nevertheless, despite the Open World Assumption, the effectiveness of our methods is competitive with those of the baseline models. Both methods performed in general well with respect to Hits@k, especially with increasing k. LDL generally performed better than CRP, especially with increasing k. One reason for this is the fact that with increasing k the impact of the Open World Assumption decreases, since more relations are proposed as missing relations. As k increases, the effect of the spurious false positive decreases, causing the first relation proposed, which should be obviously correct, but is classified as false, to be offset by the other proposed relations. In other words, with increasing k the metric Hits@k increases, while the impact of the Open World Assumption decreases and the effectiveness according to Hits@k of our methods increases.

3.5 SUMMARY AND FUTURE WORK

In this chapter we have addressed the problem of predicting missing relations in knowledge graphs which are represented under the Open World Assumption. For this purpose, two different methods for predicting relations in knowledge graphs were introduced, both based on machine learning methods. First, CRP was introduced, which first identifies correlating relations and represents them as a graph. Then, based on this graph, a clustering method is applied to the graph to determine latent groups of relations, which are then used for the prediction of missing relations. We showed in the experiments that the clusters contain coherent relations, even in cross-domain knowledge graphs like FB15k. In the experiments, this method achieved satisfying results despite the assumption of an Open World Assumption, but has the disadvantage, due to the itemset mining, of having a much worse runtime than the baseline methods. In addition, the experiments showed that the determined communities have a great impact on the performance of this method. If the communities consist of too many relations, their latent representation becomes blurred, since too many relations with

different semantics are assigned to one community, and the efficiency of CRP is reduced. The number of relations per community should therefore be low to ensure high performance. Later we will discuss this in more detail and give an outlook on how this issue can be addressed in the future.

Although the results of CRP could keep up with the baseline methods in many knowledge graphs despite the Open World Assumption, yet not on cross-domain knowledge graphs, we wanted to improve the performance even further. Due to the latent communities we have identified that the usage of relations for the head entities follow a certain distribution. Based on this finding, we introduced an unsupervised stochastic neural network, which computes a target distribution based on the usage of relations in the knowledge graph. Using this target distribution and the previously used relations for a given head entity, the probability of using further relations for head entities can be computed. The basis for this method, called LDL, was a Restricted Boltzmann Machine. We have adapted the model to handle these with the characteristics of a knowledge graph. We evaluated LDL on the same twelve knowledge graphs as CRP and its performance was more than satisfying, in many cases even outperforming baseline methods that use more information for predicting missing relations. When analyzing the results in detail, we noticed that the density of the head-relation graph, i.e. the graph indicating which relations are used by which head entities, has a great impact on the performance. As a consequence, when head entities are described by very few relations, LDL cannot learn a meaningful target distribution. However, we also found that despite a high density, LDL performs poorly on some knowledge graphs, and in general, when using the metric Hits@k, is generally better than CRP only with higher k. It has been shown that due to the limitation of the evaluation by means of a silver standard, due to the Open World Assumption, CRP, as well as LDL, predicted relations, which are quite useful and could actually be considered to be correct, but the information was not available in the test dataset and was therefore classified as incorrect. As a result, the performance of Hits@k is generally lower, which is especially noticeable with small k, since with higher k and thus more proposed missing relations, this disadvantage balances itself out to a certain extent. Nevertheless, these spurious false positives have a huge impact on the measured performance of CRP and LDL, making it difficult to accurately determine the actual efficiency of both methods. We will later discuss in more detail how we can address this problem in the future.

The hypothesis made at the beginning and the research questions derived from it are answered on the basis of the methods introduced in this chapter and the insights from the experiments in the following.

◉ Answering Research Questions

Hypothesis: Link distribution learning is suitable for predicting missing properties in knowledge graphs which are represented under the Open World Assumption.

1.1 What is the effectiveness on learning KG features for link prediction of stochastic factorization models, in contrast to rule mining methods.

Answ. Stochastic factorization models are generally more effective in predicting missing relations than rule mining methods.

1.2 What are the characteristics of knowledge graphs that allow learning models to effectively learn the distribution of links?

Answ. Important characteristics of the stochastic factorisation model are the sufficient use of relations to generate a dense head-relation graph. Similarly, in the case of rule mining methods, relations should be used frequently and grouped in smaller communities.

1.3 What is the impact of following the Open World Assumption on the effectiveness of the studied methods?

Answ. The performance of the methods is generally reported to be lower than it actually is. Due to the OWA, a comprehensive study of the effectiveness cannot be guaranteed.

Based on the insights gained from the experiments, some of the topics of this work can be studied more closely in the future. On the one hand, we have shown the impact of the number of relations in the determined communities on the performance of CRP, which demonstrates the possibility to go into more detail and improve CRP with respect to the computation of communities. The currently used method to determine the communities, Fastgreedy, determines the number of communities itself, which has the advantage that no domain knowledge is needed for determining the communities. The disadvantage, however, is that communities with different sizes are then determined. Using a community algorithm with a fixed number of communities could counteract this effect, allowing a larger number of communities to be specified, which leads to fewer relations per community. As the experiments have shown, this would have a positive impact on the performance of CRP with respect to Hits@k. A disadvantage resulting from this is a further hyperparameter, namely the specification of the number of communities, which has to be tuned depending on the knowledge graph. Thus, domain knowledge is required or a validation dataset on which the number of communities for the knowledge graph can be tuned. This would add further complexity and effort to the application of CRP. Resolving this tension must be solved individually. Besides the use of a suitable community clustering algorithm, the question of using another itemset mining algorithm arises. We chose FCHM because of its good runtime and bonding value. In general, the itemset mining has only a minor impact on the performance, since it only filters out noisy relations. However, having a complexity of $O(2^n)$ the runtime of FCHM is

very complex. Within FCHM, combinations of relations which do not correlate are pruned at an early stage, but the worst-case complexity of $O(2^n)$ remains. In general, FCHM is already a powerful method, but further research could be done to develop an itemset mining method which has a better performance regarding the determination of itemsets. The experiments also showed that the comparison with the baseline methods, which follow the Closed World Assumption, is only possible to a very limited extent due to the silver standard used. There were cases of spurious false predictions, i.e. supposedly wrong predicted relations which, on closer examination, turned out to be correct predictions, but were classified as false due to the incompleteness of the silver standard. These cases occurred during the evaluation of both methods presented. We showed that the performance of the metric Hits@k was reduced due to this and that a meaningful comparison between our methods and the baseline methods is thus not given. This problem should be addressed in the future to evaluate methods based on an Open World Assumption in order to determine their actual performance more precisely. Using full text from the web and NLP methods, the predicted relations of the methods can be validated to check if they match the extracted information from full text from the web. Again, this does not represent complete information, but is an extension of the knowledge already available, allowing the performance of methods following an Open World Assumption to be examined more precisely. A challenge is to use appropriate NLP methods that validate a fact with high certainty. This method should scale, considering the amount of information available on the web, allowing a fact to be validated quickly based on the large amount of information available.

Considering the excellent performance of both methods, especially of LDL, regarding the prediction of missing relations in a knowledge graph, it would be interesting to investigate these methods in more detail to what extent they are suitable for the prediction of missing relations on other graph structured data. In this way these methods could be applied generally to graphs or even hypergraphs to predict missing links. In the next chapter, we will partially address this by modifying the LDL method presented in this chapter to such an extent that it can be used for the prediction of missing interactions in bipartite networks. We chose LDL for its outstanding performance and better runtime complexity. As will be shown in the experiments in the next chapter, LDL can be applied to complex bipartite networks and outperform the existing and very strong baseline methods in the prediction of missing interactions.

4

LINK PREDICTION ON BIPARTITE NETWORKS

4.1 INTRODUCTION

Bipartite networks are ubiquitous data structures, used to model relationships between two types of entities. This data structure has become an important foundation to represent interactions between users and items in recommender systems [57], users connected to subjects in social networks [21], and interactions between drugs and targets in bioinformatics [124]. One major difference of bipartite networks, compared to knowledge graphs, is the different structure and characteristics. For a common understanding of bipartite networks, we define them in the following.

Definition 16: Bipartite Network

A bipartite network $BN = ((V_1, V_2), E)$ is a Network $G = (V, E)$, whereby $V_1 \cap V_2 = \emptyset$, and for every edge $(v_1, v_2) \in E$: $v_1 \in V_1$ and $v_2 \in V_2$. This results in the following properties:

1. The two different sets of nodes, V_1 and V_2 have no common elements, are therefore disjoint ($V_1 \cap V_2 = \emptyset$), and when combined, result in all nodes.
2. Edges only connect nodes of different sets. Within the node sets, V_1 and V_2, no edges connect the nodes.

The above definition already shows the major differences between knowledge graphs and bipartite networks. A major difference to knowledge graphs is the absence of labelled edges in the bipartite networks we are considering. Thus, all edges have the same meaning. Another difference is that only edges between the two sets of nodes are possible. Both results in a lower complexity of bipartite networks compared to knowledge graphs. The lower complexity makes the problem of predicting missing links seem simpler, but in fact the prediction can be more difficult as the missing semantics in the relationships between nodes in bipartite networks are not represented. This may result in our model not having enough information to learn a function that allows precise predictions of missing links. As a result, the risk of a possible underfitting is increased. We want to extend the problem a little bit further and focus this work on a special type of bipartite networks, namely complex bipartite networks. Bipartite networks used to model real-world applications are typically called complex.

The analysis of complex bipartite networks includes node clustering [13], node classification [7] and link prediction [57]. The link prediction task is in bioinformatics of special interest toward the identification and development of new uses of existing or discontinued drugs since drug development is currently time consuming and expensive.[12]

In recent years, network embeddings are often trained for encoding the nodes of a network into a low-dimensional space whilst preserving the graph structure. Based on the trained embeddings, machine learning techniques are applied to perform network analysis, such as link prediction. These methods have demonstrated good performance but vary significantly based on the settings of their hyperparameters and dataset on which they are applied [58]. For instance, the length of the random walks of DeepWalk [95] or of node2vec [42] has a significant impact on the learned embeddings and the subsequent machine learning tasks. In addition, most of the methods for link prediction are evaluated on connected networks, e.g., AttentionWalk [3] and node2vec [42]. The performance of these methods, however, also varies significantly depending on the topology of the networks. This may lead to unsatisfactory results when applying these methods to complex bipartite networks, since these networks might consist of many components. We want to address these existing limitations and create a method that predicts missing links in complex bipartite networks, even if the network is disconnected. Instead of relying on random walk methods, as existing work does, we want to develop a method based on stochastic factorization. The goal of our stochastic factorization model is to compute a target distribution over the interactions of complex bipartite networks and use this distribution to predict missing links. Our hypothesis is that a stochastic factorization model can learn the distribution of interactions of nodes in a bipartite network, which can be used to predict missing links in the bipartite networks, achieving a high performance regarding predicting missing interaction. Therefore, we want to analyse the effectiveness of a factorization model for the prediction of missing links in complex bipartite networks, even if the complex bipartite networks are disconnected. In particular, we want to compare the performance to existing work and highlight differences with respect to various metrics and requirements. Thereby we want to develop a deeper understanding of the results and their causes in order to expand our model in the future and adapt it to different requirements. Furthermore, we want to investigate the impact of the characteristics of bipartite networks on the performance of the stochastic factorization model. We are interested in the network properties, such as graph density and the number of components, which might have an impact on the performance of our model. Based on this information, we are able to estimate the performance of our model, even before it is applied. This allows for determining the area of application and characteristics of bipartite networks in which the stochastic factorization model achieves good performance. Finally, we want to analyse the impact of different hyperparameters, especially the impact of the distribution function in the hidden layer, on the performance of the stochastic factorization model. We expect to derive a recommendation for the choice of hyperparameters based on the find-

[12] It typically takes more than 10 years and more than 1 billion dollars [28]

ings of this analysis. The hypothesis we make, including the research questions based on it, is formulated in Research Question 2.

Research Question 2

Hypothesis: A stochastic factorization model is able to learn the distribution of links, even in disconnected bipartite networks.

2.1 How effective are stochastic factorization models in link prediction, even in disconnected bipartite networks?

2.2 What are the characteristics of bipartite networks that allow stochastic factorization models to effectively learn the distribution of links?

2.3 What is the impact of the hyperparameters, especially the distribution function in the stochastic unit, on the evaluation measurements?

4.1.1 *Structure of the Chapter*

The structure of this chapter is similar to the previous chapter. In the following Section 4.1.2, we will introduce the problem in more detail using a motivating example. The motivating example is in the following sections used to explain the procedure of our method based on a practical example, making it easier to comprehend the procedure. In the following section, Section 4.2, related work in the context of Link Prediction, and thus our work, will be presented. We will discuss the existing work and distinguish the existing methods from our own. The focus is not exclusively on methods for predicting missing links in bipartite networks. We will broaden the focus to gain insights from existing work on related problems in order to take them into account in our approach, if applicable. In Section 4.3 we present our method for predicting missing links in complex bipartite networks. The motivating example from Section 4.1.2 is used to illustrate the method and make it more comprehensible. In several subsections, details about the individual aspects of our method is given, such as the input to our model, how the parameters are learned, and how the predictions are made. In the following section we will conduct experimental studies. These studies are designed to support us in answering the research questions. We will discuss the effectiveness of our method and compare it with other strong baseline methods. Furthermore, we will explain which characteristics complex bipartite networks need to have in order to guarantee a high effectiveness of our method. Based on these experiments we can then answer the research questions, posed in the introduction, in Section 4.5. We thereby will summarize the presented work and, based on the findings from the experiments, give an outlook to future work.

4.1.2 *Motivating Example*

As already introduced, complex bipartite networks model interactions of real-world applications, such as in social networks or in bioinformatics to model the relationships between drugs and targets, i.e. the drugs and the diseases for

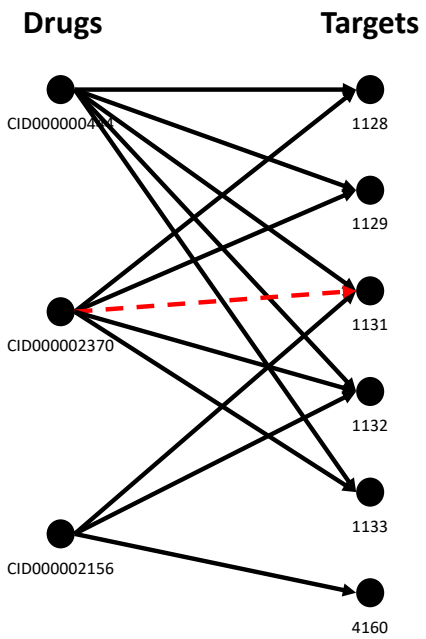


Figure 15: Running example using a subgraph of the complex bipartite network ChG-Miner. We are interested in predicting missing interactions, such as between CID000002370 and 1131.

which they are used. In our running example, we would like to focus on the latter, while clearly emphasizing that our method is applicable to all bipartite networks, and not exclusively to complex bipartite networks in bioinformatics. As already discussed in Definition 16, the nodes of a bipartite network can be divided into two disjoint subsets and there can only be interactions between the nodes of these two sets, but not between nodes of the same set. Looking at the issue of drug-target interactions, we find that this exactly reflects the definition of a bipartite network. There are exactly two sets of nodes, the drugs and the targets. Each node of the network can therefore be assigned to exactly one set, the set of drugs or the set of targets. Likewise, there are no interactions between the nodes in a node set, meaning that a drug does not interact with another drug, but only with a target. Based on this scenario and the Definition 16 we can model the drug-target interactions as a bipartite network. We have modeled this as an example of a subgraph of the ChG-Miner dataset from Stanford SNAP in Figure 15.

Based on the bipartite network shown in Figure 15, we want to predict missing interactions in the network. A possible missing interaction is shown as a red arrow in Figure 15. We want to predict if there should be an interaction between Drug CID000002370 and Target 1131. This would generally allow us to identify drugs that can be applied to new targets and, thus, increase the pace of identifying new treatments. We formulate the motivation in the following.

Motivation : Link Prediction on Bipartite Networks

Based on a bipartite network $BN = ((V_1, V_2), E)$, a function $f(n_i, n_j)$ should be learned, which indicates the likelihood of an interaction between a node pair $n_1 \in V_1$ and $n_j \in V_2$. This function is intended to predict missing interactions in the bipartite network BN.

Based on Figure 15 node2vec could be applied, which allows for transforming the nodes into a low-dimensional space whilst preserving the distribution of interactions. Based on these embeddings, a machine learning model such as logistic regression can be trained to predict whether an interaction between two nodes exists or not. However, node2vec is a transductive methods, meaning that if a new node is added to the network, the model must be retrained to embed and learn from the new node. This is very time-consuming when using node2vec which we would like to avoid. In addition, we would prefer not to train a downstream machine learning model such as Logistic Regression. Our goal is to build an end-to-end pipeline that predicts missing interactions without extensive preprocessing and downstream processing. Hence, our goal is to implement a function $f(n_i, n_j)$ which computes the probability of an interaction between nodes n_i and n_j in a bipartite network. This function may be learned on the basis of existing interactions of the bipartite network using machine learning methods. With the help of such a function on a bipartite network, as shown in Figure 15, previously unknown interactions between drugs and targets can be predicted. This is of particular interest for bioinformatics in order to apply existing drugs to a wider range of diseases beyond the intended use of the drugs for their treatment. This reduces the time-to-market for the treatment of diseases, as described above, as existing drugs can be reused instead of developing new drugs which have to undergo very long tests. In addition, the costs for the market launch are reduced as research and development costs are omitted.

4.2 RELATED WORK

Our work focuses on link prediction in complex bipartite networks. In contrast to existing work, we learn a target distribution over the bipartite network interactions allowing to predict unknown links. BiNe [39] is a method which is also focused on bipartite networks. BiNE learns low-dimensional node representations for bipartite networks, based on explicit and implicit relations. Explicit relations are modeled by using the KL-divergence to minimize the co-occurring probability between nodes and the reconstructed distribution. Implicit relations are modeled by using a Skip-gram model [74]. Both relations models are combined by a joint optimization framework. Interaction Graph Embedding (IGE) [132] allows for encoding nodes of bipartite attributed interaction graphs. Thus, temporal events in interaction graphs and heterogeneous attributes on edges are encoded in the embeddings. There is also the possibility to leverage the hierarchical structure of bipartite networks for recommending items based on communities in the network [56]. Previous works in the biomedical domain on the prediction of interactions in bipartite networks have focused on similarity metrics based

on similar relationships (Weighted profile method) or similar chemical compositions (Nearest profile method) [124]. In other works, these additional attributes were used to learn features using auto-encoders [119]. We do not consider the latter method since we assume that additional attributes, i.e. chemical composition of nodes, is not available to us.

We use a Restricted Boltzmann Machine (RBM) to learn a target distribution over the bipartite network interactions allowing to predict unknown links. RBMs have been applied in the past especially for dimensionality reduction [46], learning and reconstructing sparse representations of the input [100] and collaborative filtering [15]. Although first attempts were made to apply (Conditional) RBMs to predict different types of interactions in bipartite networks [121], they differ significantly in the encoding of the bipartite network as input, as well as in the encoding of the information in the hidden states. In a previous work [121], the input and hidden states were represented as multidimensional matrices. The results look promising, even though the method has only been evaluated on very small bipartite networks. We will evaluate our approach on a larger number of complex bipartite networks, considering only bipartite networks with the same type of interactions. An extension of RBMs, the so-called Deep Belief Networks (DBNs), which are stacked RBMs, were used in social networks to predict missing links [60]. However, due to the increased complexity of the DBNs, more data must be available to train the network sufficiently. Bipartite networks usually have a lower number of interactions between nodes. Therefore, DBNs probably cannot be trained sufficiently to achieve good performance.

Besides the above mentioned related work for link prediction with special focus on bipartite networks, there are many methods for link predictions on networks in general, also applicable on bipartite networks. Many of these method focus on learning a low-dimensional node representation. Node2Vec [42] and DeepWalk [95] are very prominent methods that use random walks as sampling strategy. The learning of the embeddings is done using the Skip-gram model [74]. However, these methods are, due to the random walk approach not robust in terms of the connectivity of the input network. Methods that rely on a random walk approach usually perform poor when having disconnected networks as input. Due to the fact that our method is not based on a random walk strategy, we hope to achieve better results, particularly in the case of disconnected networks. In addition, transductive methods like e.g. node2vec [42] and DeepWalk [95], meaning that if a new node is added to the network, the model must be retrained to embed and learn from the new node, GraphSage [44] is an inductive framework that leverages features from a nodeâs local neighborhood to efficiently generate representations on previously unseen data. We use an inductive method as well, so that our model once learned can also be used for considering new nodes. In order to overcome the manually tuning of hyperparameters (e.g. length of random walks or number of walks) for every network, AttentionWalk [3] allows for automatically learn the hyperparameters via backpropagation. This is especially useful with a large number of hyperparameters, such as in node2vec. Compared to existing link prediction methods, our method uses only a small number of hyperparameters, so that automatic learning seems unnecessary in our opinion. Community structure based methods leverages the

organizational structures and functional components of networks to learn embeddings [63, 120]. NetMF [98] is a factorization model that explicitly factorize the closed-form matrices, which is implicitly approximated by DeepWalk [95] and LINE [111]. Besides these topological methods, there are probabilistic methods that estimate the joint co-occurrence probability of two nodes. Hierarchical Bayesian model [118] captures high-dimensional node attributes and link structures with layers of latent variables. Variational Graph Auto-Encoders (VAEs) [52] are end-to-end trainable neural network models for unsupervised learning on graph-structured data. The model achieves competitive results on link prediction task, but cannot keep up with state-of-the-art methods. Compared to our model, VAE assume a Gaussian distribution. Generative Adversarial Networks (GANs) allow for learning node distributions and capture, based on the relation-aware design, the rich semantics of the relations. [47]. In order to explicitly model edges in a network, a representation of edges can be learned via a low-Rank projection [2]. Diffusion-based representations [8] allows for a latent representation of the graph-structured data which can be used for link prediction. In addition, such an approach is very runtime efficient. For more details on network representation learning and link prediction, please refer to survey papers [40, 129].

4.3 LEARNING PROBABILITY LINK DISTRIBUTION FOR LINK PREDICTION

Due to the success of LDL in predicting missing relations in knowledge graphs (see Chapter 3), we use this model as the basis for predicting missing links in complex bipartite networks, even if they are disconnected. However, LDL cannot be applied directly to bipartite networks, but must be adapted for application to complex bipartite networks. Since there are differences between bipartite networks and knowledge graphs, such as no labelled edges between nodes, and characteristic features of bipartite networks, such as the fact that connections are only possible between nodes of different types, we have to take into account for an appropriate application of LDL to complex bipartite networks. In the following sections we will discuss and explain the individual components of the model. This includes the input of the model, the model itself, the learning of the parameters, and how our model preforms the predictions using the learnt parameters. The motivating example from section 4.1.2 will be used as a running example to explain the different components with a practical example. We will follow the running example from creating the input for the model, through the learning of the parameters, to the prediction of missing edges.

4.3.1 *Input*

The input of the model is based on a numerical representation. Since a distribution of the interactions of the nodes in the bipartite network is to be determined in order to make predictions with their help, it must additionally contain the interactions of each individual node. Bipartite networks have the special characteristic of allowing the set of nodes to be divided into two disjoint subsets, so that only connections between nodes of these two sets are valid, but no connections between nodes within a subset. This can be exploited when defining the

input to the model for reducing the number of nodes considered, and thus the complexity. However, if statements are made using this model about all nodes, then the bipartite network structure cannot be exploited to reduce complexity. In the following both possible definitions for the input of the model are presented and advantages and disadvantages are discussed. Depending on the application scenario it is recommended to choose the appropriate input. Regardless of which of the two model input definitions presented below is chosen, the method presented can handle it. The fundamental difference lies in the number of nodes to be considered and thus the number of dimensions for the model as input features. In the following we define the input to our model, which exploits the characteristic feature of bipartite networks, namely that there are only edges between two different types of nodes, for reducing the complexity of the input.

Definition 17: Model Input

The model input is a bipartite network $BN = ((V_1, V_2), E)$, where V_1 and V_2 corresponds to nodes and E to links. Based on BN, our approach builds a vector $\mathbf{v}$ for a given node $n_i \in V_1$, such that $\mathbf{v}_{[j]}$ is 1 if $(n_i, n_j) \in E$ with $n_j \in V_2$, and 0 otherwise.

In the above Definition 17, depending on the application scenario, it has to be decided whether to consider the interactions starting from V_1 or V_2 and use them to compute the distribution function. For example, to determine missing edges in V_1 starting from node $n_i \in V_1$, it is recommended to define the input according to this node set. In this case, however, only the target distribution for this node set is determined and, based on this, only statements regarding this node set can be made. Focusing the analysis on a specific node set results in the number of dimensions of the vector $\mathbf{v}$ corresponding to the number of nodes of the considered node set. In our running example, we can define the drugs as node set V_1 and the targets as node set V_2. Assuming we only want to make statements about the interactions of the Drug nodes, i.e., V_1 node set, the vectors $\mathbf{v}$ would have six dimensions, as defined above, since we have six nodes in the target node set (see Figure 15). If we consider the target, V_2, the number of dimensions of the vectors $\mathbf{v}$ would be three, since there are three nodes in the drug node set. This reduction in the number of nodes to be considered has the advantage of lower complexity and thus faster computation, since the number of parameters in the Stochastic Neural Network is lower due to the reduced dimensions of the input vectors. However, the reduced complexity also has an impact on the expressivity of the model, as now not all nodes are considered. Due to the reduced complexity, we can only make statements for a limited number of nodes, corresponding to the number of nodes in the set considered. There is therefore a trade-off between a lower complexity of the model and thus a faster computation, and the expressive power of the model. For this reason, in the following we define the input to our model in the case that statements are to be made about all nodes of the bipartite network.

Definition 18: Model Input - Complex

The model input is a bipartite network $BN = ((V_1, V_2), E)$, where V_1 and V_2 corresponds to nodes and E to links. We denote $V = V_1 \cup V_2$. Based on BN, we build a vector $\mathbf{v}$ for a given node $n_i \in V$, such that $\mathbf{v}_{[j]}$ is 1 if $(n_i, n_j) \in E$ with $n_j \in V$, and 0 otherwise.

It is important to note that the Definition 18 is generally applicable to networks, and not only to bipartite networks. However, the focus of this study is on bipartite networks, although networks in general could be considered using this definition. The Definition 18 has the advantage that all nodes are considered and thus statements about all nodes of the network can be made, however, the number of parameters to be learned and thus the complexity, due to the higher number of dimensions of the input, increases. We would like to explain this in the following using our running example. We consider the node set Drug (V_1), Target (V_2) and the total node set ($V = V_1 \cup V_2$) and look at the number of parameters to be learned and thus the complexity. The input to the model is the number of dimensions of the input vector and a bias value. Therefore, the number of parameters, assuming $|\mathbf{h}|$ neurons in the hidden layer is $(|\mathbf{v}| + 1) \cdot |\mathbf{h}|$. If we assume that the final model consists of two neurons in the hidden layer ($|h| = 2$), then the number of parameters to be learned corresponds to $(|\mathbf{v}| + 1) \cdot 2$ This results in the following number of parameters for the running example according to the number of input features.

- **Considering** V_1: $(6 + 1) \cdot 2 = 14$ parameters
- **Considering** V_2: $(3 + 1) \cdot 2 = 8$ parameters
- **Considering** V: $(9 + 1) \cdot 2 = 20$ parameters

The above example demonstrates clearly that a larger number of nodes in the target set increases the number of parameters and thus the complexity of the model. The trade-off between a lower complexity of the model and thus a faster computation, and the expressivity of the model, must be solved individually depending on the application and the problem to be addressed. In the case of making statements only about a certain number of nodes or in the case of a risk of overfitting and hence the complexity of the model should be reduced, we recommend generating the input for the model according to the simple Definition 17. If our model is used to make statements about all nodes, we recommend generating the input for the model according to the more complex Definition 18. In the following, we will use the more complex definition of the input to ensure greater expressiveness of the model.

In the two definitions above, nodes are represented by a binary vector. This binary vector represents whether a link to another node exists or not. This type of representation is a very simple, but also a very common one in the field of networks. Due to the characteristics of the bipartite network, namely that there are no semantic relations, no node can have two edges to the same node. Therefore, the entries of a vector at the positions representing the corresponding

nodes n_j of the other set can have a maximum value of one, since there can be at most one edge from node $n_i \in V_1$ to another node $n_j \in V_2$. We will show later in Chapter 5 how to use the model to learn a representation of the nodes that implicitly encodes the semantic relationship between nodes and thus has a significant advantage over the currently used binary representation.

4.3.2 *Model*

Restricted Boltzmann Machines (RBMs) belong to the energy based models which consist out of one visible layer and one hidden layer. The input for the visible layer is denoted as $\mathbf{v}$ and the result of the hidden layer denoted as $\mathbf{h}$. There is no explicit output layer, since the model tries to reconstruct the input and uses the reconstructed data as output. Energy based probabilistic models use a probability distribution through an energy function to measure the quality, similar to cost functions of machine learning models. Having the hidden layer as latent variables to increase the expressive power of the model, we get the following energy-based probabilistic function (Gibbs distribution) which denotes that a certain state $\mathbf{v}$ can be observed.

$$P(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{13}$$

where Z is the sum from all possible states and called the normalizing factor:

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{14}$$

Having a low energy $E(\mathbf{v},\mathbf{h})$ leads to a high probability, whereas a high energy translates to a low probability. The goal of the method is to increase the probability of the energy-based probabilistic function $P(\mathbf{v})$. Therefore the energy function $E(\mathbf{v},\mathbf{h})$ must be minimal to get a high probability $P(\mathbf{v})$. The energy function $E(\mathbf{v},\mathbf{h})$ for an RBM with its input $\mathbf{v}$ and hidden state $\mathbf{h}$ is the following.

$$E(\mathbf{v},\mathbf{h}) = -\mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} \tag{15}$$

$\mathbf{W}$ represents the weight matrix between visible layer and hidden layer. $\mathbf{a}$ and $\mathbf{b}$ each represent the bias of the respective layer.

The first step in the computation of the hidden states $\mathbf{h}$ is similarly to a Feed-Forward Neural Network. We compute the probability of the hidden state, based on an input $\mathbf{v}$ as following.

$$P(\mathbf{h}|\mathbf{v}) = \sigma(\mathbf{W}\mathbf{v} + \mathbf{b}) \tag{16}$$

σ is the sigmoid activation function denoted by $\sigma(x) = \frac{1}{1+e^{-x}}$. The sigmoid activation function projects the weighted sum of the input into the range $(0, 1)$. The output of the sigmoid function, $P(\mathbf{h}|\mathbf{v})$, can be interpreted as the probability of a hidden state being activated. Considering the running example in Figure 16, the node *CID000002370* is converted to a binary vector according to the Definition 17, then $P(\mathbf{h} \mid \mathbf{v})$ is computed. These values can be interpreted as probabilities, which,

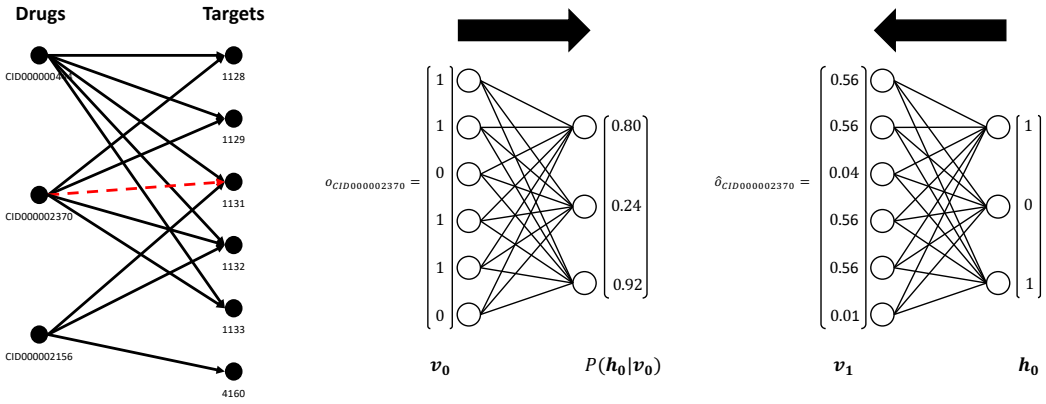


Figure 16: Computation of the output of the model, based on an input and using the Bernoulli distribution in the hidden layer.

based on the distribution function applied below, indicate the probability of a neuron being activated. According to the running example, there is an 80% probability that the first neuron in the hidden layer will be activated. Accordingly, the second neuron has a 24% probability of being activated. The determination of the activation of a neuron depends on the distribution function. Based on the computed probabilities $P(\mathbf{h}|\mathbf{v})$, we sample the values $\mathbf{h}$ based on a given distribution function. The choice of the distribution function is arbitrary, but it has a significant impact on the result, as will be shown later in the experiments. In the following, $F_X(x)$ will be an arbitrary but fixed distribution function which will be used to sample the hidden states and X is a real random variable

$$\mathbf{h} \sim F_X(P(\mathbf{h}|\mathbf{v})) \tag{17}$$

Introducing a stochastic distribution function extends the neurons to stochastic neurons. The binary values of the neurons for the hidden states $\mathbf{h}$ are obtained by sampling from a distribution function F_X using the probability $P(\mathbf{h}|\mathbf{v})$. A high $P(\mathbf{h}|\mathbf{v})$ leads to a high probability of having a positive hidden state $\mathbf{h}$, whereby a low probability leads to a zero output. For instance, the Bernoulli distribution can be selected as a distribution function. This distribution only assumes the values 0 or 1. The Bernoulli distribution for our model is defined as follows.

$$\text{Bernoulli}_X(P(\mathbf{h}|\mathbf{v})) = \begin{cases} 0 & \text{if } P(\mathbf{h}|\mathbf{v}) < X \\ 1 & \text{otherwise} \end{cases} \tag{18}$$

In the running example in Figure 16 the Bernoulli distribution was applied to determine the hidden states. For this purpose, a random variable $X \in [0, 1]$ is generated for each neuron, if the probability of the hidden state $P(\mathbf{h}|\mathbf{v})$ is less than the random variable X, the neuron will not be activated by setting it to the value 0, otherwise it will be activated by setting it to the value 1. Using this example, the output of the sigmoid function $P(\mathbf{h}|\mathbf{v})$ can easily be interpreted as the probability of the neuron being activated. In our running example, we used the Bernoulli distribution as a distribution function, sampling the hidden state

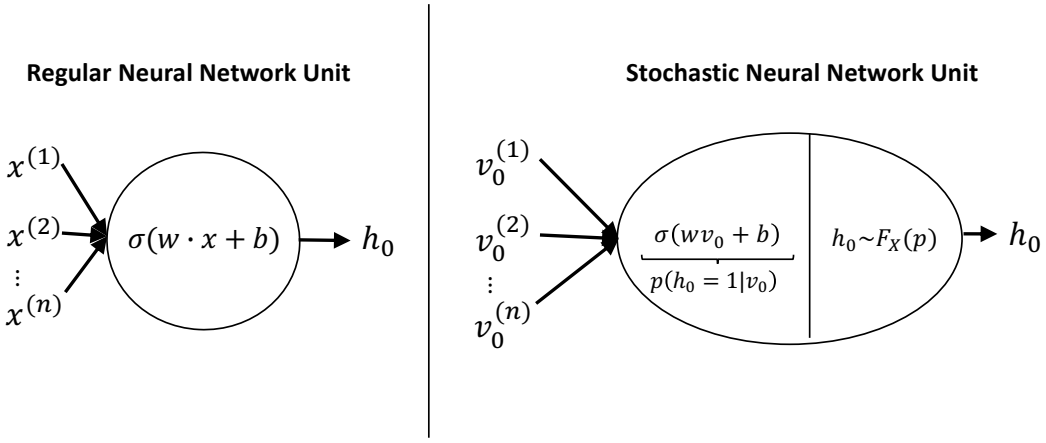


Figure 17: Comparison between a regular unit and a stochastic unit. The difference is in the application of a distribution function after applying an activation function on the weighted input.

$\mathbf{h}$ based on the vector $(0.80, 0.24, 0.92)^T$ and the Bernoulli distribution. In the current example, we have used the Bernoulli distribution, but other distribution functions such as the Gaussian or Beta distribution can be used to compute the hidden state $\mathbf{h}$ as well.

Both units are compared in Figure 17 for a better understanding of a stochastic unit and its differentiation from regular units of a neural network.

On the left in Figure 17, a regular unit is shown. It computes the weighted sum using the input x and the parameters W. Afterwards an activation function σ is applied and this result is passed on to the next layer. A stochastic unit is an extension of the regular unit by applying additionally a distribution function. First the weighted sum of the input $\mathbf{v}$ is computed using the parameters $\mathbf{W}$ and an activation function σ is applied. The activation function σ in the stochastic neural network unit is always a sigmoid function in the considered model. Afterwards the hidden state $\mathbf{h}$ is calculated using the distribution function F_X. This value $\mathbf{h}$ is then passed on to the next layer. The hidden state $\mathbf{h}$ is used to reconstruct the input data $\mathbf{v}$ in an unsupervised way, by using $\mathbf{h}$ as input and backwarded in the neural network. The hidden state $\mathbf{h}$ is multiplied with the same weight matrix $\mathbf{W}$ as it was computed and a bias value $\mathbf{b}$ added. Finally, the sigmoid activation function is applied. Since the input is a binary vector, and our goal is to reconstruct the distirbution of the inpu vector, applying a sigmoid function is useful, since its output is in the range $(0, 1)$ and can be interpreted as probabilities. The output in the visible layer can be seen as reconstruction or approximation of the original input. The formula for the reconstruction $P(\mathbf{v}|\mathbf{h})$ is the following.

$$P(\mathbf{v}|\mathbf{h}) = \sigma(\mathbf{W}^T\mathbf{h} + \mathbf{b}) \tag{19}$$

We index the input, reconstruction and hidden state since the reconstruction itself can be used as input to reconstruct them again. We denote $\mathbf{v}_0$ as initial input, according to the Definition 18, $\mathbf{h}_0$ is the hidden state based on $\mathbf{v}_0$, and $\mathbf{v}_1$ is the reconstruction of $\mathbf{v}_0$ based on $\mathbf{h}_0$. The reconstruction $\mathbf{v}_1$ can be used as input for reconstructing it again. The result of this reconstruction would be denoted as

$\mathbf{v}_2$. As this indicates, this is a Gibbs sampling process, which will later be used to learn the parameters. However, the Gibbs sampling process is run once, so the reconstruction $\mathbf{v}_1$ is not used as input for another step in the process. In contrast to the hidden layer, there is no distribution function applied in the visible layer. The right plot in the running example in Figure 16 shows the reconstruction of the input $\mathbf{v}_0$, based on the hidden state $\mathbf{h}_0$. The reconstruction can be interpreted as the likelihood that a connection between the input node represented by $\mathbf{v}_0$ and the edge $\mathbf{v}_{[j]}$ exists. Considering the running example, there is a 56% probability that an edge exists between Drug *CID000002370* and target *1128*. In contrast, according to the reconstruction $\mathbf{v}_1$, there is a 1% probability that there is an edge between Drug *CID000002370* and target *4160*. Based on the input $\mathbf{v}_0$ and the associated reconstruction by our model $\mathbf{v}_1$, the parameters $\mathbf{W}$, $\mathbf{a}$ and $\mathbf{b}$ must now be adapted to improve the reconstruction. We will go into this in more detail below and show how the learning of the parameter is done.

4.3.3 *Learning*

In typical supervised machine learning problems, a loss function is used to calculate the error between expected and predicted values. Based on this error, the parameters of the neural network are adjusted to minimize the error. This is referred to as learning in machine learning methods. However, the problem is that no labels are available, hence the error between the expected and the predicted values cannot be calculated. Furthermore, we are not interested in predicting the exact edges of the bipartite network, i.e. we do not want to restore the exact binary input vector, but rather learn a function that reflects the distribution of interactions in the bipartite network. For this reason, we will not use a loss function such as Categorical Cross-entropy based on the above energy function $E(\mathbf{v}, \mathbf{h})$, but rather, due to the unsupervised problem, we will use another loss function to minimize the error. The idea is to use a function that represents the error of the distributions between the actual and reconstructed interactions of the bipartite network.

The aim is to create the reconstructed values using samples from the model distribution. In stochastics there is the possibility to run Markov chains for an infinite time to ensure stationarity and thus a convergence to the actual distribution. However, due to infinite time, it is a very complex procedure and not suitable in practice. To counteract this there is the possibility of contrastive divergence (CD-k), in which instead of sampling from the RBM distribution, a Gibbs chain is run for only k steps. Contrastive divergence is an approximation of the log-likelihood gradient that has been found to be a successful update rule for training RBMs. Approximating the gradients using CD learning has become a standard way to train RBMs [31]. k denotes how often gibbs chain is performed for a single sample. In related work, as well as in our experiments, it has been demonstrated that $k = 1$ is sufficient for a good performance. Therefore, we use k-step contrastive divergence (CD-k) with $k = 1$ for learning the parameters of the RBM. Using a single iteration, $k = 1$, we will determine the error between the actual distribution and the distribution of our model using CD and adapt

the parameters based on this error. Using CD to learn the parameters, the time complexity is in $O(n^2)$, where n is the number of neurons in the hidden layer $\mathbf{h}$.

The input of the RBM is denoted as $\mathbf{v}_0$ (initially $\mathbf{v}_0 = \mathbf{v}$), the reconstruction and thus the output is denoted as $\mathbf{v}_1$ and is computed by $P(\mathbf{v}|\mathbf{h})$. The gradient w.r.t. log-likelihood for one training pattern $\mathbf{v}_0$ is then approximated by the following formula[16].

$$CD(W, \mathbf{v}_0) = \underbrace{-\sum_{\mathbf{h}} P(\mathbf{h}_0|\mathbf{v}_0)\frac{\partial E(\mathbf{v}_0, \mathbf{h}_0)}{\partial W}}_{\text{actual distribution}} + \underbrace{\sum_{\mathbf{h}} P(\mathbf{h}_0|\mathbf{v}_1)\frac{\partial E(\mathbf{v}_1, \mathbf{h}_0)}{\partial W}}_{\text{computed distribution}} \tag{20}$$

The difference between these two distributions should be minimized and is this our loss function for our model. Having this loss function, we can apply learning algorithms, i.e. Gradient Descent and its extension like e.g. Adam Optimizer [51], for adapting the parameters in the model. The basic equation for adapting a parameter $\mathbf{W}$ using the Gradient Descent Approach is the following.

$$\begin{aligned} \mathbf{W} &= \mathbf{W} - \alpha\Delta\mathbf{W} \\ \Leftrightarrow \mathbf{W} &= \mathbf{W} - \alpha\frac{\partial CD(W, \mathbf{v}_0)}{\partial \mathbf{W}} \end{aligned} \tag{21}$$

Based on the Gradient Descent Approach and the equation in 20 we get the following updates of the parameters:

$$\Delta\mathbf{W} = P(\mathbf{h}_0 = 1|\mathbf{v}_0) \cdot \mathbf{v}_0 - P(\mathbf{h}_0 = 1|\mathbf{v}_1) \cdot \mathbf{v}_1 \tag{22}$$

$$\Delta\mathbf{a} = \mathbf{v}_0 - \mathbf{v}_1 \tag{23}$$

$$\Delta\mathbf{b} = P(\mathbf{h}_0 = 1|\mathbf{v}_0) - P(\mathbf{h}_0 = 1|\mathbf{v}_1) \tag{24}$$

We use mini-batch with Adam Optimizer [51] to update the parameters. The parameters converge so that the distribution of the reconstructions $\mathbf{v}_1$ corresponds to the distribution of the input $\mathbf{v}_0$. Repeating the learning for a large number of iterations converges the error $CD(W, \mathbf{v}_0)$. After applying 1,000 iterations on the running example, the output of the reconstruction of the binary input vector of node *CID0000002156* is the following $(0.96, 0.97, 0.79, 0.98, 0.99, 0.04)$, shown in Figure 18. Based on this output, we will show below how we predict the missing interactions in the bipartite network.

4.3.4 *Predictions*

We predict missing links for a given node based on the reconstruction of a node's link distribution in the network. The input into our model is a binary vector representing the links of an entity to the target nodes in the complex bipartite

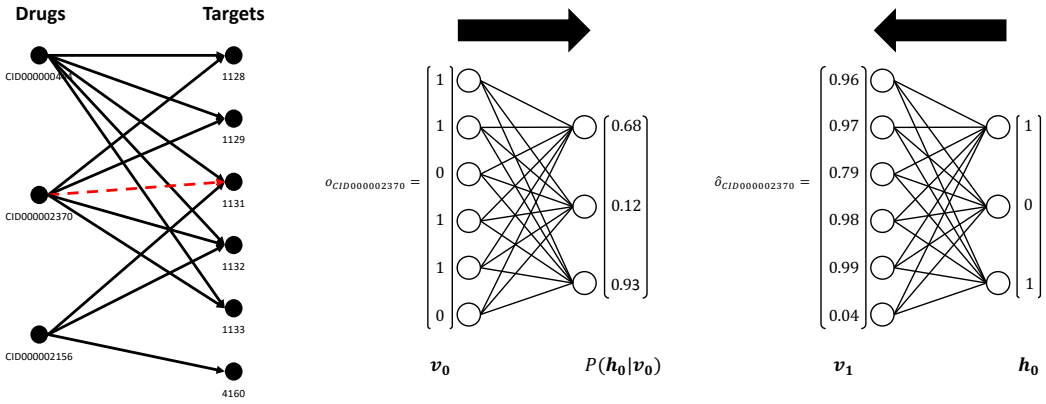


Figure 18: Computation of the output of the model, based on an input and the Bernoulli distribution in the hidden layer.

network (see Figure 18), denoted as $\mathbf{v}_0$. Based on the binary vector and a distribution function in the hidden layer, we compute the hidden states in the hidden layer (see Equations 25 and 26), denoted as $\mathbf{h}_0$. The hidden states are fed back into the RBM to reconstruct the input data (Equation 27). The output, denoted as $\mathbf{v}_1$, can be interpreted as probability of the node's likelihood of having a link to the entity of the other set. We exclude already known links in $\mathbf{v}_1$, when predicting new connections. Considering the running example, we already know about interactions from node *CIDoooooo2156* to other nodes in the network. These existing interactions are excluded from predictions and we take, from the other possible interactions, those with a probability higher than 0.5, since the likelihood is closer to a possible interaction than that there is no interaction. Following the example from Figure 18, we get a likelihood of 79% of a missing link between nodes *CIDoooooo2370* and *1131*, thus an interaction between the two nodes will be predicted.

4.4 EXPERIMENTAL STUDY

In the following we will evaluate our approach with respect to its efficiency on eight bipartite networks. The evaluation is based on well-known metrics, which are commonly used in related work. By using known metrics of related work, we enable the comparability of our method. After the presentation of the experimental setup we will present the results. Based on the results, we will then critically evaluate our method in a discussion.

4.4.1 *Experimental Setup*

Datasets. We use eight well-known complex bipartite networks from the biomedical domain: ChG-ID, ChG-Miner, ChG-TD, ChSe-D, DCh-Miner, DF-Miner, DG-AM, DG-Miner. While our approach is not domain specific, we have chosen this domain due to the large number of available networks. The networks differ in their graph structure, such as number of nodes, number of interactions and density distribution of the interactions of the nodes. An overview of the bipartite

Table 8: Overview of the bipartite networks on which the experiments were conducted.

Metric	ChG-ID	ChG-Miner	ChG-TD	ChSe-D	DCh-Miner	DF-Miner	DG-AM	DG-Miner
#Entities V1	1,774	5,018	284	639	5,535	4,294	519	5,664
#Entities V2	7795	2325	3648	10,184	1,662	16,255	7,294	17,822
#Interactions	131,034	15,139	18,690	174,977	466,656	802,760	21,357	15,509,619

networks is given in Table 13. All datasets are available online[13]. When splitting the data, we use for each conducted experiment a different random seed, but used the same seeds for the available datasets. Thereby, we guarantee that the splits differ in each experiment, but the same data for training and testing is applied to each method. We tested the methods under the assumption of a connected network as well as under the assumption of a network with several components.

Metrics. Following related link prediction models, we use Area-under-Curve (AUC) and Precision-Recall-Curve (PR) as evaluation metric. We conducted each experiment ten times and report on the average results.

Baselines. We compare our approach Link Distribution Learning (LDL) with current state-of-the-art models in Link Prediction. The models include strong baselines, e.g., AttentionWalk (AW) [3]. The selection of the models was based on their focus on biparite networks (e.g. BiNE [39]), excellent performance and novelty, e.g., node2vec (N2V) [42] and VAE [52]. In addition, we studied popular heuristics: Jaccard Coefficient (JC) [48] and preferential attachment (PA) [79].

Implementation. We implemented our method in Python3. We did not perform any hyperparameter tuning on the datasets. We used the same hyperparameter setting on each bipartite network. We chose a learning rate $\alpha = 0.01$ with a hidden layer size of 100 and $1,000$ iterations. The experiments were performed on a server with Intel(R) Xeon(R) Gold 6142 CPU@ 2.60GHz, 32 physical cores and 188GB RAM.

4.4.2 *Area Under the ROC Curve (AUC)*

We study the performance of the studied approaches on connected networks as well as on disconnected networks, in which the links were randomly removed so that the network falls apart into several components. The bipartite network DG-Miner already consists of three components. There is no split for the network DF-Miner which allows to remove 30% or more edges of the network so that the network still remains connected. Therefore we did not evaluate AUC Connected Network on these networks.

In the first experimental setup, we removed 30% of the links of the bipartite networks. The results of the experiments are shown in Table 9a. In the case of having a connected input network, our method achieves competitive results but cannot outperform AW, except for the dataset DCh-Miner. These results confirm

[13] https://snap.stanford.edu/biodata/index.html

Table 9: AUC of the studied approaches. On the left are the results where the network remains connected after link removal. N/A indicates that no connected training dataset could be created for the network. On the right are the results where the network is split into several components after link removal. Best results are marked in bold, second best in italics.

(a) Results of removing 30% links from the input networks.

	AUC Connected Network							AUC Disconnected Network						
Dataset	JC	PA	BiNE	N2V	VAE	AW	LDL	JC	PA	BiNE	N2V	VAE	AW	LDL
ChG-ID	.329	.938	.966	.931	.977	**.993**	*.979*	.333	*.970*	.965	*.927*	.967	**.979**	**.979**
ChG-Miner	.497	.944	.749	.917	.902	**.947**	*.925*	.496	.710	.698	.746	*.784*	.751	**.873**
ChG-TD	.247	*.990*	.938	.981	.975	**.997**	*.990*	.369	.861	.475	*.886*	.863	.834	**.988**
ChSe-D	.428	.965	.888	.649	.941	**.981**	*.969*	.429	.960	.890	.695	.931	*.964*	**.968**
DCh-Miner	.295	.968	.939	.536	.927	*.971*	**.986**	.298	.966	.938	.587	.930	*.970*	**.986**
DF-Miner	N/A	N/A	N/A	N/A	N/A	N/A	N/A	.363	.962	.941	.870	.933	*.968*	**.983**
DG-AM	.489	.961	.499	.695	.805	**.967**	*.963*	.491	*.789*	.696	.664	.713	.762	**.962**
DG-Miner	N/A	N/A	N/A	N/A	N/A	N/A	N/A	.183	.977	.968	.942	.952	.952	**.988**

(b) Results of removing 50% links from the input networks.

	AUC Connected Network							AUC Disconnected Network						
Dataset	JC	PA	BiNE	N2V	VAE	AW	LDL	JC	PA	BiNE	N2V	VAE	AW	LDL
ChG-ID	.386	.960	.961	.899	.964	**.994**	*.983*	.386	.960	.945	.903	.954	*.970*	**.981**
ChG-Miner	.498	.713	.528	.868	.806	*.874*	**.897**	.497	.713	.713	.718	*.741*	.725	**.863**
ChG-TD	.433	.874	.886	.974	.971	**.994**	*.989*	.431	.874	.633	*.888*	.869	.829	**.987**
ChSe-D	.454	.956	.850	.617	.925	**.980**	*.966*	.454	.956	.845	.699	.919	*.957*	**.965**
DCh-Miner	.331	.965	.930	.519	.912	*.971*	**.985**	.330	.965	.929	.617	.911	*.969*	**.984**
DF-Miner	N/A	N/A	N/A	N/A	N/A	N/A	N/A	.397	.961	.931	.846	.920	*.966*	**.980**
DG-AM	.495	.775	.679	.670	.701	*.930*	**.958**	.495	*.774*	.567	.675	.666	.707	**.958**
DG-Miner	N/A	N/A	N/A	N/A	N/A	N/A	N/A	.185	.977	.966	.931	.937	.942	**.989**

that attention mechanisms can effectively learn link distributions in connected networks. In contrast to the other approaches, our proposed solution LDL outperforms the other baselines in the studied datasets, including BiNE which is tailored to bipartite networks. Looking at the results of AUC Disconnected Network in Table 9a we observe that our method outperforms the other methods in this scenario. The reason for the robustness of our results, even in disconnected networks, is that the learned target distribution does not highly depend on the topology of the network. Furthermore, in general, the results of all the studied solutions decreases, compared to the experimental results when using a connected network as input. This indicates that effectively learning network representations is more challenging in the presence of several components. However, we observe that the negative impact of having a disconnected network is rather minor on LDL. AW in particular suffers significantly in disconnected networks, resulting in a strong performance loss in all studied networks except DCh-Miner. On the networks DG-AM and ChG-Miner this loss is particularly significant with a change in performance of 0.205 and 0.196. Similarly, the loss of performance of BiNE on ChG-TD is clearly apparent with a change in AUC of 0.463. BiNE suffers from a mode collapse on this dataset and predicts positive links between all given nodes, leading to this result.

The mean absolute difference (MAE) of the results with respect to AUC between the connected and disconnected networks of the baseline methods BiNE (0.119), AW (0.099), N2V(0.062) and VAE (0.058) are significantly larger than those of LDL with 0.009. Surprisingly, on some networks, N2V and BiNE even achieved a performance improvement on the disconnected networks, compared to connected networks. Results have shown that N2V and BiNE tend to predict the existence of a link between nodes in disconnected networks, compared to connected networks.

In order to further study the robustness of our method in the presence of highly disconnected networks, we conducted another experiment in which 50% of the links were removed. The results of the experiments are shown in Table 9b. The findings of this experimental setup confirm the robustness of LDL, as reported in the study with 30% link removal. Similarly to the previous study, AW still exhibits the best performance in the case of connected networks. Still, LDL achieves good results and can clearly keep up with the existing methods, achieving in most of the experimental results the second best result of AUC Connected Network. Yet, when considering the results of AUC Disconnected Network, we observe that LDL clearly outperforms the existing methods. Likewise to the findings of removing 30% of the links (Table 9a), we observe that the results of the other methods decrease significantly when having disconnected networks as input. In contrast, the results of LDL also slightly decrease, but not as strong as the baseline. The mean absolute difference of LDL compared to connected networks is 0.007, whereas AW (0.098), BiNE (0.095), N2V (0.071) and VAE (0.037) suffer much stronger performance losses. The change in performance when considering 50% removal of links is thus even smaller for LDL than for removing 30% links. This provides further evidence about the robustness of our method with respect to the connectivity of the network. When comparing results between 30% and 50% removal of links, the performance of most of the studied methods is consistently lower or equal with respect to AUC. Due to the lower amount of available network information from 50% removal of links, this result was also to be expected. Surprisingly, however, the effect of N2V and BiNE, in principle more likely to predict links between nodes, increased in the second experiment conducted, when removing 50% links, resulting in N2V achieving even better results on many disconnected datasets. Yet even when comparing removing 30% and 50% links in the bipartite networks, LDL achieves more robust performance compared to the other methods, with the result that the mean absolute difference when considering connected networks is smaller for LDL (0.013) than for the baseline methods BiNE (0.084), N2V (0.054), VAE (0.041) and AW (0.019). This robustness becomes even more evident when comparing the results between 30% and 50% links in the bipartite networks in which LDL has a mean absolute difference of 0.004, whereas the baselines BiNE (0.055), VAE (0.022), N2V (0.018) and AW (0.015) show a much higher difference.

4.4.3 *Impact of the Network Topology*

We showed that the results of the existing methods clearly depend on the connectivity of the network, while our method is much more robust with respect

Table 10: Average number of components in which the networks fall apart when randomly removing.

	ChG-ID	ChG-Miner	ChG-TD	ChSe-D	DCh-Miner	DF-Miner	DG-AM	DG-Miner
10% removal	96	637	310	205	60	167	389	238
30% removal	380	1606	945	725	195	576	1271	699
50% removal	878	2701	1595	1458	407	1192	2317	1191
Density	0.00286	0.00056	0.00241	0.00299	0.01802	0.00380	0.00070	0.05624

to the connectivity of the network. This finding is confirmed when considering both the metric AUC and PR. To support the hypothesis of robustness of our method with respect to the connectivity, we show in Table 10 the average number of components in which the bipartite networks fall apart by randomly removing links. The correlation between the number of components and the results of the random walk methods becomes obvious. The DG-AM network falls in case of 30% link removal into more than 1,200 components. The results of the random walk approaches decreases tremendously compared to a connected input network (see Table 9a). While considering DCh-Miner, the network falls apart on average into 195 components when randomly removing 30% of the links, and the results of all the approaches in the disconnected network remains very stable comparing to a connected network. This shows the huge impact of the connectivity of the input network to the results. In contrast, LDL does not rely on the connectivity of the network. Even though DG-AM falls apart into more than 1,200 components, the results of our approach remain stable, which clearly indicates the robustness of our method with respect to the connectivity. Likewise, the results of our method are stable, if the network falls apart into a few components, i.e. DCh-Miner. The same conclusions can be drawn when considering the experiments at 50% link removal. We can therefore state that our method outperforms the existing methods in the case of disconnected networks and, in the case of a highly disconnected bipartite network, our method provides much better and more robust results than the baseline methods.

Although our method LDL outperforms the baselines (c.f. Table 9a, AUC Disconnected Network), the effectiveness varies across the considered networks. This is clearly reflected in the different performance of LDL, for example, between the two bipartite networks DCh-Miner and ChG-Miner, considering the Metric 30% link removal AUC Disconnected Network (Table 9a). In order to understand the results of our method and its performance on different bipartite networks, we have conducted experiments on the network structures. These studies have shown that the density of the network has an impact on the effectiveness of our method. To elaborate more on this, we provide the density of the networks in Table 10. Considering DCh-Miner, we see that the density of this network is with 0.001802 rather high, compared to the others networks. Now considering the results in Table 9a, we see a very good performance with an AUC of 0.986 in disconnected network. In contrast, if we consider a network with a low density, i.e. the network ChG-Miner with a density of 0.00056, we achieved a performance of 0.873 on AUC Disconnected Network. We still could outperform the other methods, but compared to the results of our method on

the other disconnected networks, we achieve a lower performance with respect to AUC and PR. The same can be seen with AUC Connected Networks, meaning that our method performs very well when applied to high density networks, but is less effective on low density networks. Using the metric PR confirms exactly the same findings, LDL performs much better on a high-density bipartite network than on a low-density network. It is irrelevant whether the test split is 30% or 50%. The findings are the same in both splits.

To further support this hypothesis, we have applied our method on two additional bipartite networks (GF-Miner and GP-Miner) from the SNAP dataset. With a density of 0.00006124 and 0.00001306, these two networks each have a much lower density than the networks in our study above. Our method applied to these networks achieved on GF-Miner an AUC of 0.48 (PR: 0.578) and on GP-Miner an AUC of 0.46 (PR: 0.562). This confirms our hypothesis that LDL achieves a much better performance on denser graphs. The performance of LDL is related to the number of interactions in a bipartite network. LDL uses the existing interactions between the nodes in a bipartite network to learn a target distribution over the interactions. If there are few interactions available in a network, they are not sufficient to fit LDL adequately and thus learning a target distribution that sufficiently describes the network structure. For this reason, LDL does not perform well on sparse networks, i.e. networks that have only a very small number of interactions. In contrast, LDL performs very well on high-density bipartite networks as there are many interactions available to learn a target distribution over the interactions.

4.4.4 *Error Type Analysis*

To get a better understanding of the quality of our predictions, we illustrate the ROC of the considered methods for connected as well as disconnected DG-AM and ChG-Miner when removing 30% links in Figure 19. First of all, it can be seen that the standard deviation in the conducted experiments is very small, so the variance of the results for all methods is marginal. It can also be seen that AW can predict correct links between the nodes much faster, when having a smaller sample size in all considered networks, regardless of connectivity. As the size of the sample increases, LDL achieves a better True Positive Rate (TPR) than AW and the other methods. In the case of connected networks, LDL achieves a slightly better TPR than AW with increasing sample size. However, due to the lower TPR at the beginning, LDL achieves a lower AUC than AW in both connected networks, DG-AM and ChG-Miner. LDL suffers much less from false positives (Type I Error) than from false negatives (Type II Error). Compared to the other methods, the TPR is slightly lower at the beginning, which leads to the conclusion that we have much more Type II errors. As a result, LDL tends not to predict interactions even though they exist. Only above a certain threshold LDL is better at TPR and FPR. This inevitably leads to the fact that the precision of LDL tends to be higher, since the Recall is lower. We will go into this in more detail later when we look at the Precision Recall Curve. When comparing the ROCs of DG-AM connected and disconnected network, we see that our approach is robust and has a similar performance, despite the high number of

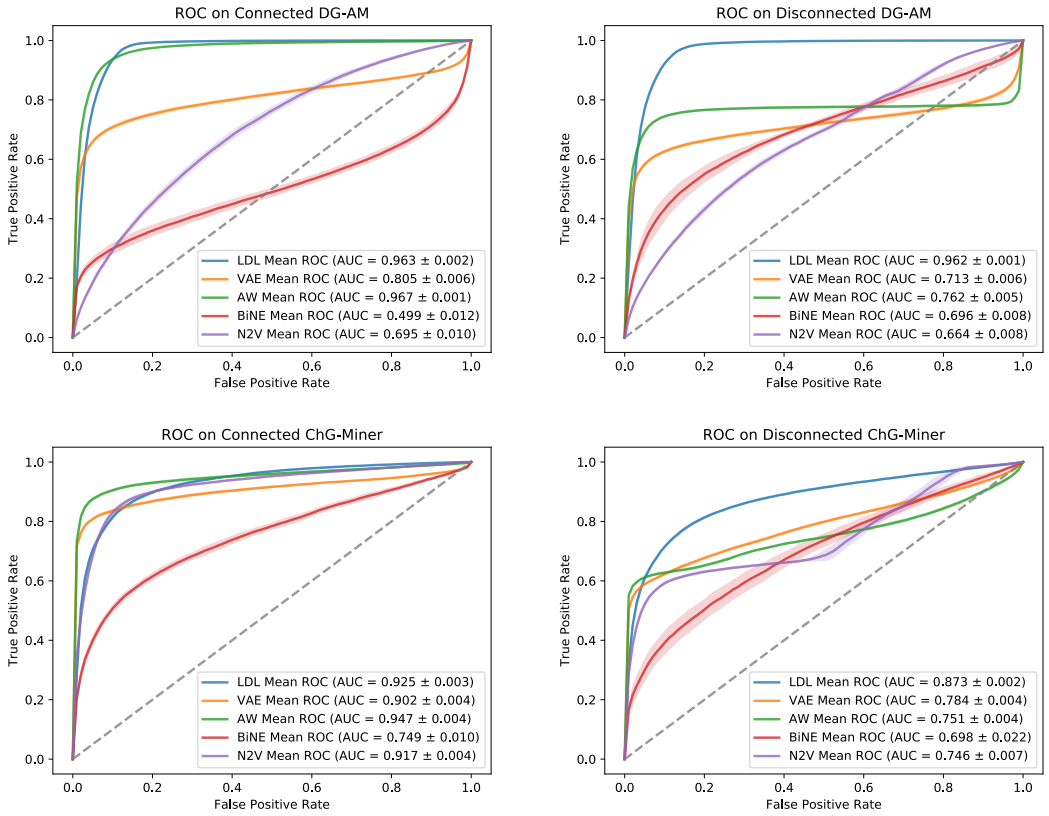


Figure 19: Average ROC for DG-AM and ChG-Miner, both for Connected and Disconnected Networks when removing 30% links. Comparing connected and disconnected networks, our method provides more robust results with respect to AUC.

components. While other models suffer significantly from the disconnection of the network, leading to lower performance, meaning that the baseline methods are no longer able to clearly identify whether a link between two nodes exist or not. It can be clearly seen how the baseline methods show a strong TPR collapse. This means that the baseline methods tend not to predict a link between the nodes, even though there should be a positive prediction. Considering ROC Disconnected DG-AM, AW achieves again a slightly better TPR with a small sample size but remains constant at a low threshold (at False Positive Rate of 0.2), so that no more links between nodes are predicted, which mostly likely is due to the number of components of the network. Likewise, when considering ROC Disconnected ChG-Miner, LDL has a lower TPR at the beginning, but outperforms the other models with increasing sample size, thus achieves better performance when considering AUC.

As mentioned above, there is a tendency for LDL to have a better precision than the other methods, but inevitably a worse recall. This relationship can be illustrated very well using a precision recall curve. Figure 20 shows, for the two bipartite networks DG-AM and ChG-Miner, the Precision Recall curves for the case of a connected network and a disconnected network. The curves are shown with a split of 0.3. In general, a high Precision value leads to a low Recall value and vice versa. Looking at the Precision-Recall-Curve of Connected-DG-AM we

see that LDL can preserve its Precision value longer, even at a higher Recall value, than e.g. AW. As already mentioned above for AUC, this is due to the fact that LDL has a lower Type I Error than the other methods. Hence LDL is much more accurate and reliable with positive predictions. However, LDL has a slightly lower recall value, because not all positive connections are identified. The Precision Recall Curve of Connected-DG-AM clearly shows that with increasing recall value, Precision can be preserved longer than compared to the baselines, but decreases significantly above a certain recall value. This decrease is much stronger than with AW, which means that the area below the Precision Recall Curve (AP) is slightly smaller than with AW. Nevertheless, it can be stated here that LDL generally has a higher precision than the baseline methods, which is why LDL should be preferred if the quality of the predictions should be very precise. However, if the focus of the predictions is more on the sensitivity, LDL is less preferable. The correlation of the improved precision over the recall becomes even more obvious when looking at the Precision-Recall-Curve of Disconnected DG-AM. As with ROC, there is a significant drop in the performance of baseline methods, compared to connected bipartite network. The area below the curves of the baseline methods is much smaller, resulting in a lower AP for the baseline methods. In contrast, the performance of LDL remains approximately the same, so that, compared to connected DG-AM, an equal performance can be achieved. Likewise, the precision can be preserved longer, but then decreases significantly at a certain recall value, so that the precision of LDL is worse than that of AW for the same recall value. The reason for this is that LDL tends to predict connections as negative, which inevitably leads to a higher false negative ratio and thus a higher Type II error. As a result, the recall is worse. However, this is not decisive when considering the area below the Precision-Recall-Curve Disconnected DG-AM, so even with a higher Type II Error, LDL outperforms the baseline methods. Nevertheless, this may have an impact when choosing a suitable method, depending on the quality of the predictions with respect to precision and recall. The progression of the Precision-Recall curve of LDL on the Connected ChG-Miner network is mostly below the baseline methods, which leads to a lower AP result compared to the baseline methods. This is caused, as discussed above, by the lower density of the network. As a result of the lower density, the network does not have a sufficient number of interactions to learn a meaningful representative distribution. The result is a lower AP compared to the baseline methods. However, considering the result on Disconnected ChG-Miner, the AP of LDL is much higher than the baseline methods. The cumulative result of LDL is therefore better than that of the other methods. LDL is capable of learning a target distribution that outperforms the baseline methods despite the lack of network connectivity. However, LDL also suffers from a stronger decrease of precision at higher recall values, so that with higher recall values the precision is worse than with AW. Nevertheless, considering AP as the cumulative result of our method, the result is significantly better than the others, so LDL outperforms the other methods.

The gained findings are confirmed when considering the other networks. The ROC and AP curves on the other networks are shown in the appendix.

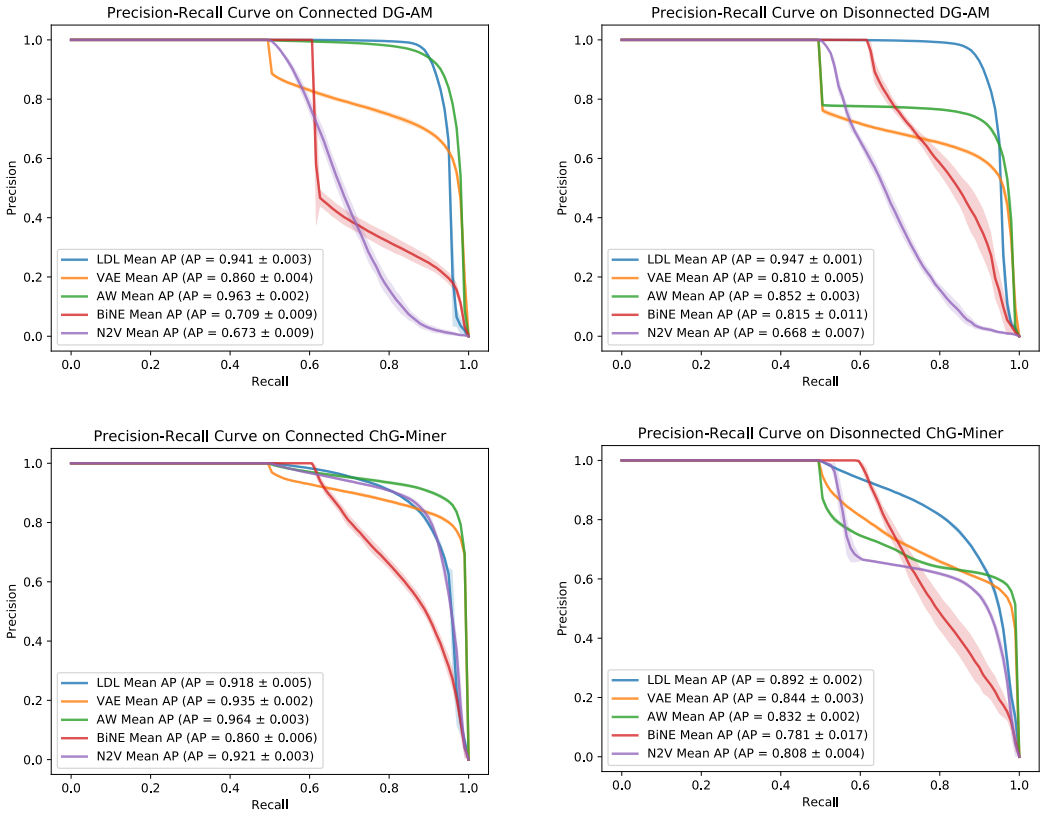


Figure 20: Average PR for DG-AM and ChG-Miner, both for Connected and Disconnected Networks when removing 30% links.

4.4.5 *Impact of the Hyperparameters on Results*

In the following part of this evaluation we want to find out to what extent the applied distribution function in the stochastic hidden units affects the performance of LDL. So far, we have used a Bernoulli distribution. However, in order to find out how the hyperparameters, especially the distribution function, affects the performance of our method, we would like to study more distribution functions. We will study different configurations of the hyperparameters, in particular the number of hidden units and the distribution function in the stochastic hidden units, and evaluate them using AUC.

Table 11 shows the results regarding AUC and PR using different distribution functions in the hidden units of LDL. The number of hidden units is the same as in the above experiments (100 hidden units) to guarantee comparability with the above results. We removed 30% of the interactions for testing. We used for the Gaussian distribution a mean of $\mu = 0.5$ and a standard deviation of $\sigma = 0.5$.

In most cases, our method achieves the best performance by using a Bernoulli distribution in the hidden layer. Using the Beta distribution in the hidden layer results only in a few cases in a better performance (in the networks ChG-ID and ChSe-Decagon). Yet this is a very small performance improvement. In general, it can be pointed out that the variations in performance between using a Bernoulli distribution and a Beta distribution are marginal, but nevertheless noticeable. Considering the performance of a Gaussian distribution in the hidden layer, the

Table 11: AUC and PR results on Disconnected Networks applying different distribution functions in the hidden layer. We removed 30% of edges from the input networks.

	AUC Disconnected Network			PR Disconnected Network		
Dataset	Bernoulli	Gaussian	Beta	Bernoulli	Gaussian	Beta
ChG-ID	.982	.853	**.988**	.981	.899	**.989**
ChG-Miner	**.873**	.677	.834	**.890**	.759	.863
ChG-TD	**.988**	.884	.960	**.989**	.923	.971
ChSe-Decagon	.968	.807	**.976**	.958	.847	**.968**
DCh-Miner	**.986**	.943	.984	**.983**	.957	.982
DF-Miner	**.983**	.827	.975	**.983**	.868	.976
DG-AM	**.962**	.804	.948	**.942**	.860	.941
DG-Miner	**.988**	.978	.819	**.984**	.973	.835

results are much lower compared to the others, causing the Gaussian distribution not to keep up with the other distribution functions. The same findings are obtained when removing 50% of the interactions from the networks for testing. The results of this experimental setup are shown in Table 12. Likewise, the Gaussian distribution cannot keep up with the other distribution functions. In most cases, our method using the Bernoulli distribution performs best in terms of AUC and PR.

Comparing removing 30% and 50% of interactions (see Table 11 and Table 12) shows that although the results are lower due to the higher split in the case of 50%, the drop in performance is not significant. The results, even when 50% of the interactions are removed, are nearly constant compared to 30%.

With respect to the varying performance of the distribution functions (cf. Table 11), we would like to focus on three more questions in the following. First, we want to understand the reasons for the lower performance of the Gaussian distribution compared to the other two distributions. And second, we want to study which distribution function converges and thus is fitted faster. In order to answer the questions, the progress of the loss function over the epochs must be considered. The loss functions of the two networks DG-AM and ChG-Miner are shown in Figure 21).

Both curves show that the Gaussian distribution has already at the beginning a loss of nearly 0. There are small volatilities over the epochs and hardly anything learned, since the loss is already very good. However, the performance using the Gaussian distribution is worse in terms of AUC and PR compared to the other distribution functions. The reason for this is the use of a very high standard deviation of $\sigma = 0.5$. This high standard deviation leads to a high variance in the hidden layer, so that the values in the hidden layer and, thus, the stochastic neural network in general, behave very unpredictably. This means that although a very precise target distribution over the interactions of the network is learned from our method (cf. very small loss in Figure 21), the predictions are subject to

Table 12: AUC and PR results on Disconnected Networks applying different distribution functions in the hidden layer. We removed 50% of edges from the input networks.

	AUC Disconnected Network			PR Disconnected Network		
Dataset	Bernoulli	Gaussian	Beta	Bernoulli	Gaussian	Beta
ChG-ID	.981	.846	**.985**	.980	.893	**.986**
ChG-Miner	**.863**	.661	.830	**.880**	.746	.858
ChG-TD	**.987**	.838	.938	**.988**	.893	.953
ChSe-Decagon	.965	.799	**.968**	.950	.842	**.959**
DCh-Miner	**.984**	.889	.981	**.981**	.912	.979
DF-Miner	**.980**	.820	.970	**.980**	.862	.971
DG-AM	**.958**	.802	.945	**.941**	.858	.939
DG-Miner	**.989**	.979	.816	**.984**	.975	.834

a very large variance in the hidden layer (cf. Table 11), which leads to a worse performance with respect to the metrics AUC and PR, compared to the other distribution functions. In order to reduce this large variance and make the predictions more accurate, we recommend reducing the standard deviation of the Gaussian distribution. This would lead to a lower variance in the hidden layer and thus to a better performance of the metrics AUC and PR.

Considering the progress of the loss functions on the networks DG-AM and ChG-Miner in Figure 21), we also notice that the distribution functions require different numbers of epochs to converge. Furthermore, they behave similarly on both networks. The Gaussian distribution models the distribution of the interactions very well from the beginning. This suggests that probably the interactions in the networks follow a Gaussian distribution. The lack in performance regarding AUC and PR is caused by a high standard deviation of the Gaussian distribution in the hidden layer, as discussed above. The Bernoulli distribution converges very fast and then remains stable over the remaining epochs with a very low loss. The reason for the fast convergence is the very high derivative of the Bernoulli distribution, leading to high adjustments. The Beta distribution function converges very slowly and requires many epochs, but finally converges as well. The reason for the slow convergence is the derivative of the function. The derivative of the Beta distribution is $\mathrm{Beta}'(x) = \frac{1}{B(p,q)} x^{p-1}(1-x)^{q-1}$ and is therefore very high for very small and very high values and small for values close to 0.5. This function converges very slowly for hidden values close to 0.5. The gradient of the Beta distribution is high in a certain range of values (approx. $[0, 0.3]$ and $[0.3, 1]$), which leads to high adjustments of the parameters and, therefore to a faster convergence. When the gradient is too small, there are only minor adjustments of the parameters, when the gradient is too big, the parameter adjustments may be too large and the network may diverge.

In the two bipartite networks considered, the parameters in the stochastic neural network are very high for this distribution function at the beginning, which

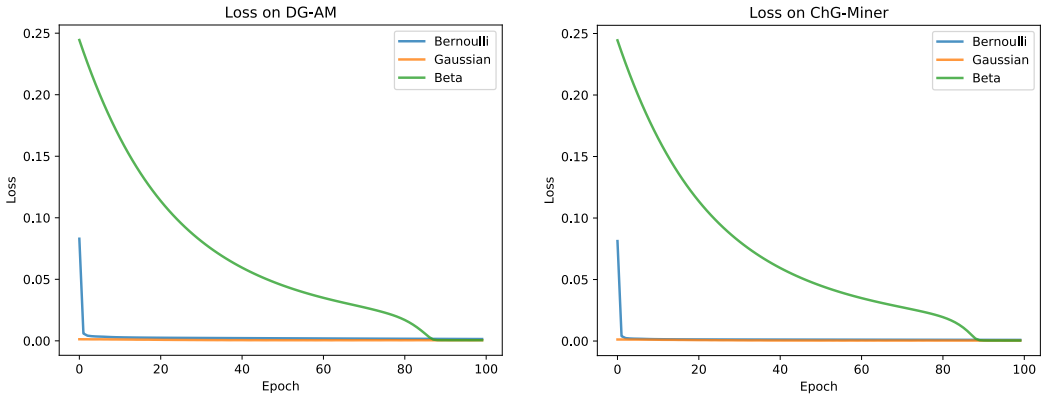


Figure 21: Progression of loss using different distribution function in the hidden layer.

leads to high adjustments. However, these adjustments cause the minimum point to be skipped, but, due to a low learning rate $\alpha = 0.01$, not so much that LDL diverges. The volatility is reduced by a high batch size in the stochastic neural network. The gradient flips around the minimum point and slowly converges towards it. Starting at an epoch of 80 of the Beta distribution, the parameters converge to zero without jumping gradients. The progression of the loss function of the Beta distribution, combined with the knowledge of the volatile gradient, indicates that the Beta distribution takes much longer to converge than the other distribution functions. Therefore, we can conclude that the Bernoulli function is recommended when a very good performance, without major hyperparameter adjustments, is desired and only few resources are available to quickly fit the model. When using the Gaussian distribution, the hyperparameters such as the standard deviation must be adjusted to guarantee good performance.

The findings from these two networks, DG-AM and ChG-Miner, are confirmed when considering the other bipartite networks. In this context we would like to point out the progress of the loss function of the Beta distribution on the network ChG-TD, where 100 epochs were not sufficient for convergence.

The remaining question we want to discuss, namely the impact of the number of hidden units, we would like to address using the Figure 22.

Figure 22 shows the AUC for different bipartite networks, based on the number of hidden units and the distribution function used. The number of hidden units chosen are: 50, 100, 300, and 500, which are very common for embedding as they represent a good compromise between a large number of units to encode a large amount of information and a small number of units to use as few resources for training and storage. Across all the bipartite networks shown, it can be seen that the Bernoulli and Beta distributions provide consistent performance regardless of the number of hidden units used. Minor variances result from low volatility during training. Nevertheless, this shows that even 50 hidden units are sufficient to learn a representative distribution over the interactions of the bipartite network, allowing to make good predictions. A smaller number of hidden units leads to a faster training, however, this has no effect on the general complexity of our method. Figure 22 also clearly illustrates the improved performance of our method using the Bernoulli and Beta distribution, considering AUC. Only on DG-Miner the Gaussian distribution can keep up with the Bernoulli distribution.

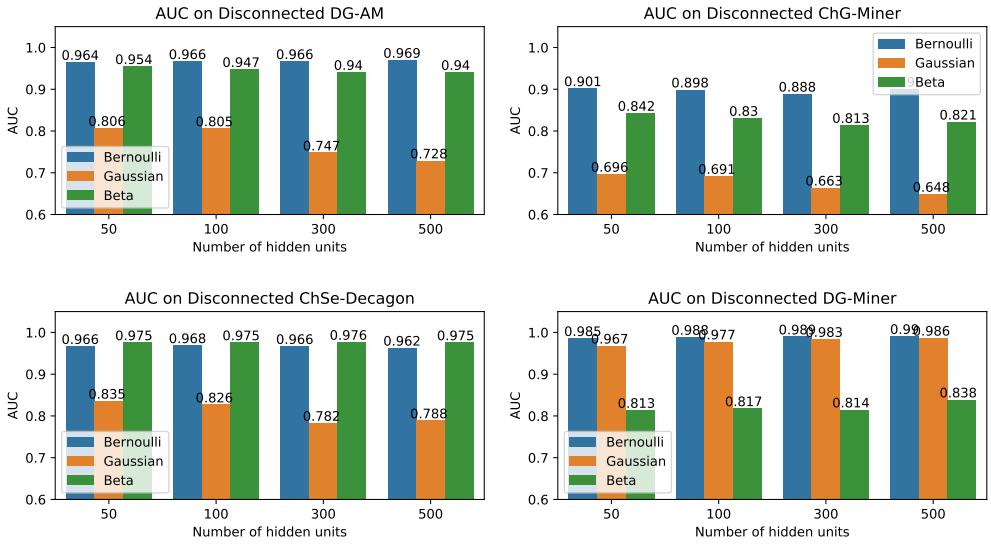


Figure 22: Performance with respect to AUC using different number of hidden units and distributions.

When considering the individual bipartite networks, the Gaussian distribution shows a high variance in the results obtained with respect to the number of hidden units. Only on DG-Miner the Gaussian distribution shows a small variance in the number of hidden units. The reason for this is the standard deviation of $\sigma = 0.5$, which is too large as mentioned above. This high standard deviation leads to a high variance in the hidden layer, so that the values in the hidden layer, and thus the stochastic neural network in general, behave very unpredictably. As the number of hidden units increases, the variance in the hidden layer increases further, causing the performance of the stochastic neural network to decrease with respect to AUC. This is best illustrated in the diagrams in Figure 22, especially when considering AUC on disconnected DG-AM. The more hidden units are present in the stochastic neural network, the lower the performance with respect to AUC. Therefore, as already described above, we recommend to reduce the standard deviation of the Gaussian distribution σ. Reducing the standard deviation of the Gaussian distribution can lead to better results with respect to AUC, as well as to a lower variance of the results with respect to the number of hidden units. In a further step it would have to be considered whether the number of hidden units also has to be reduced or whether the sole correction of the standard deviation σ leads to sufficient results. On the bipartite network DG-Miner, our method using the Gaussian distribution achieves better performance even with an increasing number of dimensions in the hidden layer. This indicates that the Gaussian distribution provides likewise constant results, regardless of the number of hidden units used, similar to Bernoulli and Beta distribution.

4.5 SUMMARY AND FUTURE WORK

In this chapter we have addressed the problem of predicting missing links on bipartite networks that may not be connected. We introduced an unsupervised

inductive stochastic factorization model for predicting missing links. The basis of our method, called LDL, was a Restricted Boltzmann Machine. We adapted it to model the characteristics of a bipartite network. Our method learns a target distribution by means of the interactions of the nodes in a bipartite network. This target distribution is used to predict missing links in the bipartite network. In contrast to existing approaches, our inductive method does not require relearning our model as the network is changed, e.g. by adding a new node.

We evaluated LDL on eight complex bipartite networks, each with different split sizes, and reported competitive results to existing methods using connected networks. Considering disconnected bipartite networks, LDL outperformed the existing methods for all networks without exception. The results were much more robust with respect to the connectivity of the networks compared to the baseline methods. An essential aspect contributing to the better performance on bipartite networks and the robustness with respect to the connectivity is not to rely on random walks like most of the existing methods. In disconnected networks, random walks starting from a fixed node, cannot reach all nodes since the network is divided into different components, resulting in a worse performance. However, we have also shown that the performance of LDL highly depends on the density of the network, i.e. the number of interactions. Given a high-density network, i.e. many interactions within the network, LDL can learn a representative target distribution across the interactions of the network, resulting in very good performance with respect to AUC and PR. Given a low number of interactions in a network, however, this superior performance cannot be assured, meaning that the performance of LDL will be lower. We, therefore, recommend LDL without exception for disconnected bipartite networks and, regardless of the connectivity, for very dense bipartite networks.

In the further part of the experiments, we conducted different studies regarding the impact of the hyperparameters, with special focus on the distribution function used in the hidden layer. The findings showed that the Bernoulli distribution is generally the preferred choice as it does not include further hyperparameters, performs very well and converges very fast, leading to a shorter training time. Another tested function, the Beta distribution, also performed very well with respect to AUC and PR, but converged more slowly than the Bernoulli distribution, resulting in a longer training time. We therefore recommend the Bernoulli distribution before the Beta distribution. Likewise, we studied LDL with respect to the usage of the Gaussian distribution in the hidden layer. The Gaussian distribution performed worst, but, as stated in the experiments, due to a too high standard deviation. The high standard deviation caused the values in the hidden layer to be subject to a high variance. Although the target distribution was promising, the AUC and PR performance was not sufficient to keep up with the other distributions. Only on the complex bipartite network DG-Miner the choice of $\sigma = 0.5$ was suitable to keep up with the other distributions. The varying performance on the different networks shows the increased effort caused by additional hyperparameters. The additional hyperparameters, μ and σ of the Gaussian distribution must be adapted to each of the bipartite networks to achieve optimal performance. The number of hidden units used had only a minor impact on the performance with respect to AUC and PR, regardless

of the distribution function used. The performance was almost identical in all conducted experiments. However, when using the Gaussian distribution, if the choice of $\sigma = 0.5$ was too high, an increased number of hidden units caused the variance in the hidden layer to increase, which in turn had the effect of reducing the performance of AUC and PR.

Given the introduced approach and the insights gained in the experiments, we can answer the research question as follows.

◉ Answering Research Questions

Hypothesis: A stochastic factorization model is able to learn the distribution of links, even in disconnected bipartite networks.

2.1 How effective are stochastic factorization models in link prediction, even in disconnected bipartite networks?

Answ. Stochastic factorization models achieve a comparable effectiveness compared to existing methods and a significantly better performance on disconnected networks than existing methods.

2.2 What are the characteristics of bipartite networks that allow stochastic factorization models to effectively learn the distribution of links?

Answ. More dense networks allow a more effective learning of a distribution function.

2.3 What is the impact of the hyperparameters, especially the distribution function in the stochastic unit, on the evaluation measurements?

Answ. Hyperparameter, especially the distribution function in the stochastic unit, has a significant impact on performance. More complex distribution functions require an adjustment of the hyperparameters depending on the network to ensure high performance.

Based on related problems and the findings of the conducted experiments, we want to study in the future to what extent the learned parameters are suitable as embeddings, i.e. as representation of nodes. As evaluation criterion we will use the classification of nodes. Our hypothesis is that nodes which have a similar target-distribution of links can be grouped into the same class. In the next chapter we will analyse this to a certain extent, but not for complex bipartite networks. In addition, we want to analyse to what extent the computed target distribution of the bipartite networks can be used to approximate or support the computation of node similarity. Our hypothesis here is that similar nodes have a similar distribution over the interactions of the bipartite network. Thereby we want to exploit the learned target distribution over the interactions in the bipartite network to specify similarities between nodes. These similarities can be used in subsequent procedures, i.e. to improve the predictions of missing links in other methods. Many methods are built on the basis of similarities. semEP [87] is a method that exploits similarities between nodes to predict missing links in bipartite networks. Another method is SimTransE [68], which considers the simi-

larities between nodes and the topology of the bipartite network to predict missing links. However, these methods can only be used if similarities between nodes are present. Since not all bipartite networks have pre-calculated similarities between nodes, these methods cannot always be applied. Using our learned target distribution, we could specify the similarities between the nodes and transfer them to the downstream procedures like semEP and SimTransE. We can therefore close an important gap and apply these methods to a wider range of bipartite networks, especially to those networks on which these methods could not be applied due to missing pre-calculated similarities.

In addition to these application scenarios, there is the potential to apply our method to networks in general, without the special focus on complex bipartite networks. Likewise to the problem of missing links in complex bipartite networks, a target distribution between the interactions of the nodes could be learned to predict missing links in networks. Extending our method to networks in general opens it to a wider range of applications. Although this allows to apply the method to a larger number of networks, other network characteristics that are not present in bipartite networks have to be considered. Thus, the restriction that interactions are only possible between two different types of nodes no longer applies. As a result, the density could be much lower, i.e. there are fewer interactions in comparison to nodes in the network. As shown in the experiments, this lower density has a significant impact on our method, resulting in a lower performance. In order to avoid this, data augmentation techniques could be used to generate more data and thus provide more interactions for learning a representative target distribution over the interactions of the nodes in the network. Data augmentation is a common technique in Computer Vision to improve results and avoid overfitting, however to the best of our knowledge not yet applied on graph structured data. Data augmentation techniques generate, given an image, several images by flipping the image either vertically or horizontally or by cropping the image to leave only the essential part visible. Likewise, we would apply augmentation techniques to networks for generating more training samples. Currently each node in the bipartite network, represented by a vector, is passed to our method which incorporates all interactions of that node. By using augmentation techniques, we would create additional virtual nodes based on one node, containing only a subset of the interactions of the original node. Thus, we generate more samples for learning the target distribution over the interactions of the network. Using the larger amount of data, we could solve the problem of lower density and thus achieve better performance of LDL. The above mentioned augmentation technique on networks could also be applied to bipartite networks. This novel application of an existing technique could be very promising to improve the performance on networks and to avoid overfitting.

5

SEMANTIC GROUPING OF NODES AND LINKS

5.1 INTRODUCTION

Knowledge Graphs (KGs) use schema assertions or axioms to model concepts and relations that serve as the foundation for describing entities and their connections in the KG. These schema assertions typically define the meaning and associations between classes, relations, and class memberships. Furthermore, reasoners can use schema axioms to logically deduce additional facts from the KG or to detect inconsistencies between statements encoded in the KG. Therefore, having complete schema assertions in KGs is key to fully exploit the power of semantics in graphs, yet, these assertions may be incomplete due to several reasons. For example, KGs created in a pay-as-you-go fashion can suffer from this kind of incompleteness since new instances, classes, and properties are sometimes added to the KG with partial information that is available at the time of insertion. Similar situation may occur when building KGs from unstructured or incomplete sources. Furthermore, some schema assertions require domain-specific knowledge provided by experts, but manually completing schema assertions is not a feasible solution for KGs with a large number of classes or relations.

Despite the relevance of schema assertions, recent advances in KG completion have been proposed to complete instance-level statements [20, 83, 116], i.e., enhancing the descriptions of entities or instances in the KG. To address problems of schema assertions, existing solutions focus on the problem of instance type prediction which associate entities to classes in the KG. To solve this problem, either specific approaches [90] have been devised or some of the aforementioned solutions [77] have also been applied. In this work, we go beyond instance type predictions and additionally target the problem of predicting domain assertions, which state the type of entities that can participate in a relation.

In order to achieve our goal, we will extend the previously introduced approach LDL from Chapter 4 for learning representations of entities and relations in KGs, specifically tailored to predict schema assertions, i.e. instance type and domain assertions. Our proposed approach, Ridle (Relation-Instance Distribution Learning), is able to learn the distribution of relations in the knowledge graph which, in turns, allows for predicting schema assertions. The hypothesis of this work is that instances from the same class are described with the same predicates in the KG which, in turn, allows for predicting the types of similar instances or even uncovering the domain for the relations. This hypothesis is shown as a motivating example in section 5.1.2.

The underlying idea of Ridle is to exploit these characteristics and compute a target distribution over the occurrence of relations and instances to learn a compressed representation of the KG, in which this distribution is latently encoded, allowing for classifying nodes and relations. The hypothesis is expressed in the Research Question 3. We will evaluate the proposed approach by comparing it to strong baseline methods in the field of knowledge graph embedding for instance-type classification. In this context we will evaluate which information – information about outgoing edges of nodes or information about incoming edges to nodes – are better for encoding information about nodes for instance type prediction. Furthermore we will evaluate the sub-symbolic representations of relations regarding their effectiveness for predicting domain assertions.

Research Question 3

The sub-symbolic KG representation learned with our model encode latent groups of nodes and links in knowledge graphs.

3.1 How effective are the learned sub-symbolic KG representation of nodes from our models for node classification?

3.2 In the sub-symbolic representation of nodes, what kind of links – incoming or outgoing – contribute the most to node classification?

3.3 How effective are the learned sub-symbolic KG representation of links from our models for domain classification?

5.1.1 *Structure of the Chapter*

Similar to the previous chapters we will motivate this chapter in the following with an example, using it to describe the problem in more detail. In the following section, we will explain the use-case in more detail using a motivating example. Subsequently, we will discuss related work in Section 5.2 and distinguish this work from the others. In this context, we will discuss works that have been used in the past in the context of instance type and domain assertion prediction. Afterwards we will introduce the proposed approach in Section 5.3, discuss the individual components of the approach, and using the motivating example to illustrate the approach and make it more comprehensible. We will conduct evaluations in Section 5.4 in order to compare the performance of our proposed approach with other methods. We will use the insights gained from the experiments to answer the research questions in Section 5.5 and to give an outlook on further research ideas.

5.1.2 *Motivating Example*

In order to illustrate the underlying problem for this chapter, we provide a motivating example in the following. Consider the KG from Figure 23, where the entity *Donald Knuth* is not associated with any class. Yet, the description of *Donald Knuth* has some relations in common with the entity *Tom Barret* of type *Person*,

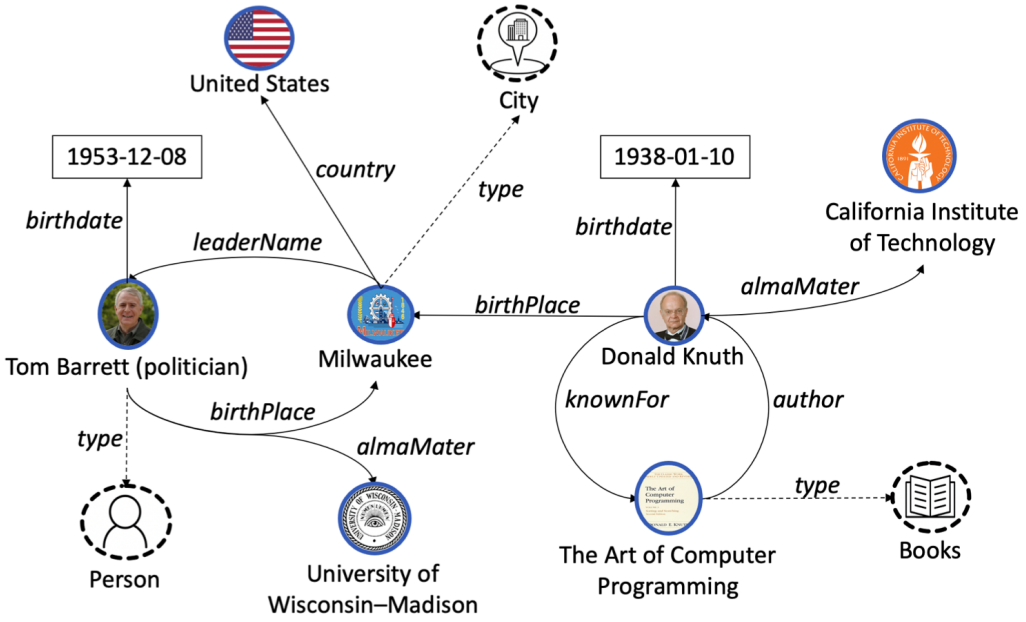


Figure 23: Motivating example. Entities from the same classes use the same predicates for description. We leverage this to predict missing type information for Donald Knuth and domain assertions of relations.

e.g., *birthDate*, *birthPlace* and *almaMater*. In contrast, *Donald Knuth* does not have relations in common with the entity Milwaukee of type *City*. Based on this information, *Donald Knuth* might also belong to the class *Person*. Note that the objects of the predicates are different for *Donald Knuth* and *Tom Barret*, still, by just looking at the relations used in their descriptions we can predict the class affiliation for *Donald Knuth*. This example shows that the occurrences of instances and relations can be an effective predictor for instance types. Thus we observe, that entities of certain classes use a similar distribution of used relations, allowing for concluding unknown classes of entities.

Further we consider the relation *author*, which originates from the book *The Art of Computer Programming*. In general, entities of type *Person* are usually author of a book, which can be latently encoded in the use of the relation in a knowledge graph. Thus besides using information of outgoing relations of entities we can use information of incoming relations as well to conclude, that *Donald Knuth* is most likely of type *Person*, as authors of books are persons in general.

In addition to the aim of predicting the instance type in this chapter, we want to predict the domain of relations. The underlying hypothesis is similar to the prediction of instance types, namely, that relations of certain domains are mostly originated from entities with a similar distribution of used relations. Consider in this context the relation *birthdate* and the assumption that we know that *Donald Knuth* is like *Tom Barret* of type *Person*. Both entities use similar relations like *almaMater* and *birthdate* for both the domain *Person* is defined. Therefore, it is reasonable to assume that *birthdate* also has *Person* as domain, as this relation is originated from entities with a similar distribution of used relations. We want to exploit this to predict the domain of relations.

Based on the above two scenarios, we define the motivation of this chapter as follows.

Motivation : Semantic grouping of nodes and links

Given a knowledge graph G, the goal is to learn a function f, indicating the probability of an instance e belonging to the class c, i.e. $f(e, c)$. Furthermore we want to learn a function to predict the probability of classes c being domain of relations r, i.e. $f_2(r, c)$.

5.2 RELATED WORK

Our work focuses on learning a representation for entities and relations in knowledge graphs for predicting instance types and domain assertions. Previous work has focused particularly on instance type prediction. Different approaches have been used, including SDType [90] as heuristic link-based type inference mechanism and RDF2Vec [102], an approach for learning latent numerical representations of entities in RDF graphs. As pointed out in related works [49, 96] traditional reasoning methods tend to struggle with noisy data, false or unforeseen schemas. The heuristic method SDType uses a statistical distribution of the actually used scheme in the computation and thus making it more robust. This characteristic of robustness of statistical distributions is used in this paper as well, in order to deal with noise in the data, but in contrast to SDType we use a stochastic factorization model learn representations of entities and relations based on the use of relations of instances. Based on the representations, we learn a model for predicting instance types and domain assertions respectively. RDF2Vec [102] is an adaptation of the language model Word2Vec [74, 75] by using random walks and Weisfeiler-Lehman Subtree RDF graph kernels to create sequences of nodes that are passed to Word2Vec for learning low-dimensional numerical representations of entities and relations. Despite the promising results of previous studies on instance type prediction using RDF2Vec [17, 50, 107], we consider RDF2Vec especially suitable for measuring the semantic similarity of entities [23, 102, 103], as well as the entity alignment between knowledge graphs [10], due to the use of Word2Vec and thus the ability to represent relationships between entities accurately. Using a random walk approach, the performance of RDF2Vec varies significantly depending on the topology of the knowledge graphs and the random walk strategy. In contrast, we learn features based on a target distribution and thus do not depend on topology of the knowledge graph or the chosen random walk strategy. We prefer the low-dimensional representation of entities and relations as learned by RDF2Vec, but consider that the complete triple is not relevant for the prediction schema assertions and thus consider RDF2Vec to be too complex for this task. In a further related work on instance type prediction, entities were classified exclusively on the basis of the relations used [72]. For the sole use of the relation of entities, the results are very promising. However, this approach focuses only on the prediction of instance types and not on the representation, and it neglects the semantic relationship, as well as correlations expressed in latent features, between the relations. In contrast, we want to pro-

vide representations of entities as well as relations by using a target distribution over the usage of relations of entities and thus the identification of latent features based on relation usage of entities.

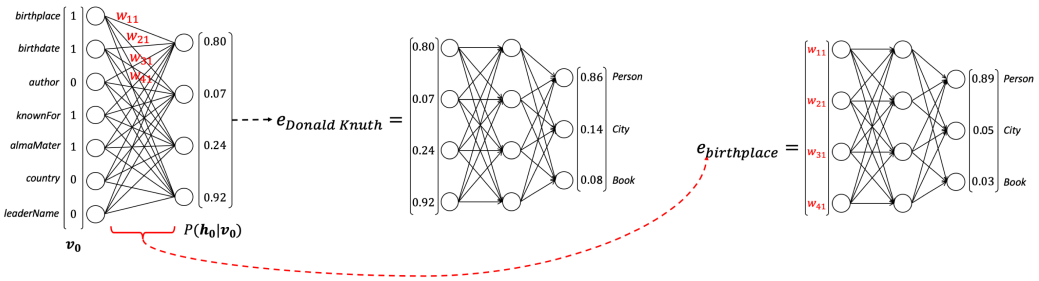
While in previous work the focus was mostly on the prediction of instance types, the prediction of domain assertions has been addressed less in the literature. There has been a first attempt to use deep learning techniques to learn semantic reasoning-RDFS rules [67]. However, the focus was less on predicting schemas based on facts, but rather on modeling RDFS rules. Besides that, most of the work focused on the completion of facts [29]. There are also a number of embedding methods, such as RDF2Vec [102], RESCAL [83], InteractE [116] and TransE [20], which can be used for predicting facts. RESCAL is a relational learning approach based on the factorization of a three-way tensor. Hereby, the representations learned by RESCAL capture the semantic meaning entities with relationships will have similar latent representations [84]. Further factorization methods were introduced [29, 36], although these methods were not exclusively used to learn representations for knowledge graph completion tasks.

We use a Restricted Boltzmann Machine (RBM) to learn a target distribution over the usage of relations of entities allowing to use the hidden layer as latent features for representation of entities and the learned weights as representation of relations. RBMs have been applied in the past especially for dimensionality reduction [46], learning and reconstructing sparse representations of the input [100], collaborative filtering [15, 104] and link prediction [60, 121]. Although first attempts were made to apply RBMs for feature learning [54, 81], it hasn't been applied for learning features in knowledge graphs. In addition we want to mention in this context, that RBMs share a similar idea as auto-encoders, but use stochastic units. Instead of reconstructing the exact input as done with auto-encoders, we are trying to identify the distribution of the used relations to determine latent features, which we will use as representation of entities.

5.3 RIDLE: RELATION-INSTANCE DISTRIBUTION LEARNING

For the downstream tasks, the prediction of instance types and domain assertion, we need a semantically meaningful representation of entities and relations. Our hypothesis is that the relations used by the instances are most relevant for their classification. For example, considering the knowledge graph in Figure 23, we notice that *Donald Knuth* has among others the relations *birthplace* and *birthdate*. Knowing only these relations, already gives a hint, that this instance is most likely of type person, as both relations are in general used only by entities of the class *Person*. The complete statements, i.e., in which place exactly *Donald Knuth* was born or at which date, is less relevant to predict the instance type. Likewise, we notice in this example, that certain relations are more likely to occur in the context of certain types of instances than others and thus there is a distribution of used relations.

We devise an approach to exploit the aforementioned two characteristics about associations between instances and relations in knowledge graphs. Our proposed approach, Ridle (Relation-Instance Distribution Learning), is able to learn the distribution of relations in the knowledge graph which, in turns, allow for



(a) Vector representation of the entity *Donald Knuth*. Learning representation for target distribution of the used relations using a RBM.

(b) Using latent features to perform instance type prediction.

(c) Using weights of each relations for domain assertion predictions.

Figure 24: Proposed approach to instance type and domain assertion prediction. Left: Representation of entities as a binary vector, encoding the usage of relations. Middle: Learning a target distribution over the used relations using RBM. Afterwards the compressed vector representation of the hidden layer $P(\mathbf{h}_0|\mathbf{v}_0)$ is used as representation of the entities and the weights $\mathbf{W}$ as representation of the relations. Right: Using 2-layer neural networks for predicting instance types and domain assertions.

predicting instance types and domains of relations. Figure 24 depicts the three components of Ridle: (a) a representation model based on instance-relation occurrences in the knowledge graph, (b) a neural network for predicting instance types based on the learned entity representations, and (c) a neural network for predicting domain assertions based on the latent information of instance-relations associations. In the following, we describe each of these components.

5.3.1 *Learning Instance-Relation Representation*

The goal of the first component of Ridle is to learn a representation of a given knowledge graph. For this, we we define a knowledge graph in this chapter as follows:

Definition 19: Knowledge Graph

A knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{L}, \mathcal{T})$, is a tuple of pair-wise disjoint sets $\mathcal{E}$, $\mathcal{R}$, $\mathcal{L}$, and $\mathcal{T}$ correspond to the set of entities, relations, literals, and types or classes, respectively. A statement in $\mathcal{G}$ is modelled as a triple (s, p, o), with $s \in \mathcal{E} \cup \mathcal{R} \cup \mathcal{T}$, $p \in \mathcal{R}$, and $o \in \mathcal{E} \cup \mathcal{R} \cup \mathcal{L} \cup \mathcal{T}$.

We denote the predicates *rdf:type* and *rdfs:domain*, defined in the RDF [123] and RDFS [43] specifications, as the relations $\mathtt{type} \in \mathcal{R}$ and $\mathtt{dom} \in \mathcal{R}$, respectively. In addition, we denote $\mathcal{R}^- \subset \mathcal{R}$ the set of domain-specific relations, which excludes type, dom, and further predicates from meta-models or general-purpose ontologies.

To learn an effective instance-relation representation, our approach encodes each entity of the knowledge graph $\mathcal{G}$ based on its properties. Ridle focuses on domain-specific properties $\mathcal{R}^-$, which allow for uncovering entities with similar semantic descriptions. General-purpose predicates are left out from the entity representation as they might hinder the learning process for two reasons: (i) these predicates alone do not provide information that allow for distinguishing entities from different classes,[14] and (ii) these predicates typically occur in the majority of entities. Therefore, Ridle models an entity $s \in \mathcal{E}$ as a binary $|\mathcal{R}^-|$-vector $\mathbf{v}$, where:

$$\mathbf{v}[i] = \begin{cases} 1 & \text{if } (s, r_i, o) \in \mathcal{G}, \text{ for some } o \in \mathcal{E} \cup \mathcal{R}^- \cup \mathcal{L} \cup \mathcal{T} \\ 0 & \text{otherwise} \end{cases}$$

Note that we choose a binary representation as the frequency of the used relations is irrelevant for the downstream classification of instance types. Figure 24a shows an example of this representation of the instance *Donald Knuth* encoded as the vector $\mathbf{v}$.

Then, the binary vector $\mathbf{v}$ serves as input for a Restricted Boltzmann Machine (RBM), for which the relation distribution in a knowledge graph $\mathcal{G}$ is learned. An RBM is a generative model to simulate input distributions of binary data, consisting of one visible layer, denoted as $\mathbf{v}$, and one hidden layer $\mathbf{h}$ with size h. In RBMs, there is no explicit output layer, since the unsupervised model tries approximate the distribution of the input data. The distribution is used to compute latent features in the hidden layer, which can be seen as a compressed representation of the input data.

In RBMs, the weights or parameters between the layers represent the impact of the individual input nodes on the latent features in the hidden layer. The learning of the parameters in our approach is done by means of Gibbs-sampling. Therefore, in the following notations, the indexing is used to indicate the step of the Gibb-sampling process. The input $\mathbf{v}_0$ is multiplied with a weight matrix $\mathbf{W}$ and added with a bias $\mathbf{a}$. Similar to a feed-forward neural network, a sigmoid activation function σ is used to compute the hidden values, denoted as $P(\mathbf{h}_0|\mathbf{v}_0)$.

$$P(\mathbf{h}_0|\mathbf{v}_0) = \sigma(\mathbf{W}\mathbf{v}_0 + \mathbf{a}) \tag{25}$$

Afterwards, samples based on the computation of the hidden layer $P(\mathbf{h}_0|\mathbf{v}_0)$, are taken from a Bernoulli distribution to compute the hidden state $\mathbf{h}_0$.

$$\mathbf{h}_0 \sim \text{Bernoulli}(P(\mathbf{h}_0|\mathbf{v}_0)) \tag{26}$$

Introducing a stochastic distribution function extends the neurons to stochastic neurons. While a high $P(\mathbf{h}_0|\mathbf{v}_0)$ results in a high probability of having a positive hidden state $\mathbf{h}_0$, a low probability results in zero output. Based on the hidden

[14] Note that the object of triples are not considered in our representation. Therefore, encoding that e.g. the predicate *type* occurs in an entity is not informative to predict the class to which the entity belongs to.

state $\mathbf{h}_0$, the input data $\mathbf{v}_0$ will be reconstructed by using the hidden state $\mathbf{h}_0$ as input and backwarded in the neural network. Hereby, the hidden state $\mathbf{h}_0$ is multiplied with the same weight matrix $\mathbf{W}$ as it was computed, but transposed, and a bias value $\mathbf{b}$ added. Afterwards, the sigmoid activation function is applied to this weighted sum. The resulting vector, denoted as $P(\mathbf{v}_1|\mathbf{h}_0)$, in the visible layer can be seen as an approximation of the original input.

$$P(\mathbf{v}_1|\mathbf{h}_0) = \sigma(\mathbf{W}^T\mathbf{h}_0 + \mathbf{b}) \tag{27}$$

RBMs are energy-based probabilistic models, using a probability distribution through an energy function to measure the quality, similar to cost functions of machine learning models. The hidden layer serves as latent variable to increase the expressiveness of the model, therefore the following energy-based probabilistic function (Gibbs distribution) specifies that a certain state $\mathbf{v}$ can be observed:

$$P(\mathbf{v}) = \frac{1}{Z}\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{28}$$

where Z is the sum from all possible states and called the normalizing factor:

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{29}$$

Using the Eq. 28 we can conclude that a low energy $E(\mathbf{v},\mathbf{h})$ leads to a high probability $P(\mathbf{v})$, while a high energy leads to a low probability $P(\mathbf{v})$. In order to increase the probability $P(\mathbf{v})$ we therefore have to minimize the energy function $E(\mathbf{v},\mathbf{h})$. The energy function $E(\mathbf{v},\mathbf{h})$ for an RBM with its input $\mathbf{v}$ and hidden state $\mathbf{h}$ is the following:

$$E(\mathbf{v},\mathbf{h}) = -\mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} \tag{30}$$

The aim is to approximate the distribution, therefore the difference in the distribution of the input data $\mathbf{v}_0$ and the reconstructed input data $P(\mathbf{v}_1|\mathbf{h}_0)$ should be minimized. Thus, the energy functions described in equation 30 of these two distributions are to be aligned. In previous work it has been shown that contrastive divergence with Gibbs-sampling, as an approximation of the log-likelihood gradient, is a very efficient method to learn the parameters of the RBM to compute the target distribution [31]. The number how often the Gibbs chain is applied for a single sample is denoted by the parameter k. In related studies as well as in preliminary conducted experiments, it has been shown that $k = 1$ already achieves sufficient results in the approximation of the target distribution. Therefore, similar to related work, we use k-step contrastive divergence to learn the parameters of the RBM. The gradient w.r.t. log-likelihood for one sample $\mathbf{v}_0$ is then approximated by the following formula [16].

$$\begin{aligned} CD(W,\mathbf{v}_0) = &-\sum_{\mathbf{h}} P(\mathbf{h}_0|\mathbf{v}_0)\frac{\partial E(\mathbf{v}_0,\mathbf{h}_0)}{\partial W} \\ &+\sum_{\mathbf{h}} P(\mathbf{h}_0|\mathbf{v}_1)\frac{\partial E(\mathbf{v}_1,\mathbf{h}_0)}{\partial W} \end{aligned} \tag{31}$$

Based on Equation 31 we get the following updates of the parameters:

$$\Delta \mathbf{W} = P(\mathbf{h}_0 = 1|\mathbf{v}_0) \cdot \mathbf{v}_0 - P(\mathbf{h}_0 = 1|\mathbf{v}_1) \cdot \mathbf{v}_1 \tag{32}$$

$$\Delta \mathbf{a} = \mathbf{v}_0 - \mathbf{v}_1 \tag{33}$$

$$\Delta \mathbf{b} = P(\mathbf{h}_0 = 1|\mathbf{v}_0) - P(\mathbf{h}_0 = 1|\mathbf{v}_1) \tag{34}$$

Using the update rules the parameters converge so that the distribution of the reconstructions $\mathbf{v}_1$ corresponds to the distribution of the input $\mathbf{v}_0$.

5.3.2 *Predicting Instance Types*

We model the problem of instance type prediction as a multi-label classification problem, since entities can belong to several classes in $\mathcal{G}$. To perform the predictions, Ridle exploits the latent features of entities learned with the RBM, which are fed into a supervised learning algorithm. For every entity $s \in \mathcal{E}$ modelled as $\mathbf{v}_0$ in the RBM, Ridle obtains the learned representation $\mathbf{e}_s = P(\mathbf{v}_0|\mathbf{h}_0)$ (c.f. Figure 24b). I.e., instances in Ridle are represented as the learned probabilities of activating a hidden state. This representation was chosen over the binary vector $\mathbf{h}_0$ obtained after the Bernoulli sampling, as $P(\mathbf{v}_0|\mathbf{h}_0) \in [0,1]^h$ corresponds to the likelihood of membership to the latent features that encode classes in the KG and, therefore, carries more in information than binary values. Then, Ridle constructs a vector with the classes to which the entity s belongs to, i.e., $\mathbf{t}_s[i] = 1$ if $(s, \mathit{type}, t_i) \in \mathcal{G}$ for some $t_i \in \mathcal{T}$, $\mathbf{t}_s[i] = 0$ otherwise. The vector $\mathbf{t}_s$ is used as labels in the classification problem.

For the supervised learning algorithm, Ridle implements a 2-layer neural network,[15] with input layer size h and output layer size $|\mathcal{T}|$. In the hidden layer, Ridle uses an approximation of the GELU activation function. We propose the GELU activation function due to its excellent performance in related machine learning tasks, as shown in BERT [27]. For an input x in the network, the used approximation of the GELU function, as described in more detail in the corresponding paper [45], is defined as follows and illustrated in Figure 25:

$$\mathrm{GELU}(x) = \frac{1}{2}x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right]\right) \tag{35}$$

In the output layer Ridle applies a sigmoid function, thus the results can be interpreted as probability that an instance belongs to a certain class or type.

[15] We conducted a preliminary evaluation using neural networks with varying number of layers. However, the 2-layer setting exhibited the best F1 performance.

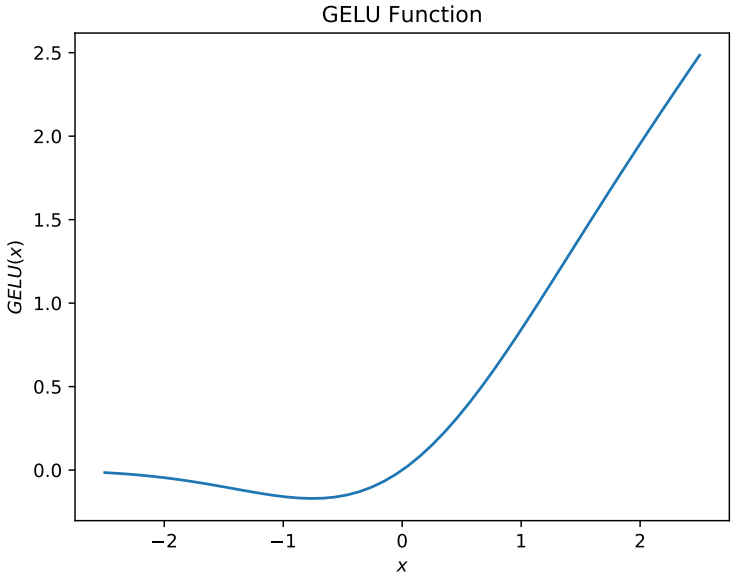


Figure 25: Approximation of the GELU activation function used in the hidden layer.

5.3.3 *Predicting Domain Assertions*

In the general case, the prediction of domain assertions can be modelled as a multi-label classification problem. Under this assumption, this problem could be solved with the architecture presented in Section 5.3.2. Yet, current KGs that are publicly available specify domain assertions using only one class per predicate. Based on this observation, we then model the problem of domain prediction as a single-label multi-class classification problem.

To perform domain predictions, Ridle exploits the latent features of relations learned with the RBM, which are fed into a supervised learning algorithm. The associations of the relations to the latent features of the hidden layer encoded in the weight matrix $\mathbf{W}$. Therefore, Ridle represents each relation $r_i \in \mathcal{R}^-$ in the KG as $\mathbf{e}_{r_i} = \mathbf{W}_{ki}$ with $k \in \{0, h\}$ (cf. Figure 24c). Due to the use of the outgoing relations of an instance, this model is particularly suitable for the prediction of domain assertions. Then, Ridle obtains the subset of classes $\mathcal{T}' \subseteq \mathcal{T}$ that occur as domain for the domain-specific relations in the KG i.e., $\mathcal{T}' = \{t \mid (r, \mathrm{dom}, t) \in \mathcal{G}$, for some $r \in \mathcal{R}^-\}$. The labels used for the classification task for a relation r_i are codified as $\mathbf{t}_{r_i}[j] = 1$ if $(r_i, \mathrm{dom}, t_j) \in \mathcal{G}$, $\mathbf{t}_{r_i}[j] = 0$ otherwise.

Similar to the prediction of instance types, Ridle implements a 2-layer neural network with the GELU [45] activation function for the hidden layer. The size of the input and output layers, in this case, is $|\mathcal{R}^-|$ and $|\mathcal{T}'|$, respectively. For the activation function of the output layer, we propose the usage of a softmax function, a generalized sigmoid function, which is typically used in single-label classification problems. In Ridle, this function can also be interpreted as the probability of assigning the domain of a relation to a class in the KG.

5.4 EXPERIMENTAL STUDY

In the following we will evaluate the introduced method with respect to the effectiveness in instace type classification and domain assertion prediction. The evaluation is based on well-known metrics, which are commonly used in related work.

5.4.1 *Experimental Setup*

Datasets. Following related work [70, 90], we use well-known public KGs such as english DBpedia [9] (3.8 and 2016-04), Wikidata (WD_2017-03-13) [117], YAGO4 [110], UMLS [18] and DBLP[16]. In the DBpedia graphs, we removed common relations including *prov:wasDerivedFrom*, *dbo:wikiPageRevisionID*, and *dbo:wikiPageID*. All those relations occur in most instances and, therefore, do not provide class-specific information. Furthermore we have removed schema assertions i.e. *rdf:type* and *rdfs:domain* to not bias the downstream prediction tasks. Given the size of the KGs and limited computational resources, we performed a data pre-processing step on DBpedia, Wikidata and YAGO4, where only a subset of entities that occur in at least 10 triples and at most in 1,000 triples are considered[17]. In addition, we extracted category-based subgraphs from DBp_2016-04 (DBp) and WD_2017-03-13 (WD) to study the performance of approaches in KGs limited to specific topics: persons (Pers), books (Books), chemical compounds (Chem), companies (Comp), movies (Mov), songs (Songs), and universities (Uni). Table 13 describes the datasets.

Metrics. We use F1-score for measuring the effectiveness of the approaches in both tasks: instance type and domain assertion predictions. We report on both F1-Macro and F1-Micro for the aggregation of multi-label performance in order to show the impact of prediction errors in more detail. In addition, we report on the hierarchical F1-Micro [70]. This metric is a micro-averaged measurement incorporating the taxonomy of the classes by extending both the correct labels and predictions using the transitive superclasses. For a resource i, we extend the predictive labels $\hat{\mathcal{T}}_i$ and the actual classes $\mathcal{T}_i^*$ with with the corresponding ancestors using the transitive closure of the subclass relation, i.e., $\hat{\mathcal{T}}_i := \hat{\mathcal{T}}_i \cup \{t_j \mid (t_i, \textit{rdfs:subClassOf}^*, t_j) \in \mathcal{G}, t_i \in \hat{\mathcal{T}}_i\}$ and $\mathcal{T}_i^* := \mathcal{T}_i^* \cup \{t_j \mid (t_i, \textit{rdfs:subClassOf}^*, t_j) \in \mathcal{G}, t_j \in \mathcal{T}_i^*\}$. We conducted each experiment by using 10-fold cross-validation and report on the average F1-score. The results can be reproduced using the k-fold cross-validator implemented in scikit-learn [91] using a seed of 42.

Baselines. We compare our approach, Ridle, with current state-of-the-art models in schema knowledge prediction. The models include strong baselines, e.g., RDF2Vec [102]. The selection of the models was based on their focus on instance type prediction in RDF data e.g. SDType [90], excellent performance, e.g., TransE [20] and RESCAL [83], as well as latest developments in link predictions, e.g. InteractE [116]. Since the focus of SDType is only on the prediction of instance types, this method was not used for predicting domain assertions. The state-of-the-art methods were used to learn a KG representation for each of

[16] https://dblp.org

[17] This pre-processing step removes *noisy* entities with too many or too few descriptions.

Table 13: Characteristics of the studied KGs. For each KG $\mathcal{G}$, $|\mathcal{G}|$=number of triples, $|\mathcal{E}|$=number of subjects, $|\mathcal{R}|$=number of relations, $|\mathcal{T}|$=number of classes, $|\mathcal{T}'|$=number of classes as domains of relations.

KG	$\|\mathcal{G}\|$	$\|\mathcal{E}\|$	$\|\mathcal{R}\|$	$\|\mathcal{T}\|$	$\|\mathcal{T}'\|$
DBp_3.8	3,246,924	31,952	10,200	294	121
DBp_2016-04	2,457,561	49,004	11,070	354	142
WD_2017-03-13	3,141,087	49,884	1,763	1,939	27
YAGO4	2,230,760	147,464	109	823	49
UMLS	6,029	135	45	46	0
DBLP	2,712,914	136,485	26	11	0
Pers(DBp)	333,296	64,423	2,239	126	1
Pers(WD)	249,059	8,400	1,509	53	28
Books(DBp)	242,989	13,361	619	17	3
Books(WD)	285,757	59,819	519	461	27
Chem(DBp)	58,952	9,674	265	5	3
Chem(WD)	268,534	16,872	339	1,008	25
Comp(DBp)	162,887	9,531	1,274	40	7
Comp(WD)	14,943	6,456	330	217	24
Movies(DBp)	416,834	69,761	959	13	0
Movies(WD)	410,295	8,807	382	74	26
Songs(DBp)	115,833	6,200	332	9	3
Songs(WD)	204,542	41,990	321	230	27
Uni(DBp)	183,700	9,029	2,021	13	5
Uni(WD)	66,182	12,133	472	274	27

the datasets described in Table 13. Then, for a direct comparison, each learned representation was fed to the same neural network architecture detailed in section 5.3.2 and section 5.3.3. The only exception is SDType, as it does not learn a KG representation, but directly produces an instance type prediction.

Implementation. We implemented our method in Python3. We used the same hyperparameter settings on every knowledge graph. We chose a learning rate $\alpha = 0.01$ with a hidden layer size of 50 and 100 iterations for learning the representations. The experiments were performed on a server with Intel(R) Xeon(R) Gold 6142 CPU@2.60GHz, 32 physical cores and 188GB RAM. For the baselines, we used the standard hyperparameters recommended by the authors. The pre-trained RDF2Vec vectors were trained by the authors [23] using Page Rank split strategy on DBp_2016-04 and is available online[18].

5.4.2 *Effectiveness of Instance Type Predictions*

In this section, we compare the correctness of the instance type predictions of the studied approaches. We analyse the F1-score (Sect. 5.4.2.1) achieved when predicting the exact statements from the test set. Then, we analyse the performance in terms of the Hierarchical F1-score (Sect. 5.4.2.2) taking into account the class hierarchies in the KG ontology, i.e., when predicting classes linked to the correct classes in the test set via sub-class relationships. In Sect. 5.4.2.3, we visualise the learned representations with Principal Component Analysis (PCA).

[18] https://zenodo.org/record/1320042#.X43POy8RrOR

Table 14: Results for predicting instance types specified with the predicates rdf:type (DBpedia) and wd:P31 (Wikidata). Bold values represent best average results.

(a) Results for cross-domain knowledge graphs

	F1-Macro						F1-Micro					
KG	Ridle	SDType	RDF2Vec	RESCAL	IntE	TransE	Ridle	SDType	RDF2Vec	RESCAL	IntE	TransE
DBp_3.8	**.840±.01**	.224±.02	.331±.02	.370±.01	.098±.01	.376±.01	**.965±.00**	.662±.01	.000±.00	.688±.00	.002±.00	.716±.00
DBp_2016-04	**.846±.01**	.222±.01	.209±.02	.317±.02	.188±.02	.371±.02	**.968±.00**	.595±.01	.000±.00	.624±.00	.000±.00	.715±.00
WD_2017-03-13	**.805±.01**	.115±.01	.774±.01	.784±.01	.784±.01	.779±.01	.590±.01	.563±.01	.000±.00	.751±.00	.752±.00	**.801±.00**
YAGO4	**.727±.01**	.056±.00	.693±.02	.623±.01	.657±.01	.621±.01	**.965±.00**	.888±.00	.643±.00	.889±.00	.725±.00	.890±.00

(b) Results for category-specific knowledge graphs

	F1-Macro						F1-Micro					
KG	Ridle	SDType	RDF2Vec	RESCAL	IntE	TransE	Ridle	SDType	RDF2Vec	RESCAL	IntE	TransE
UMLS	**.669±.04**	.064±.02	.555±.06	.617±.05	.557±.06	.387±.18	**.598±.08**	.281±.08	.315±.06	.524±.09	.318±.05	.508±.12
DBLP	**.803±.04**	.018±.00	.198±.05	.593±.00	.134±.03	.645±.01	**.995±.00**	.051±.00	.630±.01	.970±.00	.504±.01	.970±.00
Pers(DBp)	**.680±.03**	.329±.01	.210±.02	.331±.03	.212±.03	.375±.02	**.943±.00**	.880±.00	.735±.00	.842±.00	.743±.01	.879±.00
Pers(WD)	.844±.08	.322±.23	**.848±.10**	**.848±.10**	**.848±.10**	**.848±.10**	**.997±.00**	**.997±.00**	**.997±.00**	**.997±.00**	**.997±.00**	**.997±.00**
Books(DBp)	.859±.10	.603±.18	**.865±.12**	.770±.24	**.865±.12**	**.865±.12**	**.999±.00**	**.999±.00**	**.999±.00**	**.999±.00**	**.999±.00**	**.999±.00**
Books(WD)	**.734±.01**	.036±.01	.712±.02	.720±.02	.712±.02	.720±.02	**.932±.00**	.901±.00	.912±.00	.914±.00	.912±.00	.916±.00
Chem(DBp)	**.820±.19**	.729±.18	**.820±.19**	**.820±.19**	**.820±.19**	**.820±.19**	**.999±.00**	.994±.00	**.999±.00**	**.999±.00**	**.999±.00**	**.999±.00**
Chem(WD)	.765±.01	.012±.00	**.766±.02**	**.766±.02**	**.766±.02**	**.766±.02**	**.847±.01**	.816±.01	.797±.01	.798±.01	.797±.01	.831±.01
Comp(DBp)	**.762±.07**	.386±.14	.681±.13	.737±.13	.681±.13	.753±.13	**.993±.00**	.980±.00	.969±.00	.981±.00	.969±.00	.990±.00
Comp(WD)	**.828±.03**	.070±.02	.819±.03	.819±.03	.819±.03	.819±.03	**.939±.01**	.892±.01	.935±.01	.935±.01	.935±.01	.935±.01
Movies(DBp)	**.650±.18**	.331±.11	.608±.13	.608±.13	.608±.13	.608±.13	**.999±.00**	.998±.00	**.999±.00**	**.999±.00**	**.999±.00**	**.999±.00**
Movies(WD)	.785±.08	.197±.08	**.787±.07**	.685±.22	**.787±.07**	**.787±.07**	**.989±.00**	**.989±.00**	**.989±.00**	.985±.01	**.989±.00**	**.989±.00**
Songs(DBp)	**.854±.10**	.739±.10	.731±.10	.837±.10	.733±.10	.842±.10	**.990±.00**	.989±.00	.952±.00	.986±.00	.952±.00	.989±.00
Songs(WD)	**.745±.02**	.062±.01	.731±.03	.736±.03	.731±.03	.740±.03	**.917±.00**	.806±.00	.889±.00	.895±.00	.889±.00	.911±.00
Uni(DBp)	**.766±.16**	.613±.20	.708±.11	.683±.14	.708±.11	.708±.11	**.998±.00**	**.998±.00**	**.998±.00**	**.998±.00**	**.998±.00**	**.998±.00**
Uni(WD)	**.710±.03**	.047±.01	.701±.03	.704±.03	.701±.03	.704±.03	**.854±.01**	.790±.01	.824±.01	.828±.01	.824±.01	.831±.01

5.4.2.1 *F1-score Performance*

The effectiveness of the approaches in terms of the F1-Macro and F1-Micro is presented in Table 14. Overall, we can observe that none of the methods completely outperforms the other methods throughout all the studied graphs.

Considering the cross-domain KGs (cf. Table 14a), Ridle significantly outperforms the state-of-the-art methods with respect to the metric F1-Macro. This indicates that, even in the presence of large KGs with a high number of classes and relations like is the case of DBpedia, Wikidata, and YAGO, our proposed solution is still able to produce accurate predictions. The main reason for this is that the KG representation learned with the RBM model (cf. sect. 5.3.1) is able to capture the distribution of relations across the entity in the KG. This, in turn, enables the identification of entities that belong to the same classes based on the relations used to describe the entity. In contrast, the studied baselines are mostly tailored to learn statement-level representations which cannot effectively encode the knowledge about instance types when considering large KGs with a high number of classes and relations. With respect to the metric F1-Micro, we can observe that Ridle clearly outperforms the other approaches except for the Wikidata KG. In this case, Ridle cannot correctly predict entities for the most popular classes, i.e., classes with a large number of entities like *human settlement (wd:Q486972)*. The reason for this behavior is that the entities (i.e. *wd:Q13071219*)

in the most popular Wikidata classes contain a few class-specific relations, thus, affecting the performance of Ridle. In contrast, the baseline methods use additional object information while learning the KG representations, which allows for differentiating subjects from different classes. These results confirm that Ridle achieve a high performance in scenarios where entities in the KG are described with sufficient class-specific relations. In the rest of the KGs, Ridle can correctly classify entities for both large and small classes as shown with both metrics. Another important result is the low F1-Micro values achieved by RDF2Vec in DBpedia and Wikidata. In particular, RDF2Vec is always predicting *foaf:Agent* as class, which is considered a false positive according to the test data.

Next, we look at the performance of the approaches in the category-specific KGs (cf. Table 14a). We can observe that, in some KGs, the performance of all approaches is significantly higher in comparison to the cross-domain KGs. This indicates that the correct classification of entities is easier in certain classes of a given KG. In terms of average F1-Macro, Ridle outperforms the state of the art in 12 out of the 16 studied KGs. Still, in the other 4 KGs – Pers(WD), Books(DBp), Chem(WD), and Movies(WD) – we can conclude that Ridle achieves competitive performance in comparison to the best approaches when considering the difference between the average F1-Macro (in the order of 10^{-2}) and the standard deviation (in the order of 10^{-1}). In terms of F1-Micro, Ridle achieves a very high performance on average. Furthermore, the other approaches also achieve a high performance for the datasets Pers(WD), Books(DBp), Chem(DBp), Movies(DBp), Movies(WD), and Uni(DBp). These KGs are mostly characterized by having a low to moderate number of classes ($|\mathcal{T}|$ between 5 and 53), and hundreds of relations ($|\mathcal{R}|$ between 265 and 959) with the exception of Pers(WD) with $|\mathcal{R}| = 1509$. Yet, even in category-specific KGs with hundreds of classes or few relations, our approach outperforms the state of the art.

In the previous analyses, we compared Ridle to current solutions in pre-processed datasets. Nonetheless, Ridle is also computationally efficient, i.e., it is able to run in full KGs (without pre-processing) using our hardware commodities. To perform a fair comparison with the state of the art, we obtained pre-trained RDF2Vec vectors (RDF2Vec(p)) for DBp_2016-04 computed with biased graph walks. Then, we learned a representation for the same KG with Ridle. To measure the ability to transfer the encoded knowledge to other DBpedia graphs, we applied the learned Ridle and RDF2Vec(p) representations to perform predictions on DBp_3.8 and the category-specific KGs extracted from DBp_2016-04. The results are presented in Table 15, which shows that the learned representations from DBp_2016-04 with both approaches can be transferred to other graphs derive from DBpedia. Moreover, for the F1-Macro scores, we observe the same trends as in the results presented in Table 14: Ridle outperforms RDF2Vec(p) in cross-domain KGs, while achieving better or competitive average performance in the category-specific KGs. However, we observe an improvement for RDF2Vec(p) in terms of F1-Micro especially in the cross-domain KGs. The improvement of RDF2Vec(p) could be two-fold: (1) the pre-processed datasets with less entities are negatively affecting the random walks of RDF2Vec, (2) the biased graph walks of RDF2Vec(p) are actually contributing to the class prediction. To further analyse the performance of our proposed solution on full KGs, we applied

Table 15: Results for predicting instance types with representations learned from the full KG DBp_2016-04.

	F1-Macro		F1-Micro	
KG	Ridle	RDF2Vec(p)	Ridle	RDF2Vec(p)
DBp_3.8	**.880±.01**	.792±.03	**.966±.00**	.938±.00
DBp_2016-04	**.872±.01**	.736±.01	**.971±.00**	.926±.00
Pers(DBp)	**.851±.03**	.755±.03	**.982±.00**	.962±.00
Books(DBp)	.870±.11	**.876±.11**	**.999±.00**	**.999±.00**
Chem(DBp)	**.900±.10**	.880±.13	**.999±.00**	**.999±.00**
Comp(DBp)	**.809±.09**	.794±.08	**.994±.00**	.993±.00
Movies(DBp)	.662±.16	**.681±.16**	**.999±.00**	**.999±.00**
Songs(DBp)	**.863±.10**	.855±.10	**.989±.00**	.987±.00
Uni(DBp)	**.775±.13**	.683±.11	**.998±.00**	**.998±.00**

Ridle to the entire Wikidata KG,[19] achieving a performance of .806 in F1-Macro and of .590 in F1-Micro. This performance is comparable to the results on the pre-processed dataset, reported in Table 14.

Overall, when comparing the F1-Macro and F1-Micro values presented in Table 14 and Table 15, we observe that all the approaches achieve a higher performance in the F1-Micro metric. This result confirms that accurately classifying instances into popular classes is an easier tasks, as the probability of correct predictions is higher. However, predicting instance types for smaller classes is a more challenging tasks for all the studied approaches, as captured in the F1-Macro scores. The reason for this is that these classes might not have sufficient entities (and associated descriptions) to learn an effective representation from these. Yet, Ridle shows on average a better performance for this case than the state of the art.

5.4.2.2 *Hierarchical F1-score Performance*

To further understand the performance of the studied approaches, similar to the related work [70, 105], we analyse the quality of the predictions while considering the hierarchical relation of the classes in the KGs. We measure the hierarchical F1-Micro scores of the approaches. By definition, a high increase in the hierarchical F1-Micro values, indicates that the predictions are not exact but close to the actual value in the taxonomy of classes. On the contrary, a lower value than the classical F1-Micro indicates that the predictions are far away from the actual values with respect to the taxonomy. Therefore, in Table 16, we report on the difference in performance with respect to the results report in the previous section.

In the cross-domain KGs, Ridle still outperforms the others methods except in Wikidata, where now RESCAL achieves a better performance than TransE. In general, the performance of all methods – except for SDType – slightly increased

[19] To the best of our knowledge, there are no pre-trained RDF2Vec representations for Wikidata. Hence, a comparison for this dataset is not possible.

Table 16: Difference between the F1-Micro (cf. Table 14) and the hierarchical F1-Micro. N/A indicates no class hierarchy available for that KG. Values in bold indicate the highest value with respect to the hierarchical F1-Micro score.

	Hierarchical F1-Micro					
KG	Ridle	SDType	RDF2Vec	RESCAL	IntE	TransE
DBp_3.8	**+.012**	+.084	+.000	+.063	+.002	+.064
DBp_2016-04	**+.011**	+.111	+.000	+.072	+.000	+.069
WD_2017-03-13	+.042	+.124	+.000	**+.013**	+.013	+.000
YAGO4	**+.009**	+.000	+.080	+.036	+.090	+.035
UMLS	**+.019**	+.000	+.000	+.000	+.001	+.011
DBLP	N/A	N/A	N/A	N/A	N/A	N/A
Pers(DBp)	**+.014**	+.026	+.056	+.039	+.052	+.029
Pers(WD)	**+.002**	**+.002**	**+.002**	**+.002**	**+.002**	**+.002**
Books(DBp)	**+.000**	**+.000**	**+.000**	**+.000**	**+.000**	**+.000**
Books(WD)	**+.013**	+.024	+.022	+.021	+.022	+.017
Chem(DBp)	**+.000**	**+.000**	**+.000**	**+.000**	**+.000**	**+.000**
Chem(WD)	**+.024**	+.033	+.040	+.040	+.040	+.028
Comp(DBp)	**+.002**	+.004	+.007	+.004	+.007	+.002
Comp(WD)	**+.002**	+.000	+.001	+.001	+.001	+.001
Movies(DBp)	**+.001**	+.001	**+.001**	**+.001**	**+.001**	**+.001**
Movies(WD)	**+.010**	**+.010**	**+.010**	**+.010**	**+.010**	**+.010**
Songs(DBp)	**+.004**	+.004	+.019	+.005	+.019	+.004
Songs(WD)	**+.078**	+.174	+.105	+.099	+.105	+.083
Uni(DBp)	**+.000**	**+.000**	**+.000**	**+.000**	**+.000**	**+.000**
Uni(WD)	**+.098**	+.106	+.111	+.110	+.111	+.111

by around 10^{-2} w.r.t F1-Micro scores. This indicates that when the approaches produce false positives, these classes are not connected via *rdfs:subClassOf* relations to the actual classes. In contrast, SDType benefits significantly when considering hierarchies, achieving a larger improvement of 10^{-1}. These results show that SDType predictions deviate only slightly from the actual class in the taxonomy.

In the category-specific KGs, Ridle is still among the best approaches. Furthermore, when comparing the increase in performance between the cross-domain and the category-specific KGs, we observe a larger improvement for the cross-domain KGs. In the selected cross-domain KGs exist larger taxonomies, resulting in a stronger effect on the hierarchical performance than with small taxonomies as it is the case for the topic-based KGs.

Lastly, we observe in Table 16 that Ridle benefits the least (on average 0.018) from including hierarchical information in the evaluation. This indicates that the false positives produced by Ridle are generally not related to the actual classes with respect to the hierarchy. This occurs in cases where instances from different classes are annotated with a few, similar predicates in the KG. Therefore, the representation learned with Ridle does not allow for accurately distinguishing the class membership of those entities, as their representation based on entity-relation frequency is very similar.

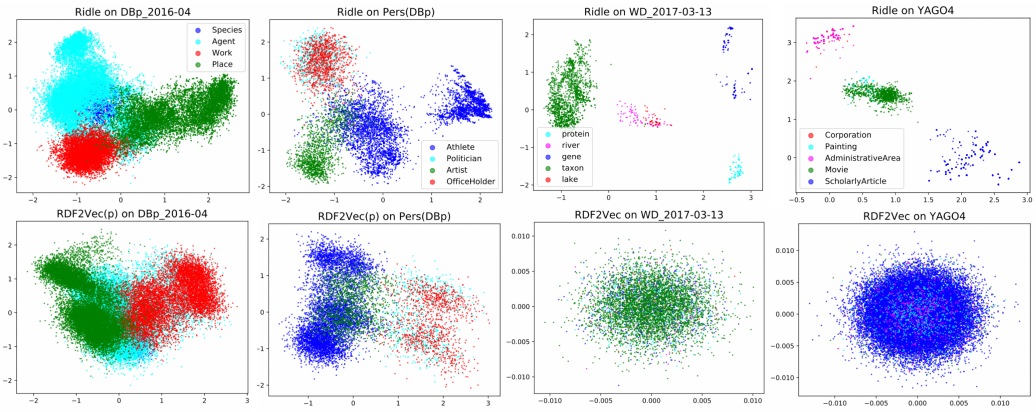


Figure 26: PCA projections for learned entity representations. Popular classes from cross-domain KGs were selected for visualization. Ridle allows for a better separation of the instances into their respective classes.

5.4.2.3 *Visualizing Entities from the Learned KG Representations.*

To get insights into the effectiveness of our solution in instance type predictions, we computed Principal Component Analysis (PCA) projections of the learned entity representations into a two-dimensional space. Besides Ridle, we present the results for RDF2Vec(p) (in DBpedia) and RDF2Vec (in Wikidata and YAGO), as two exemplary approaches that exhibit a good performance in the category-specific KGs. In the following, we analyse the results for selected classes in the KGs in Figure 26.

For the DBpedia KG, we focus on the top four most popular classes[20]. We can observe that, although the class *Agent* is broad in both representations, Ridle allows for better distinguishing the classes *Work* and *Place* from *Agent*. This separation in vectors from different classes is essential to achieve a high performance in the downstream task for predicting instance types. The representations of RDF2Vec(p), in contrast, are suggesting that the classes *Work* and *Place* are (semantically) related to *Agent*, which does not hold in DBpedia. Furthermore, the class *Species* is not visible in RDF2Vec(p) since it is covered by the representation *Agent*.

Next, we analyse the learned representations of the top four classes from the Pers(DBp) KG. In both approaches, we observe that the instances of the classes *OfficeHolder* and *Politician* are strongly interwoven and are difficult to distinguish from each other. A closer look at the instances of *OfficeHolder* and *Politician* revealed that these instances frequently use the same relations and that there is no variety of specific relations for these classes. In regards to the classes *Athlete* and *Artist*, Ridle achieves a greater separation of the computed vectors in comparison to RDF2Vec(p). The reason for this is that RDF2Vec(p) considers the object values in the triples, therefore, under this representation the classes *Athlete* and *Politician* are considered similar as their instances share in some cases the same object. This behaviour, however, negatively affects the instance type prediction capabilities of RDF2Vec(p).

[20]The DBpedia Ontology hierarchy is available at http://mappings.dbpedia.org/server/ontology/classes/

To analyse the results for Wikidata and YAGO, we selected some similar and some dissimilar classes to show the behavior of Ridle in different scenarios. In Wikidata, Ridle clearly distinguishes distant classes – i.e., *protein* and *river* – and represents closely those with semantic proximity – i.e., *river* and *lake*, which often use similar relations e.g. *located in the administrative territorial entity (P131)* and *tributary (P974)*. Similar to previous results, the YAGO instance representations of Ridle allow for distinguishing the different classes,[21] although entities using similar relations, e.g, entities of the classes *Painting* and *Movie*, are closer to each other due to the frequent use of the same relations. In contrast, RDF2Vec cannot effectively distinguish the entity types in these datasets.

Using the insights gained from the PCA projections, we can further comprehend the results for predicting the instance types presented Table 14. In the cross-domain KGs like DBp_2016-04 and YAGO4, there exist a larger number of classes whose instances are mostly described by class-specific relations. By computing a target distribution over the usage of these relations, the representation of Ridle can classify the entities more accurately on average than the baseline methods. Including the object information of the statements, as done by the baseline methods, a closer proximity of the instances is caused, leading to a more difficult classification of the instances into their correct classes. This was observed, for example, in RDF2Vec(p) with the classes *Agent* and *Place* in DBp_2016-04. By using very few class-specific relations to describe instances, Ridle can no longer distinguish between classes, causing a loss of performance with respect to the F1-score. Overall, we can conclude that when entities are described with class-specific relations, Ridle is able to obtain a representation that effectively encodes both semantically similar and dissimilar classes.

5.4.3 *Effectiveness of Domain Predictions*

In this section, we investigate the effectiveness of the approaches when predicting property domain assertions. In the studied KGs, these assertions are represented with the predicate *rdfs:domain*, except for Wikidata where the domain of relations can be expressed via the predicate *wd:P2302*.[22] Similar to the previous analyses, we first look into the F1-scores when predicting exact statements in Sect. 5.4.3.1. In sect. 5.4.3.2, we analyse the Hierarchical F1-score, in which we include the class hierarchy for evaluation. Lastly, in sect. 5.4.3.3, we visualize the learned representations with PCA. To the best of our knowledge, there are no pre-trained RDF2Vec representations of relations available. Hence, a comparison of Ridle with pre-trained RDF2Vec vectors for this task was not possible.

[21] The plot includes all the entities, but they are superimposed. Ridle maps many of the entities from the class *Scholarly Article* to the same point in space.

[22] While *wd:P2302* and *rdfs:domain* are not semantically equivalent, we follow the Wikidata data model described in https://www.wikidata.org/wiki/Wikidata:Relation_between_properties_in_RDF_and_in_Wikidata

Table 17: Using the representations of the relations for predicting domain assertions. N/A indicates that either no domain assertions were available (UMLS and DBLP) or the number of relations with domain assertions was not enough to evaluate using k-fold cross-validation (Pers(DBp), Chem(DBp) and Movies(DBp). Bold values represent best average results.

(a) Results for cross-domain knowledge graphs

	F1-Macro					F1-Micro				
KG	Ridle	RDF2Vec	RESCAL	IntE	TransE	Ridle	RDF2Vec	RESCAL	IntE	TransE
DBp_3.8-en	**.538±.05**	.002±.00	.142±.04	.014±.01	.066±.02	**.687±.04**	.041±.02	.187±.03	.041±.02	.157±.03
DBp_2016-04	**.528±.08**	.002±.00	.098±.02	.007±.01	.088±.03	**.671±.07**	.058±.02	.155±.03	.019±.01	.187±.03
WD_2017-03-13	**.275±.05**	.200±.02	.173±.01	.214±.02	.217±.02	**.620±.03**	.586±.02	.357±.02	.586±.02	.589±.03
YAGO4	**.274±.07**	.029±.02	.076±.05	.039±.03	.104±.06	**.505±.10**	.128±.06	.223±.15	.119±.07	.259±.14

(b) Results for category-specific knowledge graphs

	F1-Macro					F1-Micro				
KG	Ridle	RDF2Vec	RESCAL	IntE	TransE	Ridle	RDF2Vec	RESCAL	IntE	TransE
UMLS	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DBLP	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Pers(DBp)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Pers(WD)	**.431±.03**	.339±.05	.152±.01	.403±.05	.374±.05	**.763±.02**	.732±.03	.426±.04	**.763±.02**	.754±.03
Books(DBp)	**.567±.37**	.233±.30	.333±.37	.300±.38	.400±.33	**.650±.32**	.300±.33	.400±.37	.350±.39	.500±.32
Books(WD)	.384±.06	.334±.09	.147±.02	**.421±.08**	.347±.09	.471±.06	.421±.06	.352±.04	**.559±.04**	.428±.05
Chem(DBp)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Chem(WD)	**.406±.05**	.283±.06	.142±.03	.274±.10	.326±.07	.424±.06	.297±.05	.282±.04	**.478±.03**	.331±.08
Comp(DBp)	.478±.33	.128±.11	**.540±.28**	.350±.33	.416±.34	.550±.31	.242±.18	**.608±.26**	.417±.33	.508±.32
Comp(WD)	**.376±.12**	.369±.06	.182±.02	.368±.05	.369±.05	.521±.04	**.531±.05**	.349±.03	.504±.04	.501±.04
Movies(DBp)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Movies(WD)	.417±.13	.428±.12	.166±.02	**.477±.12**	.446±.12	.603±.06	.611±.08	.429±.04	**.663±.05**	.622±.06
Songs(DBp)	.333±.45	.533±.48	.233±.40	**.600±.49**	.433±.47	.350±.45	.550±.47	.250±.40	**.600±.49**	.450±.47
Songs(WD)	.436±.09	.360±.07	.134±.02	**.438±.07**	.379±.08	.388±.06	.254±.04	.258±.04	**.425±.05**	.315±.03
Uni(DBp)	.206±.30	.238±.14	**.359±.34**	.322±.36	.222±.17	.267±.32	.383±.22	**.450±.33**	.400±.35	.317±.23
Uni(WD)	**.413±.09**	**.413±.07**	.164±.01	**.413±.07**	.412±.07	.567±.03	**.600±.05**	.327±.02	.586±.06	.591±.06

5.4.3.1 *F1-score Performance*

The effectiveness of the approaches for this task is presented in Table 17. We observe that there is no representation that outperforms the other methods. Depending on the KG, different representations achieve a better performance with respect to the F1-Macro and F1-Micro scores.

In the cross-domain KGs, Ridle significantly outperforms the state of the art in both F1-Macro and F1-Micro scores (cf. Table 17a). In particular, Ridle achieves a good performance in the DBpedia KGs, in contrast to the Wikidata and YAGO KGs. One explanation for this is the low number of classes (27 in Wikidata and 49 in YAGO) that occur as relation domains in those KGs (cf. Table 13). Despite this, Ridle still produces more accurate predictions than the other approaches, even in the presence of a small training set.

Regarding the category-specific KGs, we observe a more heterogeneous performance. In 5 out of the 11 studied KGs, Ridle showed a better average performance in terms of F1-Macro. All these graphs are characterized for hav-

ing $\mathcal{T}' > 20$. Furthermore, in the Movies(WD) and Songs(WD) KGs (also with $\mathcal{T}' > 20$), Ridle did not achieve the highest average F1-Macro, yet, its performance is competitive with the best method IntE. In particular, IntE exhibits a good F1-Macro performance when considering category-specific knowledge graphs, as the feature permutation used by IntE allows for learning more complex relationships, especially suitable for KGs extracted from Wikidata, in which the relation *wd:P2302* is used for both, domain and range assertions. Conversely, it is challenging for Ridle to predict exact domain assertions for classes that participate as domain of several relations, as reflected in the F1-Micro results.

In general, the performance of the models with respect to F1-Micro is higher than with respect to F1-Macro. The only exception is the case of Songs(WD) in which all methods, except RESCAL, were less accurate. The improved performance of the methods with respect to the F1-Micro score is to be expected, since classes with a small number of samples tend to be predicted less correctly and classes with a large number of samples are generally predicted more accurately. The greater the discrepancy in class distributions, the greater the difference between F1-Macro and F1-Micro. Surprisingly, however, in the case of Songs(WD) F1-Micro is lower than F1-Macro, indicating that there is a frequent number of false positives in the most popular classes (i.e. *allowed qualifiers constraint (Q21510851)*) is present. This is due to the use of *wd:P2302* as an assertion, used for both domain and range constraints, and thus used in different contexts (e.g., relation *characters (P674)* to restrict to voice speaker or *ISBN-10 (P957)* to restrict the publisher), which is not considered in our model. It is therefore difficult for all models to find a distinct pattern for prediction.

5.4.3.2 *Hierarchical F1-score Performance*

Table 18 presents the difference between the hierarchical and the regular F1-Micro scores (cf. Table 17). A positive number means that predictions are close to the actual class in the hierarchy; a negative number means otherwise.

In the cross-domain KGs, Ridle is able to achieve the best hierarchical F1-scores and most approaches improve in the Wikidata and YAGO datasets. For the DBpedia dataset, the improvement is either marginal or it slightly decreases, as is the case of RDF2Vec. These results indicate that the other approaches are predicting classes that are not hierarchically related to the actual class. For example, RDF2Vec tends to predict always the class *Person*, as this is a popular class in DBpedia. We hypothesize that this behavior of the other approaches is caused by (i) including object information in the learned representation, which perturbs the encoding of the domains, and (ii) incorrect statements in the KG, as we further discuss in Sect. 5.4.5, which makes the domain prediction task more challenging. In the following, we illustrate the predictions of Ridle and TransE for DBp_2016-04; $\subseteq$ and $\supseteq$ indicate subclass and superclass of the actual class, respectively.

Table 18: Difference between F1-Micro (cf. Table 17) and hierarchical F1-Micro for evaluating the quality of the domain assertion. N/A indicates no sufficient domain assertions available for evaluation. Values in bold indicate the highest value with respect to the hierarchical F1-Micro score.

	Hierarchical F1-Micro				
KG	Ridle	RDF2Vec	RESCAL	IntE	TransE
DBp_3.8	**+.032**	-.001	-.002	-.001	+.004
DBp_2016-04	**+.021**	+.000	+.004	+.006	+.004
WD_2017-03-13	**+.301**	+.327	+.502	+.307	+.305
YAGO4	**+.129**	+.342	+.250	+.303	+.281
UMLS	N/A	N/A	N/A	N/A	N/A
DBLP	N/A	N/A	N/A	N/A	N/A
Pers(DBp)	N/A	N/A	N/A	N/A	N/A
Pers(WD)	+.193	+.223	+.466	**+.196**	+.193
Books(DBp)	**+.185**	+.000	+.100	+.150	+.000
Books(WD)	+.407	+.479	+.507	**+.358**	+.447
Chem(DBp)	N/A	N/A	N/A	N/A	N/A
Chem(WD)	+.330	**+.654**	+.562	+.401	+.530
Comp(DBp)	+.033	+.000	**+.017**	+.000	+.092
Comp(WD)	+.355	**+.375**	+.515	+.360	+.367
Movies(DBp)	N/A	N/A	N/A	N/A	N/A
Movies(WD)	+.304	+.301	+.452	**+.274**	+.264
Songs(DBp)	**+.300**	+.000	+.000	+.000	+.000
Songs(WD)	+.472	+.616	+.590	**+.464**	+.550
Uni(DBp)	+.100	+.000	**+.050**	+.000	-.034
Uni(WD)	+.330	**+.321**	+.536	+.308	+.296

Relation	Actual Domain	Ridle	TransE
placeOfBurial	*Person*	*Saint* ($\subseteq$)	*Software*
draftTeam	*Athlete*	*IceHockeyPlayer* ($\subseteq$)	*Person* ($\supseteq$)
fileSize	*Work*	*Software* ($\subseteq$)	*SpaceMission*
alumni	*EducationalInstitution*	*School* ($\supseteq$)	*MilitaryUnit*

With respect to the category-specific KGs, most of the approaches show a strong hierarchical F1-Micro improvement, particularly in the graphs extracted from Wikidata. The improvement may be due to the fact that most of the classes in the category-specific KGs are related in the ontology hierarchy. Still, in case of predicting a class that is far away from the actual class would cause a high decrease in performance, which is not the case. Furthermore, we look into the kinds of predictions (sub- or superclasses) produced by the approaches to get insights into the results. Overall, the approaches predict sub-classes in around 24% of the cases, while superclasses are predicted around 18% of the cases, which together contribute to the increase in the hierarchical F1-scores observed in Table 18.

5.4.3.3 *Visualizing Relation Representation*

To understand the performance of Ridle when predicting domain assertions, we visualize the PCA projections of relations for selected KGs in Figure 27. Similar to the insights we gained in the visualization of instance vectors, the represen-

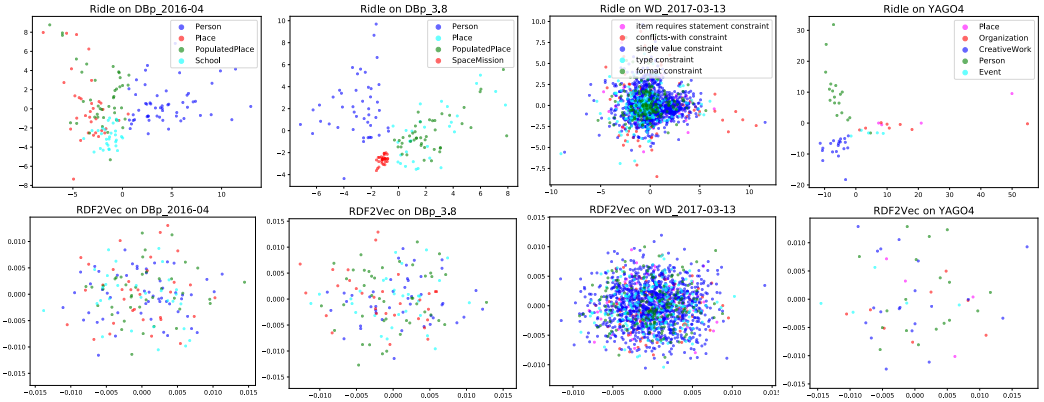


Figure 27: PCA projections for learned relation representations. Popular domain classes from cross-domain KGs were selected for visualization. Overall, the Ridle representations allow for a better separation of the relations into their respective domains.

tations of Ridle are more separable by their domain classes than the representations learned with RDF2Vec.

For the DBpedia KGs, we selected the top 4 domains as shown in Figure 27. We observe that the representations of Ridle allows for (i) encoding the semantic similarity of the relations, and (ii) providing a clear distinction between relations with dissimilar domain classes. For example, in DBp_2016-04, the relations with domains *Place* and *PopulatedPlace* are represented closely and, in fact, *PopulatedPlace* is a subclass of *Place*. Note that ontological information was not provided to the approaches during the learning process, yet, these results indicate that Ridle is able to reconstruct these semantic associations from the instance-relation distributions. This proximity that we can observe in the representation of Ridle is contributing to the increase of performance when considering the hierarchical F1-Micro score. Conversely, relations with dissimilar domains, e.g. *Person* in both DBpedia KGs and *SpaceMission* in DBp_3.8, are represented slightly further from the other relations. The insights from these KGs are also confirmed with the YAGO KG.

Lastly, the representations of Ridle for Wikidata relations do not allow for a precise separation by domain class, which explains the relatively low F1-scores reported in Table 17a. The reason for this is most likely the mixing of different constraints using the property *wd:P2302*, causing Ridle to be less effective in this KG.

5.4.4 *Impact of Encoding Incoming and Outgoing Relations*

In the following we will, in addition to the outgoing edges, also encode the incoming edges in the binary representation of the entities to include more information into the computation of learning a latent representation. The hypothesis is, that an incoming relation like e.g. *author* of a book is descriptive for a person just like the outgoing relations of this entity, so that incoming relations can be used to learn a latent reparation as well.

Table 19: Comparison of the different encoding of incoming and outgoing relations to learn latent representations for predicting instance types.

	F1-Macro		F1-Micro	
KG	Outgoing	Inc/Out	Outgoing	Inc/Out
DBp_3.8	**.840±.01**	.802±.02	**965±.00**	**.965±.00**
DBp_2016-04	**.846±.01**	.802±.01	**.968±.00**	.936±.00
WD_2017-03-13	**.805±.01**	.120±.00	.590±.01	**.828±.00**
YAGO4	.725±.01	**.729±.14**	.965±.00	**.999±.00**

Therefore we define the binary vectors of the entities $s \in \mathcal{E}$ which serve as input for Ridle as follows.

$$\mathbf{v}[i] = \begin{cases} 1 & \text{if } (s, r_i, o) \in \mathcal{G} \text{ or } (o, r_i, s) \in \mathcal{G}, \text{ for some } o \in \mathcal{E} \cup \mathcal{R}^- \cup \mathcal{L} \cup \mathcal{T} \\ 0 & \text{otherwise} \end{cases}$$

Based on these vectors we train latent representations and use them for predicting instance types. The model for predicting instance types is the same as described in section 5.3.2. Table 19 shows the results of these experiments (Inc/Out) compared to the results with the encoding described in section 5.3 using only the outgoing relations to describe the entities (Outgoing). The experiments were performed only on the cross-domain knowledge graph due to too few incoming entity relations on the category-specific knowledge graphs to get significant results.

Considering the results in Table 19 we notice that there are different results, depending on the knowledge graphs examined. In general, on the DBpedia knowledge graphs, the additional inclusion of the incoming relations leads to a deterioration of the performance with respect to F1-Macro. In contrast, the additional inclusion of the incoming relations leads to an improvement of the performance of Ridle with respect to F1-Macro and F1-Micro on YAGO4. Considering the number of incoming relations in detail, we see that DBpedia has much more incoming relations to describe the entities than for instance YAGO4. DBpedia-3.8 has a total of 3.53% triples with incoming relations to describe the entities whereas YAGO4 has only 0.435%. Thus DBpedia has more incoming relations of entities. This is confirmed by the use of incoming relations for describing the individual entities on YAGO4 compared to DBpedia-3.8. On the knowledge graph YAGO4, the individual entities have much fewer incoming edges for describing them than in DBPedia. According to the results in Table 19 it can be assumed, that the incoming relations for description, in addition to the outgoing relations, have a disturbance and thus a certain negative impact on the learned latent representations. Therefore, the performance for predicting instance type generally deteriorates with respect to F1 scores. On average, YAGO4 has a better performance with respect to F1-Macro, however, the standard deviation is also significantly higher. Taking this increased standard deviation into account, the results are not as good as expected with regard to the inclusion of incoming relations.

Table 20: Difference between the F1-Micro (cf. Table 19) and the hierarchical F1-Micro. Values in bold indicate the highest value with respect to the hierarchical F1-Micro score.

	Hierarchical F1-Micro	
KG	Ridle	Ridle-Inc/Out
DBp_3.8	**+.012**	.000
DBp_2016-04	**+.011**	.000
WD_2017-03-13	**+.301**	.000
YAGO4	**+.129**	.000

The results on the DBpedia knowledge graphs even indicate a clear deterioration of the performance with respect to F1-Macro, whereby considerably more incoming relations are used for description compared to YAGO4. This confirms, that the incoming relations are not as good as expected for the description of entities and input for Ridle and thus learning latent representations. This is confirmed by the Wikidata knowledge graph, for which the performance of the F1-Macro deteriorates significantly when considering incoming relations. The larger amount of information leads to a greater distortion, resulting in the learned representations not being effective for the prediction of instance types.

Similar to the experiments in section 5.4.2.2, we evaluated the performance of the different encodings with respect to the hierarchical F1-Micro score. The results in Table 20 confirm the findings from Table 19. Including incoming relations in Ridle to learn a representation of entities has no positive effect on performance. Instead, the additional information about incoming relations introduces noise, causing a lower performance with respect to the F1-score. A common example in the examined knowledge graphs is the relation *author*. The relation describes that a book has a certain author. The hypothesis was that this entity is usually a person, but organizations can also be authors of e.g. a tech report, leading to an incorrect prediction of the class of the entity, which results in a decrease in performance with respect to the correct classification.

5.4.5 *Final Remarks*

When comparing the achieved results of the methods for instance type and domain assertion prediction, we observe that the performance with respect to F1-score for predicting domain assertions is generally lower than the performance of instance type prediction. Based on our experimental evaluation, we identify two main reasons for this behavior: incorrect statements in KGs, and the limited number of samples available about domain assertions.

Regarding incorrect statements, there are inconsistent uses of relations with respect to the domains defined in the ontology of the KG. For example, the ontology of the DBp_2016-04 KG defines the class *SoccerClub* as the domain of the relation *ground*[23]; this domain assertion states that soccer clubs have home stadi-

[23] http://dbpedia.org/ontology/ground

ums (or grounds). However, we found the triple *(Dinamo_Zagreb_season, ground, Stadion_Maksimir)*, where the entity *Dinamo_Zagreb_season*[24] is of type *SoccerClubSeason*, and it is not an instance of *SoccerClub* or any of its subclasses. This is an example of incorrect statements that might perturb the learning process for domain assertions, especially, if the number of incorrect statements is relatively large. For example, in the latest DBpedia dataset[25], we encountered 6,026 entities from the class *SoccerClubSeason* with the *ground* predicate (i.e. incorrect usage), and 16,825 entities from the class *SoccerClub* (i.e. correct usage).[26] Based on the frequency that entities from *SoccerClubSeason* co-occur with the relation *ground*, the learning process of Ridle may also encode these inconsistencies in the obtained representation which, in turn, affect the performance of downstream prediction tasks of domains assertions.

In addition, the prediction of domain assertions is intrinsically restricted by the number of samples in the KGs. In this case, the set of labels or correct relation-domain pairs is limited to the number of predicates in the KG, which in general is relatively low. In contrast to the task of instance type prediction, KGs usually have thousands of entities that can be used during training and validation. For domain prediction, our empirical results suggest that in most of the cases, the amount of data is not sufficient for the models to produce accurate predictions, as shown with the F1-scores.

The evaluation regarding the impact of incoming relations for learning latent representations has shown that this does not lead to a significant improvement in performance. For this reason, the unconditional use of incoming relations for learning latent representations cannot be recommended. Some incoming relations like *author* are not descriptive enough and sometimes lead to a distortion of the available information.

5.5 SUMMARY AND FUTURE WORK

In this chapter, we presented an inductive stochastic factorization model to represent entities and relations of knowledge graphs (KGs), suitable for predicting schema knowledge. Our approach, Ridle, first implements an unsupervised learning model based on Restricted Boltzmann Machines (RBMs) to leverage the distribution over the usage of relations in instances in KGs. We then devise entity and relation representations based on the latent features learned with the RBM. Using the learned representations, Ridle then implements two neural network architectures for predicting instance type and domain assertions respectively. The experimental results showed that Ridle on average outperforms current state-of-the-art models in several KGs, which sets a new baseline in the tasks of predicting instance types and domain assertions. Furthermore we have shown in the experiments that the encoding of outgoing relations provides much more information than the encoding of incoming relations, leading to a better performance regarding instance type predictions. When using a hybrid encoding system for both outgoing and incoming relations, it has been shown that while the per-

[24] http://dbpedia.org/page/2009-10_NK_Dinamo_Zagreb_season
[25] Queryable at http://dbpedia.org/sparql
[26] Here we denote (in)correct usage w.r.t. the domain definition in the ontology.

formance with respect to the F1-score was slightly better regarding the instance type predictions, the performance was not significant enough to recommend this encoding system without restrictions, especially considering the higher runtime. The visualization of the learned KG representation shows that Ridle is able to correctly group entities with similar distribution of relations, whereas dissimilar entities are represented far away. This property of Ridle is key for achieving a high performance in entity prediction. Likewise, Ridle was able to reconstruct semantic associations between relations from instance-relation distributions, even though ontological information was not available during training. This learned semantic associations in the relation representation of Ridle is a decisive factor for the downstream performance of the schema predictions.

Based on the experiments performed and the insights gained from them, we can answer the research questions posed at the beginning of this chapter in the following.

◉ Answering Research Questions

Hypothesis: The sub-symbolic KG representation learned with our model encode latent groups of nodes and links in knowledge graphs.

3.1 How effective are the learned sub-symbolic KG representation of nodes from our models for node classification?

Answ. The introduced sub-symbolic KG representations could clearly show the effectiveness in the performed experiments and outperform baseline methods. The effectiveness of the representations could be shown especially on cross-domain knowledge graphs.

3.2 In the sub-symbolic representation of nodes, what kind of links – incoming or outgoing – contribute the most to node classification?

Answ. The outgoing links contribute the most to node classification. Including incoming links generally causes a distortion of information and therefore results in a lower performance regarding node classification.

3.3 How effective are the learned sub-symbolic KG representation of links from our models for domain classification?

Answ. The effectiveness of the learned sub-symbolic KG representations could clearly be shown for predicting domain assertions. As with the sub-symbolic KG representations of nodes, the effectiveness compared to the baseline methods was shown in cross-domain knowledge graphs, in which they were outperformed.

Future work may focus on studying the problem of range assertion predictions. In order to enable this, the process for learning an effective representation must be extended in two ways: (i) to encode information from the objects, and (ii) to handle literal values. Besides, we want to further study the kinds of predictions (sub- or superclasses) in instance types and domain assertions predictions

both in terms of their contribution to the hierarchical F1-score and their relevance in real-world application systems. In addition, we plan to investigate the instance representation for their usefulness in creating a taxonomy. As shown in this chapter, instances using a similar target distribution are closed in dimensional space and thus provide a reasonable representation with respect to the class membership. Based on the learned representations, a hierarchical clustering method could provide a reasonable taxonomy.

6

CONCLUSION

6.1 SUMMARY

In this thesis, we focused on learning latent features of graph structured data using Stochastic Neural Networks. We used the distribution function of the links of the nodes to encode latent information into the representations. We considered different types of graph structured data to learn latent features such as knowledge graphs in Chapter 3 and bipartite networks in Chapter 4, which were used to predict new links in the graph, as well as knowledge graphs in Chapter 5 to use the latent features to predict instance types and domain assertions for the relations. We adapted the methods according to the considered graph and problem, given the different properties of the graphs. In knowledge graphs the different types of edges are taken into account, while in bipartite networks the method and the input are adapted in such a way that they do not take into account different types of edges, as they were not present in this considered problem. The methods introduced for predicting missing properties in knowledge graphs have shown that learning a link distribution is capable of predicting missing properties even if an Open World Assumption is present. Although the distribution function encodes latent features that can be used to identify missing properties, the corresponding object cannot be identified. However, even with this information, the services have more information at their disposal that can be used in question answering, as shown in the beginning of this thesis, in the motivation. In a further downstream process the triple can be completed using existing methods such as ComplexE [115] and QuatE [131]. With the knowledge gained we can answer Hypothesis 1 introduced in the introduction as follows.

☑ Hypothesis 1

Link distribution learning is suitable for predicting missing properties in knowledge graphs which are represented under the Open World Assumption.

Learning latent features using the link distribution allows for effective predictions of missing properties in knowledge graphs, even if they follow the Open World Assumption.

In the evaluation of the method in the context of bipartite networks in Chapter 4 we have shown that the latent features learned with our method are particularly suitable for predicting interactions in disconnected networks. Previous

methods mostly use random walk approaches which cannot reach all nodes in a graph consisting of different components. As a result, only limited information is available from the corresponding components. In comparison, the method introduced in this thesis can also take into account information from different components and can therefore be recommended without restriction for disconnected networks. This allows for much more robust results with respect to the topology of the considered graph. A study to evaluate the applied distribution function has shown that the Bernoulli distribution, although very simple, achieves the best performance regarding the correct prediction of interactions. The second hypothesis, introduced in the beginning of this thesis, can now be answered on the basis of the studies conducted in this thesis.

☑ **Hypothesis 2**

A stochastic factorization model is able to learn the distribution of links, even in disconnected bipartite networks.

The introduced stochastic factorization model was able to keep up in terms of performance for the prediction of missing interactions with existing approaches in connected bipartite networks, and outperformed them when disconnected bipartite networks were considered.

Due to the excellent results in predicting links in knowledge graphs under the Open World Assumption and interactions in bipartite networks, we have adapted the method in Chapter 5 to learn latent features in knowledge graphs for entities and relations, especially suitable for instance type classification and predicting domain assertions. In particular with cross-domain knowledge graphs we could show that the latent features learned with our method can distinguish the subtleties of the different types of instances and domains of relations much better, which led to a better performance compared to the previous methods. Hypothesis 3 addressed the analysis of the encoded latent groups of nodes and links in the sub-symbolic KG representations of the knowledge graphs. Based on the experiments performed in this thesis we can now answer this hypothesis.

☑ **Hypothesis 3**

The sub-symbolic KG representation learned with our model encode latent groups of nodes and links in knowledge graphs.

Using our model, we could encode latent features about the group affiliation of nodes and links within the sub-symbolic KG representations. The experiments showed that these representations are highly effective for instance type prediction and domain assertions, especially for cross-domain knowledge graphs.

In summary, our introduced model, which learns latent features using stochastic neural networks, provides much more robust results regarding the topology of graph structured data, as well as the ability to easily apply them to different types of graphs.

6.2 OUTLOOK

Based on the insights gained from this thesis, new problems arise which can be tackled in the future. In Hypothesis 1, we have predicted links considering the Open World Assumption. This could also be extended to other graphs like bipartite networks and thus addressed to what extent the insights gained from knowledge graphs can be applied to other types of graphs. Thus, even in bipartite networks it can be assumed that the modelled information in the networks is not complete. This assumption is particularly useful for systems using an open world assumption, as for example in bioinformatics for modeling information between drugs and targets, in which it cannot be guaranteed that the available information is complete. Therefore, it would also be useful to study this assumption in the context of predicting interactions in networks, using an open world assumption.

Within this work the focus was put on general graph structured data. We specified for each hypothesis and research question which graphs we were considering in detail, for example, we considered knowledge graphs and bipartite networks. But despite this broad focus, there are many other graphs on which the developed methods can be applied. Thus, latent features in undirected graphs with multiple edges, multigraphs and hypergraphs could be learned to predict links within these graphs. The extension to the other graphs, like e.g. hypergraphs is especially important in domains where these graphs are used, such as physics and chemistry, to help identify new compounds in the graphs. Due to the different types of graphs there would be different structures available, which has to be considered when learning latent features.

In the context of learning about latent features of relations, we have focused on predicting domain assertions. Within this context the focus could be extended to predict the ranges of relations. However, the currently used approach would have to be changed to include the information of the nodes to which the relation point to. Currently the focus is on the outgoing nodes. Possibly a context-related representation of the relations would be useful, depending on the focus on predicting domain and range assertions, in order to use the appropriate representation and the knowledge encoded therein.

Furthermore, the representations of the individual entities could be aggregated according to their class affiliation to create a representation of the classes. With the help of these representations it can be tried to create class alignments between different knowledge graphs. Besides, the information of the latent features could also be used to identify prominent relations of entities within the classes, which are particularly prominent for the corresponding entities within the classes. This information can be used for entity summarization to identify class-specific relations, which are highly descriptive for entities of a specific class.

6.3 CLOSING REMARKS

This thesis attempted to learn latent features in graph structured data. While we used the learned latent features mostly for predicting missing links and classifying nodes and links, this is only a small part of what can be done with the

learned features. Using the learned features of the individual components of the graph, it is possible to aggregate them to generate a complete representation of the graph. While the models are applicable to all graph structured data, we have focused on specific graphs in answering the research questions. This allowed us to perform in-depth analyses of the methods with respect to the specific graphs, but has the disadvantage that some types of graphs, such hypergraphs, were not addressed.

Furthermore, an efficient runtime of the methods was never the focus of this work. The limitation of current matrix-vector multiplication algorithm is $O(n^2)$ using a sequential implementation. Due to the use of such an implementation, the runtime of the introduced methods are $O(n^2)$. This runtime is generally unsatisfactory within computer science. However, as already mentioned, the focus is less on the efficient runtime and more on the effectiveness of learning latent features in graph structured data, in which we have shown the advantages of a stochastic neural network over classical methods and the robustness of the results.

REFERENCES

[1] Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, 69:29 – 39, 2017.

[2] Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. Learning edge representations via low-rank asymmetric projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM'17, pages 1787 – 1796, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349185. doi: 10.1145/3132847.3132959. URL https://doi.org/10.1145/3132847.3132959.

[3] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. Watch your step: Learning node embeddings via graph attention. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 9198 – 9208, Red Hook, NY, USA, 2018. Curran Associates Inc.

[4] Maribel Acosta, Elena Simperl, Fabian Flöck, and Maria-Esther Vidal. HARE: A hybrid SPARQL engine to enhance query answers via crowdsourcing. In *Proceedings of the 8th International Conference on Knowledge Capture 2015*, pages 11:1–11:8.

[5] Maribel Acosta, Elena Simperl, Fabian Flöck, and Maria-Esther Vidal. Enhancing answer completeness of SPARQL queries via crowdsourcing. *J. Web Semant.*, 45:41–62, 2017.

[6] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB '94*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[7] Leman Akoglu. Quantifying political polarity based on bipartite opinion networks. In Eytan Adar, Paul Resnick, Munmun De Choudhury, Bernie Hogan, and Alice H. Oh, editors, *Proceedings of the Eighth International Conference on Weblogs and Social Media, ICWSM 2014, Ann Arbor, Michigan, USA, June 1-4, 2014*. The AAAI Press, 2014. URL http://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8073.

[8] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2001 – 2009, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

[9] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of ISWC*, pages 722–735. Springer, 2007.

[10] Michael Azmy, Peng Shi, Jimmy Lin, and Ihab F. Ilyas. Matching entities across different knowledge graphs with graph embeddings. *CoRR*, abs/1903.06607, 2019.

[11] I. Balažević, C. Allen, and T. M. Hospedales. TuckER: Tensor Factorization for Knowledge Graph Completion. *arXiv e-prints*, January 2019.

[12] Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. Hypernetwork knowledge graph embeddings. *CoRR*, 2018. URL http://arxiv.org/abs/1808.07018.

[13] Michael J. Barber. Modularity and community detection in bipartite networks. *Physical Review E*, 76(6), Dec 2007. ISSN 1550-2376. doi: 10.1103/physreve.76.066102. URL http://dx.doi.org/10.1103/PhysRevE.76.066102.

[14] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52, July 2009.

[15] Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, 9(2):75 – 79, December 2007. ISSN 1931-0145. doi: 10.1145/1345448.1345465. URL https://doi.org/10.1145/1345448.1345465.

[16] Yoshua Bengio and Olivier Delalleau. Justifying and generalizing contrastive divergence. *Neural Comput.*, 21(6):1601 – 1621, June 2009. ISSN 0899-7667. doi: 10.1162/neco.2008.11-07-647. URL https://doi.org/10.1162/neco.2008.11-07-647.

[17] Russa Biswas, Rima Türker, F. B. Moghaddam, Maria Koutraki, and H. Sack. Wikipedia infobox type prediction using embeddings. In *DL4KGS@ESWC*, 2018.

[18] O. Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 32 Database issue:D267–70, 2004.

[19] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of ACM SIGMOD*, pages 1247–1250. ACM, 2008.

[20] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of NIPS*, pages 2787–2795, 2013.

[21] Francois Caron. Bayesian nonparametric models for bipartite graphs. *Advances in Neural Information Processing Systems*, 25:2051–2059, 2012.

[22] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70, Dec 2004.

[23] Michael Cochez, Petar Ristoski, Simone Paolo Ponzetto, and Heiko Paulheim. Biased graph walks for rdf graph embeddings. In *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*, WIMS '17. Association for Computing Machinery, 2017.

[24] Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. Completeness management for RDF data sources. *TWEB*, 12(3):18:1–18:53, 2018.

[25] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. *CoRR*, abs/1707.01476, 2017. URL http://arxiv.org/abs/1707.01476.

[26] J. Chitra Devi and E. Poovammal. An analysis of overlapping community detection algorithms in social networks. *Procedia Computer Science*, 89:349–358, 2016.

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[28] Joseph A. DiMasi, Henry G. Grabowski, and Ronald W. Hansen. Innovation in the pharmaceutical industry: New estimates of R&D costs. *Journal of Health Economics*, 47(C):20–33, 2016. doi: 10.1016/j.jhealeco.2016.0. URL https://ideas.repec.org/a/eee/jhecon/v47y2016icp20-33.html.

[29] Lucas Drumond, Steffen Rendle, and Lars Schmidt-Thieme. Predicting rdf triples in incomplete knowledge bases with tensor factorization. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 326 – 331. Association for Computing Machinery, 2012.

[30] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

[31] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. In Luis Alvarez, Marta Mejail, Luis Gomez, and Julio Jacobo, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 14–36, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33275-3.

[32] Asja Fischer and Christian Igel. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39, 2014.

[33] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3): 75–174, 2010.

[34] Philippe Fournier-Viger, Cheng-Wei Wu, Souleymane Zida, and Vincent S. Tseng. Fhm: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In *Foundations of Intelligent Systems*, pages 83–92. Springer, 2014.

[35] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Tai Dinh, and Hoai Bac Le. Mining correlated high-utility itemsets using the bond measure. In *Hybrid Artificial Intelligent Systems*, pages 53–65. Springer, 2016.

[36] Thomas Franz, Antje Schultz, Sergej Sizov, and Steffen Staab. Triplerank: Ranking semantic web data by tensor decomposition. In *The Semantic Web - ISWC 2009*, pages 213–228. Springer Berlin Heidelberg, 2009.

[37] Paul A Gagniuc. *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[38] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of WWW*, pages 413–422. ACM, 2013.

[39] Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. Bine: Bipartite network embedding. In *The 41st International ACM SIGIR Conference on Research Development in Information Retrieval*, SIGIR'18, pages 715 – 724, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356572. doi: 10.1145/3209978.3209987. URL https://doi.org/10.1145/3209978.3209987.

[40] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *CoRR*, abs/1705.02801, 2017. URL http://arxiv.org/abs/1705.02801.

[41] Daniel Graupe. *Principles of artificial neural networks*, volume 7. World Scientific, 2013.

[42] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'16, pages 855 – 864, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939754. URL https://doi.org/10.1145/2939672.2939754.

[43] Ramanathan Guha and Dan Brickley. RDF schema 1.1. W3C recommendation, W3C, February 2014. https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[44] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 1025 – 1035, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[45] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

[46] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. ISSN 0036-8075. doi: 10.1126/science.1127647. URL https://science.sciencemag.org/content/313/5786/504.

[47] Binbin Hu, Yuan Fang, and Chuan Shi. Adversarial learning on heterogeneous information networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD'19, pages 120 – 129, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330970. URL https://doi.org/10.1145/3292500.3330970.

[48] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[49] Qiu Ji, Zhiqiang Gao, and Zhisheng Huang. Reasoning with noisy semantic data. In *The Semanic Web: Research and Applications*, pages 497–502. Springer Berlin Heidelberg, 2011.

[50] Mayank Kejriwal and Pedro A. Szekely. Supervised typing of big graphs using semantic embeddings. *CoRR*, abs/1703.07805, 2017.

[51] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[52] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[54] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, pages 536 – 543. Association for Computing Machinery, 2008.

[55] Claude Lemaréchal. Cauchy and the gradient method, 2012.

[56] Chong Li, Kunyang Jia, Dan Shen, C.J. Richard Shi, and Hongxia Yang. Hierarchical representation learning for bipartite graphs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2873–2879. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/398. URL https://doi.org/10.24963/ijcai.2019/398.

[57] Xin Li and Hsinchun Chen. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems*, 54(2):880 – 890, 2013. ISSN 0167-9236. doi: https://doi.org/10.1016/j.dss.2012.09.019. URL http://www.sciencedirect.com/science/article/pii/S0167923612002540.

[58] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM'03, pages 556 – 559, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137230. doi: 10.1145/956863.956972. URL https://doi.org/10.1145/956863.956972.

[59] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of AAAI*, pages 2181–2187. AAAI Press, 2015.

[60] Feng Liu, Bingquan Liu, Chengjie Sun, Ming Liu, and Xiaolong Wang. Deep learning approaches for link prediction in social network services. In Minho Lee, Akira Hirose, Zeng-Guang Hou, and Rhee Man Kil, editors, *Neural Information Processing*, pages 425–432, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-42042-9.

[61] Hanxiao Liu, Yuexin Wu, and Yiming Yang. Analogical inference for multi-relational embeddings. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2168–2178. JMLR.org.

[62] Xi Liu, Peng Lu, Xiaohan Zuo, Jianxin Chen, Hongjun Yang, Yiping Yang, and Yibo Gao. Prediction of network drug target based on improved model of bipartite graph valuation. *China journal of Chinese materia medica*, 37 2: 125–9, 2012.

[63] Xin Liu, Tsuyoshi Murata, Kyoung-Sook Kim, Chatchawan Kotarasu, and Chenyi Zhuang. A general view for network embedding as matrix factorization. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM'19, pages 375 – 383, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405. doi: 10.1145/3289600.3291029. URL https://doi.org/10.1145/3289600.3291029.

[64] Ying Liu, Wei-keng Liao, and Alok Choudhary. A fast high utility itemsets mining algorithm. In *Proceedings of UBDM '05*, pages 90–99. ACM, 2005.

[65] Ying Liu, Wei-keng Liao, and Alok Choudhary. A two-phase algorithm for fast discovery of high utility itemsets. In *Proceedings of PAKDD'05*, pages 689–695. Springer, 2005.

[66] Gaurav Maheshwari, Priyansh Trivedi, Denis Lukovnikov, Nilesh Chakraborty, Asja Fischer, and Jens Lehmann. Learning to rank query graphs for complex question answering over knowledge graphs. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtěch Svátek, Isabel Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web – ISWC 2019*, pages 487–504, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30793-6.

[67] Bassem Makni and James Hendler. Deep learning for noise-tolerant rdfs reasoning. *Semantic Web*, 10(5):823–862, 2019.

[68] Awais Manzoor Bajwa, Diego Collarana, and Maria-Esther Vidal. Interaction network analysis using semantic similarity based on translation embeddings. In Maribel Acosta, Philippe Cudré-Mauroux, Maria Maleshkova, Tassilo Pellegrini, Harald Sack, and York Sure-Vetter, editors, *Semantic Systems. The Power of AI and Knowledge Graphs*, pages 249–255, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33220-4.

[69] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[70] André Melo, Heiko Paulheim, and Johanna Völker. Type prediction in rdf knowledge bases using hierarchical multilabel classification. In *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, pages 1–10, 2016.

[71] R. Merris. *Graph Theory*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011. ISBN 9781118031292. URL https://books.google.de/books?id=dsf6wcWYgrgC.

[72] Nandana Mihindukulasooriya and Mariano Rico. Type prediction of rdf knowledge graphs using binary classifiers with structural data. In *Current Trends in Web Engineering*, pages 279–287. Springer International Publishing, 2018.

[73] Nandana Mihindukulasooriya, Mohammad Rifat Ahmmad Rashid, Giuseppe Rizzo, Raúl García-Castro, Oscar Corcho, and Marco Torchiano. Rdf shape induction using knowledge base profiling. In *Proceedings of ACM SAC*, SAC'18, pages 1952 – 1959, New York, NY, USA, 2018. Association for Computing Machinery.

[74] Thomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[75] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pages 3111–3119. Curran Associates Inc.

[76] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[77] Changsung Moon, Paul Jones, and Nagiza F. Samatova. Learning entity type embeddings for knowledge graph completion. CIKM '17, pages 2215 – 2218. Association for Computing Machinery, 2017. ISBN 9781450349185.

[78] M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001.

[79] M. E. J. Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2), Jul 2001. ISSN 1095-3787. doi: 10.1103/physreve.64.025102. URL http://dx.doi.org/10.1103/PhysRevE.64.025102.

[80] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004.

[81] Tu Dinh Nguyen, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Latent patient profile modelling and applications with mixed-variate restricted boltzmann machine. In *Advances in Knowledge Discovery and Data Mining*, pages 123–135. Springer Berlin Heidelberg, 2013.

[82] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 1955–1961. AAAI Press.

[83] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 809–816, USA, 2011. Omnipress.

[84] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: Scalable machine learning for linked data. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 271 – 280. Association for Computing Machinery, 2012.

[85] Edward R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE TKDE*, 15(1):57–69, January 2003.

[86] Sergio Oramas, Vito Claudio Ostuni, Tommaso Di Noia, Xavier Serra, and Eugenio Di Sciascio. Sound and music recommendation with knowledge graphs. *ACM Trans. Intell. Syst. Technol.*, 8(2):21:1–21:21, October 2016.

[87] Guillermo Palma, Maria-Esther Vidal, and Louiqa Raschid. Drug-target interaction prediction using semantic similarity and edge partitioning. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble, editors, *The Semantic Web – ISWC 2014*, pages 131–146, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11964-9.

[88] Alexandre Passant. dbrec — music recommendations using dbpedia. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *The Semantic Web – ISWC 2010*, pages 209–224, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-17749-1.

[89] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8:489–508, 2017.

[90] Heiko Paulheim and Christian Bizer. Type inference on noisy rdf data. In *The Semantic Web – ISWC 2013*, pages 510–525. Springer Berlin Heidelberg, 2013.

[91] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[92] Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. Completeness-aware rule learning from knowledge graphs. In *Proceedings of ISWC*, pages 507–525, 2017.

[93] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, October .

[94] Bianca Pereira. Entity linking with multiple knowledge bases: An ontology modularization approach. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble, editors, *The Semantic Web – ISWC 2014*, pages 513–520, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11915-1.

[95] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'14, pages 701 – 710, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329569. doi: 10.1145/2623330.2623732. URL https://doi.org/10.1145/2623330.2623732.

[96] Axel Polleres, Aidan Hogan, Andreas Harth, and Stefan Decker. Can we ever catch up with the web? *Semantic Web*, 1(1, 2):45–52, 2010.

[97] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In *Proceedings of ISCIS'05*, 2005.

[98] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM'18, pages 459 – 467, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355810. doi: 10.1145/3159652.3159706. URL https://doi.org/10.1145/3159652.3159706.

[99] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76, Sep 2007.

[100] Marc' Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 1185 – 1192, Red Hook, NY, USA, 2007. Curran Associates Inc. ISBN 9781605603520.

[101] Wala Rebhi, Nesrine Ben Yahia, and Narjs Bellamine Ben Saoud. Hybrid modeling approach for contextualized community detection in multilayer social network. *Procedia Comput. Sci.*, 112(C):673–682, September 2017.

[102] Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *The Semantic Web - ISWC 2016 : 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, volume 9981, pages 498–514. Springer International Publishing, 2016.

[103] Petar Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, and Heiko Paulheim. Rdf2vec: Rdf graph embeddings and their applications. *Semantic Web*, 10(4):721–752, 2019.

[104] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML'07, pages 791 – 798. Association for Computing Machinery, 2007.

[105] Carlos N Silla and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

[106] Robert F Simmons. *Synthetic language behavior*. System Development Corporation, 1963.

[107] Radina Sofronova, M. Alam, and H. Sack. Entity typing based on rdf2vec using supervised and unsupervised methods. 2020.

[108] Wei Song, Yu Liu, and Jinhong Li. Bahui: Fast and memory efficient mining of high utility itemsets based on bitmap. *Int. J. Data Warehous. Min.*, 10(1): 1–15, January 2014.

[109] Karl Steinbuch. Die lernmatrix. *Kybernetik*, 1(1):36–45, 1961.

[110] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge. In *16th International Conference on the World Wide Web*, pages 697–706, 2007.

[111] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW'15, pages 1067 – 1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee. ISBN 9781450334693. doi: 10.1145/2736277.2741093. URL https://doi.org/10.1145/2736277.2741093.

[112] Peihao Tong, Qifan Zhang, and Junjie Yao. Leveraging domain context for question answering over knowledge graph. *Data Science and Engineering*, 4 (4):323–335, 2019.

[113] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1174. URL https://www.aclweb.org/anthology/D15-1174.

[114] Minh Duc Tran, Claudia d'Amato, Binh Thanh Nguyen, and Andrea G. B. Tettamanzi. An evolutionary algorithm for discovering multi-relational association rules in the semantic web. In *Proceedings of GECCO*, GECCO'17, pages 513 – 520, New York, NY, USA, 2017. ACM.

[115] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of ICML*, pages 2071–2080. JMLR.org, 2016.

[116] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, Nilesh Agrawal, and Partha Talukdar. Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 3009–3016. AAAI Press, 2020.

[117] Denny Vrandečić and Markus Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, September 2014.

[118] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Relational deep learning: A deep latent variable model for link prediction. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 2688 – 2694. AAAI Press, 2017.

[119] Huiqing Wang, Jingjing Wang, Chunlin Dong, Yuanyuan Lian, Dan Liu, and Zhiliang Yan. A novel approach for drug-target interactions prediction based on multimodal deep autoencoder. *Frontiers in Pharmacology*, 10:1592, 2020. ISSN 1663-9812. doi: 10.3389/fphar.2019.01592. URL https://www.frontiersin.org/article/10.3389/fphar.2019.01592.

[120] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding, 2017. URL https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14589.

[121] Yuhao Wang and Jianyang Zeng. Predicting drug-target interactions using restricted boltzmann machines. In *Bioinform.*, 2013.

[122] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of AAAI*, pages 1112–1119, 2014.

[123] David Wood, Richard Cyganiak, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[124] Yoshihiro Yamanishi, Michihiro Araki, Alex Gutteridge, Wataru Honda, and Minoru Kanehisa. Prediction of drugâtarget interaction networks from the integration of chemical and genomic spaces. *Bioinformatics*, 24(13):i232–i240, 07 2008. ISSN 1367-4803. doi: 10.1093/bioinformatics/btn162. URL https://doi.org/10.1093/bioinformatics/btn162.

[125] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of ICLR*, 2015.

[126] Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*, pages 1170–1175. IEEE Computer Society.

[127] Jaewon Yang and Jure Leskovec. Overlapping communities explain core-periphery organization of networks. *Proceedings of the IEEE*, 102:1892–1902, 2014.

[128] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 555–564. ACM.

[129] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *CoRR*, abs/1801.05852, 2018. URL http://arxiv.org/abs/1801.05852.

[130] Peng Zhang, Xiang Wang, Futian Wang, An Zeng, and Jinghua Xiao. Measuring the robustness of link prediction algorithms under noisy environment. *Scientific reports*, 6(1):1–7, 2016.

[131] Shuai Zhang, Yi Tay, L. Yao, and Q. Liu. Quaternion knowledge graph embeddings. In *NeurIPS*, 2019.

[132] Yao Zhang, Yun Xiong, Xiangnan Kong, and Yangyong Zhu. Learning node embeddings in interaction graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM'17, pages 397 – 406, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349185. doi: 10.1145/3132847.3132918. URL https://doi.org/10.1145/3132847.3132918.

LIST OF FIGURES

LIST OF TABLES

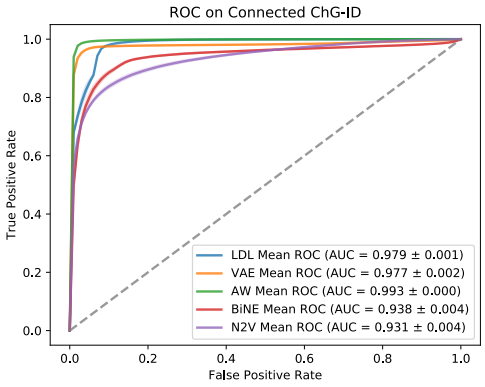
ACRONYMS

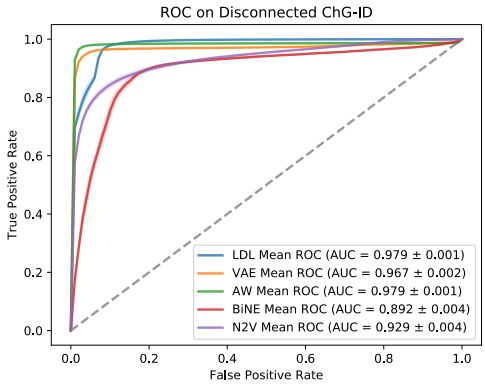
KG	Knowledge Graph
FHM	Faster High-Utility Itemset Mining
AGM	Community-Affiliation Graph Model
CRP	Community-based Relation Prediction
PCA	Principal Component Analysis
LDL	Link Distribution Learning
FCHM	Fast Correlated High-Utility Itemset Miner
RBM	Restricted Boltzmann Machine
CD	Contrastive Divergence
CWA	Closed World Assumption
OWA	Open World Assumption
IGE	Interaction Graph Embedding
DBN	Deep Belief Network
VAE	Variational Graph Auto-Encoder
GAN	Generative Adversarial Network
AUC	Area-under-Curve
PR	Precision-Recall-Curve
TPR	True Positive Rate
Ridle	Relation-Instance Distribution Learning

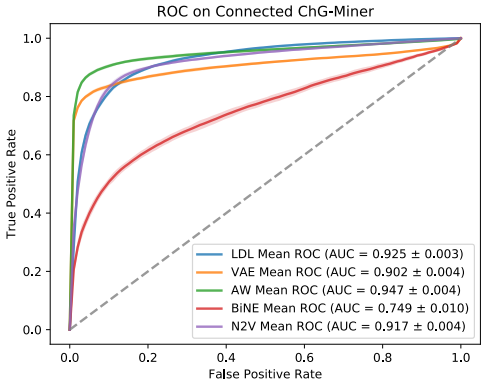
A

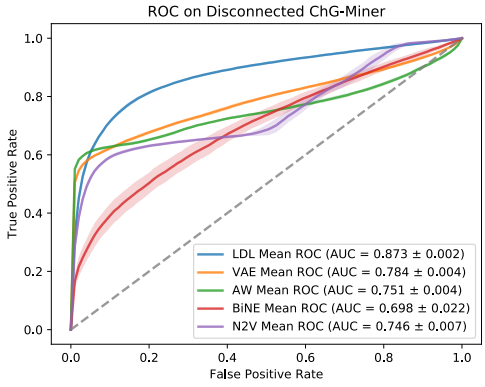
APPENDIX: ADDENDUM

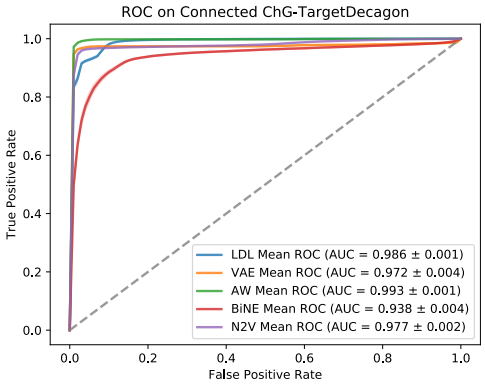
A.1 ROC ANALYSIS OF CONSIDERED BIPARTITE NETWORKS

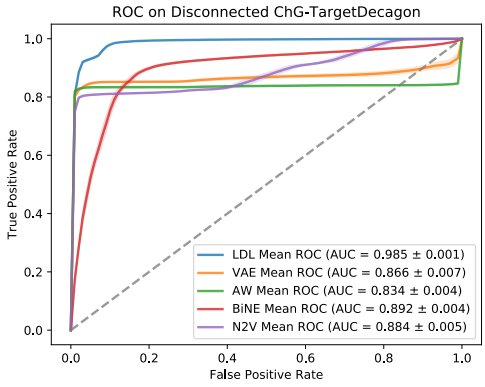


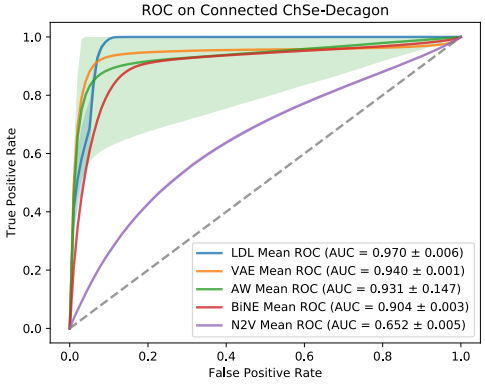


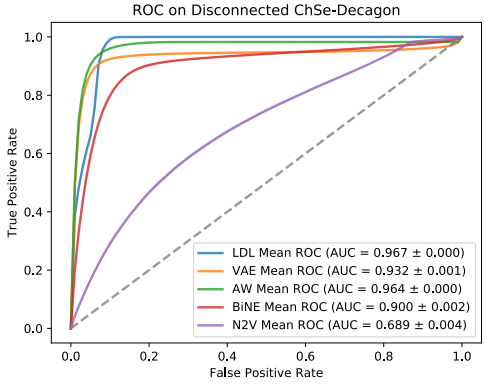












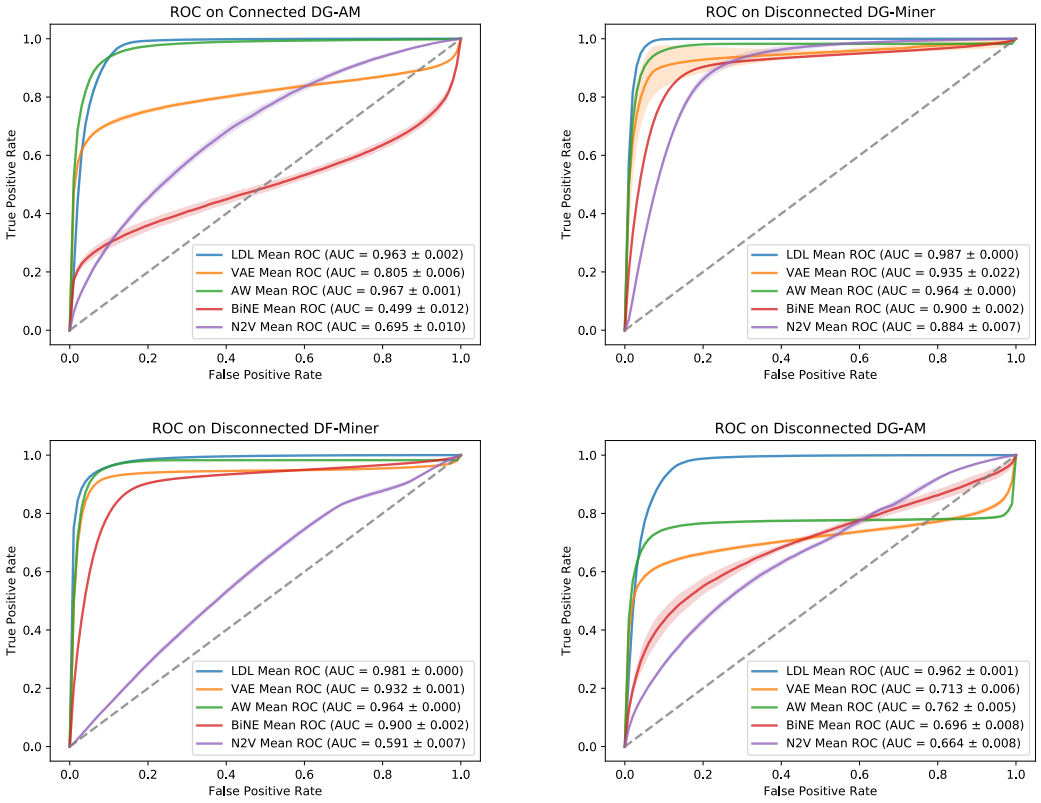
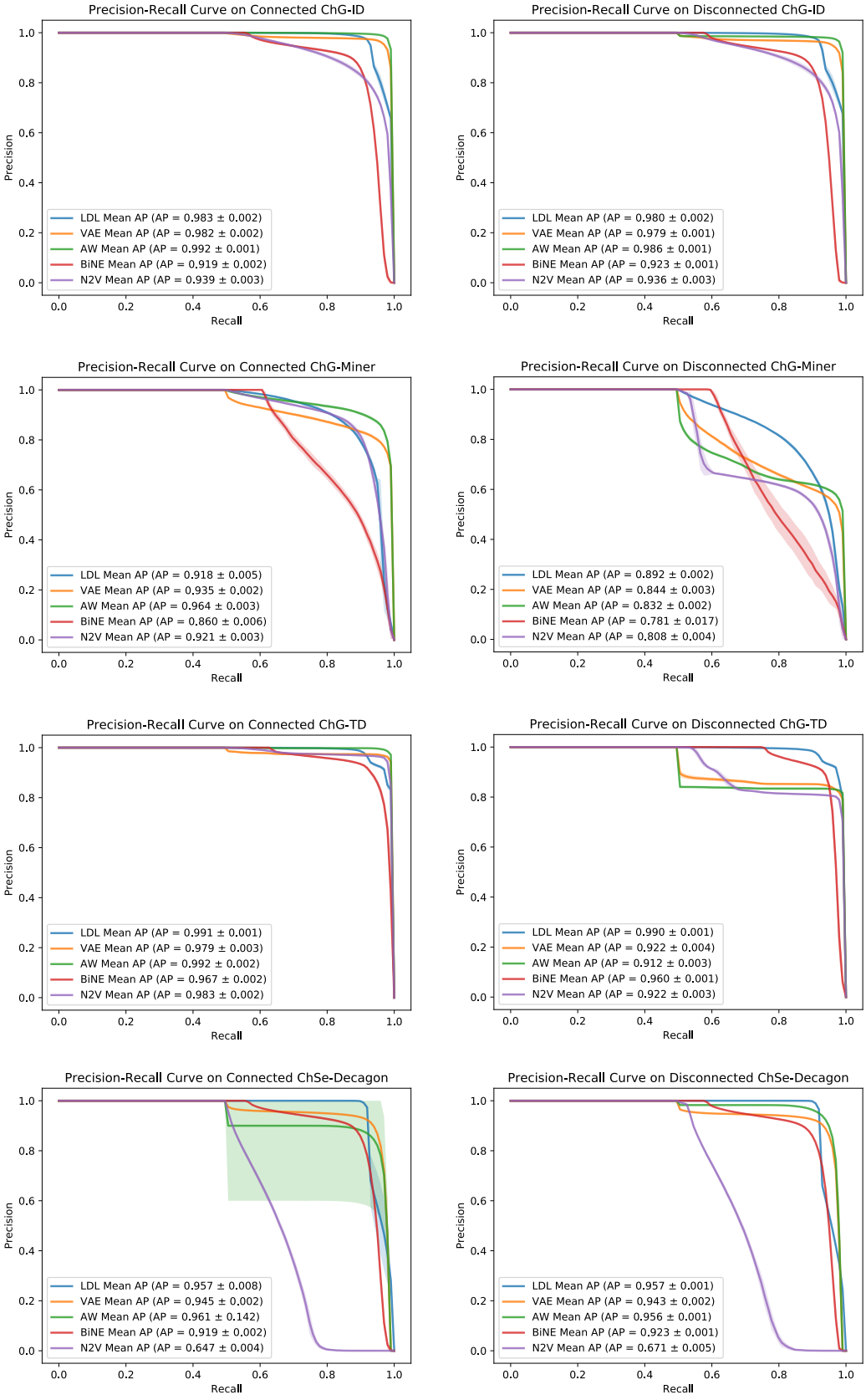


Figure 28: Average ROC for all networks, both for connected and disconnected Networks when removing 30% links.

A.2 PRECISION-RECALL ANALYSIS OF CONSIDERED BIPARTITE NETWORKS



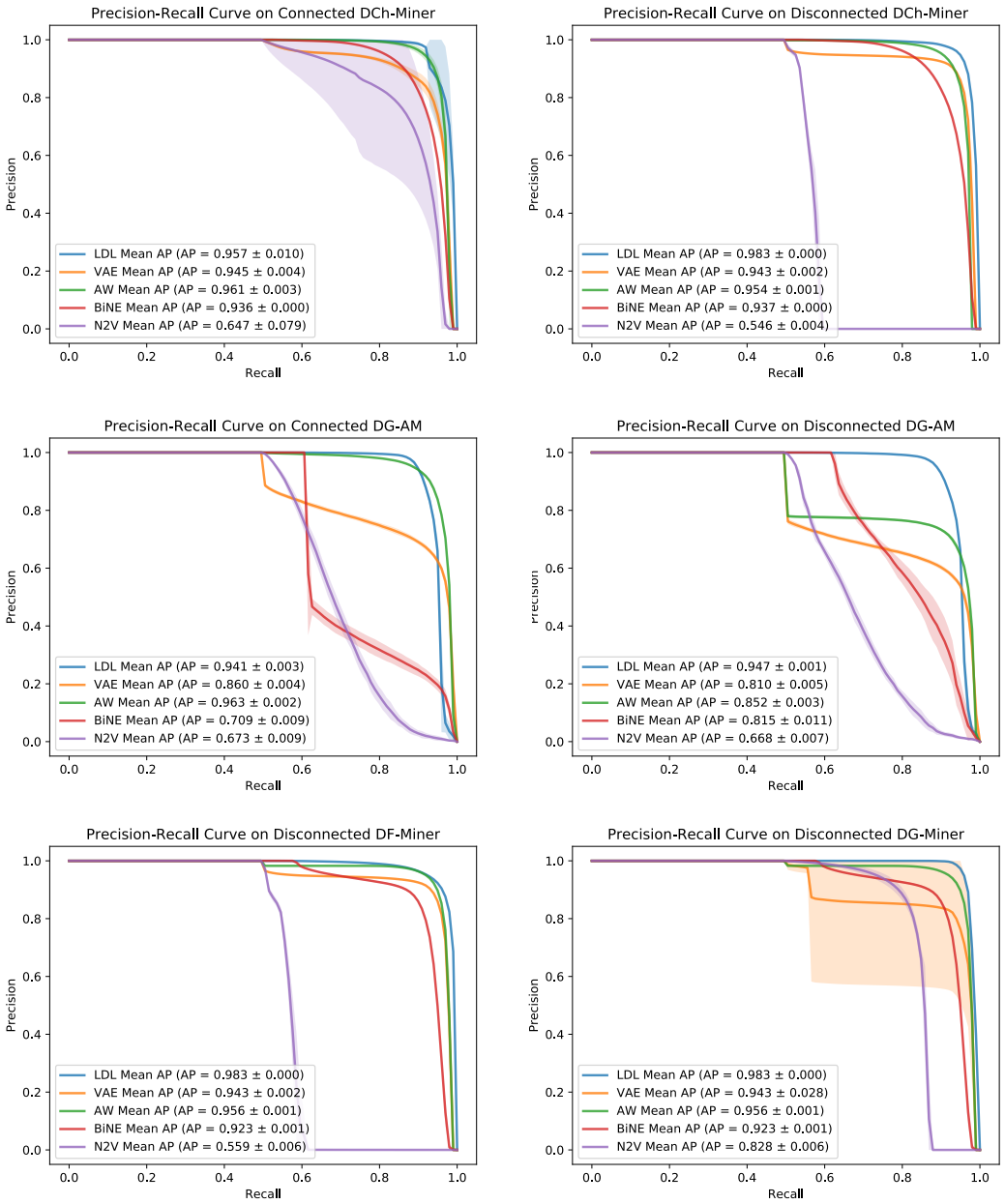


Figure 29: Average Precision-Recall Curve for all networks, both for connected and disconnected Networks when removing 30% links.

A.3 ANALYSIS OF USED DISTRIBUTION FUNCTION

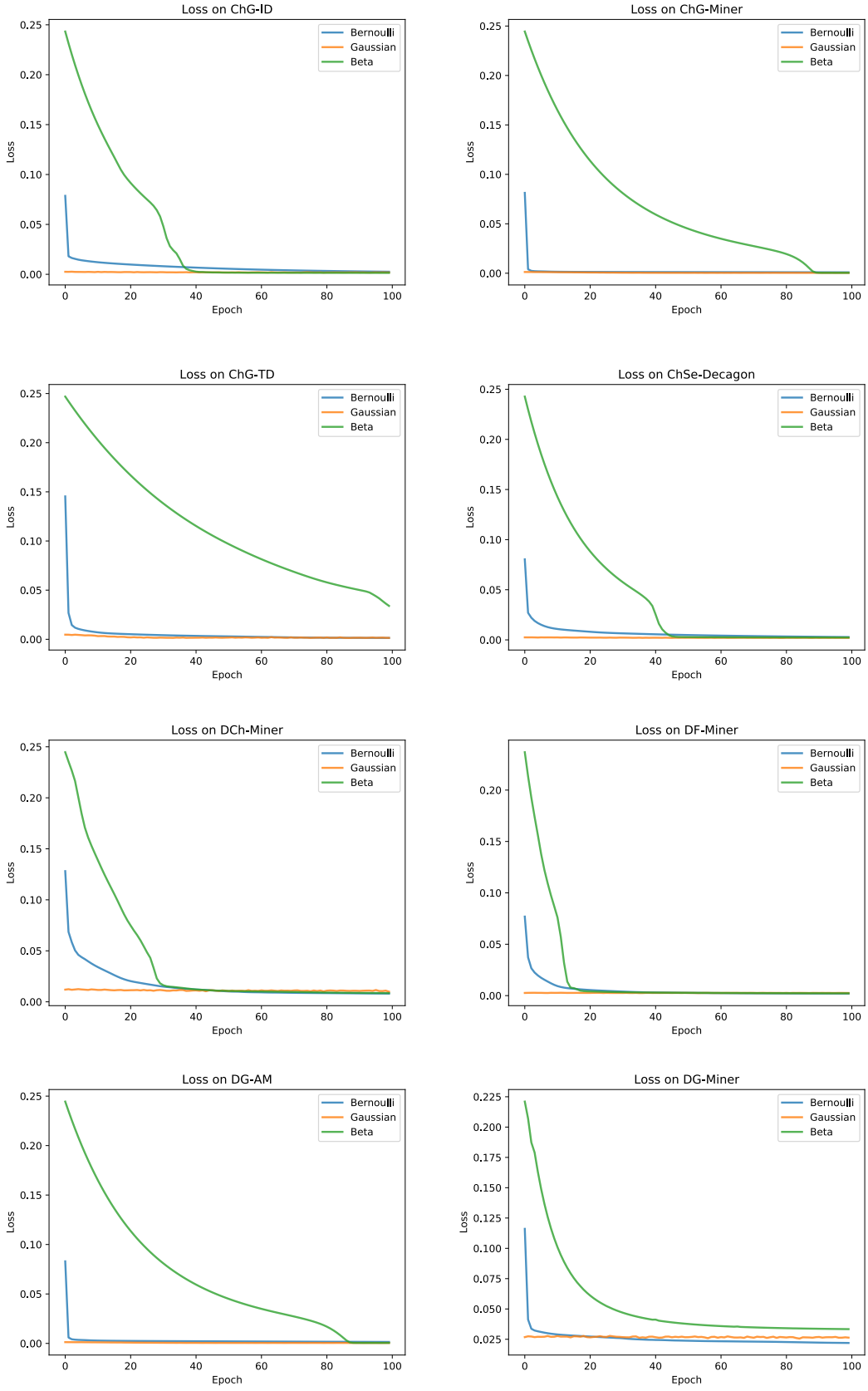


Figure 30: Progression of loss using different distribution function in the hidden layer.