UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

WeST People and Knowledge Networks
Institute for Web Science and Technologies

Representing Evolving Knowledge Graphs through Incremental Embeddings

Masterarbeit

zur Erlangung des Grades eines Master of Science (M.Sc.)
im Studiengang Wirtschaftsinformatik

vorgelegt von

Rashid Lafraie

Erstgutachter:	Prof. Dr. Steffen Staab Institute for Web Science and Technologies
Zweitgutachter:	Dr. Zeyd Boukhers Institute for Web Science and Technologies

Koblenz, im September 2020

Erklärung

Hiermit bestätige ich, dass die vorliegende Arbeit von mir selbstständig verfasst wurde und ich keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe und die Arbeit von mir vorher nicht in einem anderen Prüfungsverfahren eingereicht wurde. Die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium (CD-Rom).

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.		
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.		
Der Text dieser Arbeit ist unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar.		
Der Quellcode ist unter der MIT Lizenz (MIT) verfügbar.		
Die erhobenen Daten sind unter einer Creative Commons Lizenz (CC BY-SA 4.0) verfügbar.		

....

(Ort, Datum) (Unterschrift)

Anmerkung

If you would like us to contact you for the graduation ceremony,

please provide your personal E-mail address: . . . .

If you would like us to send you an invite to join the WeST Alumni

and Members group on LinkedIn, please provide your LinkedIn ID : . . . .

Zusammenfassung

Das Forschungsgebiet von Knowledge Graph Embedding besteht aus Verfahren, die numerische Repräsentationen für die Entitäten und Relationen eines Knowledge Graphen lernen. Typischerweise verändern sich Knowledge Graphen wie Wikidata oder YAGO mit der Zeit. Die Mehrheit solcher Verfahren ist jedoch statisch, d.h. für Knowledge Graphen ausgelegt, die als unveränderlich angenommen werden. Wird ein Graph modifiziert, so müssen gelernte Repräsentationen von Grund auf neu gelernt werden. Die Forschungsrichtung von Incremental Knowledge Graph Embedding behandelt dieses Problem. Bis dato enthält sie drei Modelle, die in der Lage sind, bestehende Repräsentationen nach einer Modifikation eines Graphen beizubehalten und inkrementell anzupassen. Alle Modelle wurden jedoch in unterschiedlichen Szenarien evaluiert, sodass ihre Ergebnisse nicht vergleichbar sind. Des Weiteren berücksichtigen diese untersuchten Evaluationsszenarien nicht die Besonderheiten von inkrementell-gelernten Repräsentationen. Infolgedessen existiert in der Literatur kein angemessener Maßstab für die Evaluation inkrementeller Modelle. In dieser Arbeit wird ein solcher Evaluationsrahmen entwickelt. Zunächst werden die bestehenden Forschungsarbeiten auf ihre inkrementellen Konzepte, sowie die Stärken und Schwächen ihrer Evaluationsansätze hin untersucht. Basierend auf dem Analyseergebnis werden Anforderungen für eine angemessene Bewertung inkrementeller Verfahren erhoben, die wir in der Konstruktion unseres Evaluationsrahmens berücksichtigen. Dieser ermöglicht es, inkrementelle Modelle im Laufe der Evolution eines Knowledge Graphen zu testen und miteinander zu vergleichen. Ebenso können aber auch statische Repräsentationsmethoden für einen Vergleich herangezogen werden. Um unser Evaluationsframework auf authentischen Daten zu stützen, konzipieren wir den Datensatz WikidataEvolve. Dieser ist aus Wikidata extrahiert und stellt einen sich entwickelnden Knowledge Graphen dar. Wir orientieren seine Konstruktion eng an das konzipierte Evaluationsframework. Insofern, stellt er den ersten öffentlichen Benchmark für das Forschungsgebiet Incremental Knowledge Graph Embedding dar, der für eine angemessene Evaluation inkrementeller Verfahren verwendet werden kann. Darüber hinaus implementieren wir in dieser Arbeit das inkrementelle Modell von PuTransE und entwickeln auf Basis seines inkrementellen Konzeptes die neuen Modelle PuTransH und PuTransD. Letztlich wird das entwickelte Evaluationsframework unter Verwendung von WikidataEvolve in einem Experiment angewandt, um die implementierte Methode PuTransE zu evaluieren und mit der statischen Methode TransE zu vergleichen. Die Ergebnisse verdeutlichen, dass die statische Methode im Verlauf des Graphen deutlich besser abschneidet.

Abstract

The research area of knowledge graph embedding describes methods that learn numerical representations for a knowledge graph's entities and relations. Knowledge graphs like Wikidata or YAGO typically evolve over time. However, most of such methods are mainly designed for knowledge graphs whose set of facts is assumed to be unchanging. If a graph is manipulated, these static methods have to reject their learned representations and learn them from scratch. The research area of incremental knowledge graph embedding tackles this issue. It consists of methods that can keep their representations after a graph modification and adapt them incrementally. Up to now, only three papers are describing such techniques. As they have been tested in different scenarios, their evaluation results are not comparable. Further, these scenarios do not consider the characteristics of incremental methods. Thus, there is no benchmark in the literature to assess and compare incremental models throughout evolving knowledge graphs adequately. In this thesis, we develop a comprehensive evaluation framework. First, we analyze these three related works concerning their incremental concepts and the benefits and drawbacks of their evaluation scenarios. Based on the analysis results, we define requirements for the appropriate evaluation of incremental methods, which we incorporate into the construction of our evaluation framework. It enables to evaluate and compare incremental models in the context of an evolving knowledge graph. Besides, it is suitable for applying static procedures so that these can be included in the comparison. In order to support our evaluation framework with factual data, we compile the dataset WikidataEvolve. This benchmark is extracted from Wikidata and represents an evolving knowledge graph. Further, we implement the incremental model of PuTransE and, based on its incremental concept, create the new models PuTransH and PuTransD. Finally, the developed evaluation framework is applied in an experiment using our WikidataEvolve benchmark to evaluate the implemented method PuTransE and compare it with the static method TransE. The results show that the static method performs significantly better throughout the evolution of the knowledge graph.

Contents

1. Introduction

A knowledge graph is a set of triples, that each are composed of two entities h, t sourced from a set of entities E, and a semantic relation running between them, sourced from a set of relation types R, to form a fact (h, r, t). Knowledge graphs are widely used to store knowledge of specific domains (Paulheim, 2017). Prominent examples are the Google knowledge graph[1], YAGO and Wikidata (Suchanek et al., 2007; Vrandecic and Krötzsch, 2014). YAGO, for instance, stores facts about people, locations, media and other things of the real world (Suchanek et al., 2007).

In the past, these graphs served as a base for intelligent applications like social network analysis, recommendation systems and question answering (Sun et al., 2018; Wang et al., 2019). Basically, such applications reason over stored information and often involve numerical algorithms (Wang et al., 2017). Due to the vast amount of entities and relations knowledge graphs typically contain and the symbolic nature of their triples, underlying computations become difficult.

In order to make inferences more efficient, the domain of knowledge graph embedding describes techniques to numerically encode the information contained in knowledge graphs. Based on the facts of a graph, related models form numerical representations, called embeddings, for all its entities and relations and learn a function assigning an energy score to an arbitrary triple (Bordes et al., 2011). This score serves as a measure of truth and expresses how plausible a triple is considered by a model (Wang et al., 2017).

Naturally, knowledge graphs change over time as in the domains they are applied to, new knowledge is acquired, or existing insights are rejected (Wu et al., 2019). Real-world examples illustrate this. Wikidata, for instance, recorded approximately 1.2 billion updates since its creation[2]. Referred to as evolving knowledge graphs, their set of fact changes with insertions or deletions of triples.

However, most of the existing knowledge graph embedding techniques are designed for static knowledge graphs, whose set of facts is assumed to remain unchanged (Wu et al., 2019). For such models all facts, entities and relations of the graph must be known at training time. In case the underlying knowledge graph is modified afterwards, these models cannot adapt their representations. Referred to as static knowledge graph embedding models, they reject their embeddings and learn the modified graph from scratch (Jia et al., 2016; Wu et al., 2019).

This issue is addressed in the research direction of incremental knowledge graph embedding. It manifests embedding models that can retain their embeddings and incrementally optimize them after a graph was updated without having to learn its whole set of facts from scratch (Wu et al., 2019). To the best of our knowledge, in the literature only three papers have been published that describe incremental techniques. These include the works of Jia et al. (2016), Tay et al. (2017) and Wu

[1] `https://www.blog.google/products/search/introducing-knowledge-graph-things-not/` last retrieved September 20, 2020.

[2] `https://www.wikidata.org/wiki/Wikidata:Statistics/de` last retrieved June 19, 2020.

et al. (2019).

As the authors reported, their models achieved promising results in the context of evolving knowledge graphs. However, these results are not comparable, as all models have been evaluated in different test scenarios. Further, their evaluation strategies fail to capture the specifics of incremental models. Since incremental embedding models maintain and update their embeddings, it is particularly relevant for their application in real-world scenarios to what extent they can learn facts that have been inserted into the knowledge graph and unlearning those that have been deleted. All in all, there exists no uniform procedure nor an appropriate benchmark for the evaluation and comparison of incremental models.

In this thesis, we have developed a comprehensive evaluation framework tailored to assess incremental knowledge graph embedding methods. First, it allows to evaluate incremental models in the context of an evolving knowledge graph and to compare them with static knowledge graph embedding models. Second, it defines a test scheme specifically designed to consult incremental models' strengths and weaknesses in learning and unlearning facts. To support our evaluation framework with data, we have compiled a dataset called WikidataEvolve. Extracted from the real-world knowledge graph Wikidata, it represents an authentic evolving knowledge graph and therefore contributes as a benchmark to the domain of incremental knowledge graph embedding. Furthermore, we implemented the incremental model of PuTransE. Based on its incremental strategy, the novel incremental models of PuTransH and PuTransD has been created. To apply our evaluation framework, we used the WikidataEvolve benchmark to evaluate the incremental embedding method of PuTransE and compare it with the static embedding method of TransE.

The remainder of this thesis is structured as follows. Section 2 outlines the theoretical background of this work. Here, we introduce the domain of of knowledge graphs and cover fundamentals of knowledge graph embeddings. Next, a range of popular static knowledge graph embedding methods are reviewed, as their concepts serve as a base for the majority of incremental embedding methods. Section 3 introduces the domain of incremental knowledge graph embedding which is the thematic core of this work. First, we classify incremental knowledge graph embedding models. Subsequently, we review the incremental concepts of existing techniques and examine the approaches of the authors to evaluate them. In section 4 we describe our implementation of the incremental methods PuTransE and the newly developed methods PuTransH and PuTransD. In section 5 the elaborated evaluation framework for the assessment of incremental embedding models is presented. Subsequently, in section 6, we describe the construction of WikidataEvolve. It contributes to the domain of incremental knowledge graph embedding by manifesting the first benchmark for the appropriate evaluation of incremental models. Here, it is explained how we extracted and transformed a stream of triple operations from Wikidata to create the benchmark. In section 7 we reflect on the combination of our evaluation framework and the WikidataEvolve benchmark and discuss the benefits it provides for evaluating incremental models but also explain its limitations.

Finally, section 8 describes the experiment we conducted in which we applied our framework and used the WikidataEvolve benchmark to evaluate the incremental method of PuTransE and the static embedding method of TransE in context of an evolving knowledge graph.

2. Background

In this section, we convey the theoretical foundations of this thesis. By introducing the domain of knowledge graphs, we provide a definition in the first place, followed a basic notation of its elements used in the course of the work and explain their evolving nature. Afterwards we outline the research direction of knowledge graph embedding. Here, we draw on the basic design of embedding techniques and describe benchmark evaluation tasks and datasets. Lastly, we review a range of popular knowledge graph embedding models.

2.1. Knowledge Graphs

In the literature, there exists no clear definition of the term *"knowledge graph"*. Nickel et al. (2016) define a knowledge graph as a collection of facts modeled by entities as nodes and their relations as edges. Accordingly, Kroetsch and Weikum (2016) describe it as a network, which consists of entities, their related properties and semantic types on the one hand and relations between entities on the other hand. Extending these definitions, most of the existing papers characterize a knowledge graph as a multi-relational graph, meaning that semantic relations between entities can be of different types (Bordes et al., 2013; Wang et al., 2017).

We view a knowledge graph (KG) as a set of triples T, each consisting of a head entity h, a tail entity t, both sourced from the set of KG entities E, and a relation r, sourced from the set of KG relations R. This definition is formalized by:

$$T = \{(h, r, t)\}\text{, whereby } h, t \in E \text{ and } r \in R.$$

Depicting a multi-relational graph, it is used to store facts of a specific knowledge domain, where nodes represent entities and edges represent semantic relations between them (Paulheim, 2017; Wang et al., 2017). Included entities may represent all things from real-world objects to abstract concepts, which are part of a specific knowledge domain (Ehrlinger and Wöß, 2016; Paulheim, 2017). A relation r, in turn, can be of different types and runs from the head entity h to the tail entity t to manifest a triple (h, r, t), following the subject-predicate-object scheme (Shi and Weninger, 2018).

Figure 1 samples a KG which depicts several book titles, their author and publishers and their semantic relations towards each other. The fact *"Stephen King is the author of Night Shift"*, for example, is encoded as (*"Stephen King"*,*"is author of"*,*"Night Shift"*), where *"Stephen King"* and *"Night Shift"* act as head and tail entities connected by the semantic relation *"is author of"* running from the head to the tail entity.

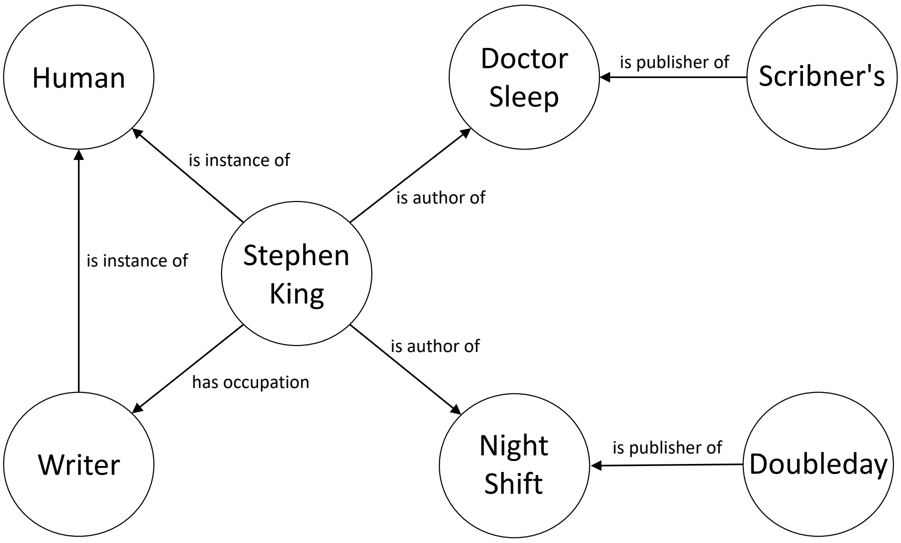


Figure 1: Illustration of a knowledge graph depicting semantic relations between book titles and their author and publishers.

Works like García-Durán et al. (2018) and Lacroix et al. (2020) further focus on temporal KGs, which define a time validity for facts. Here, a triple is extended by a temporal dimension to form a quadruple. The quadruple (*"Barack Obama", "position held", "President of the USA", "2009-2017"*) serves as an example, meaning that the fact (*"Barack Obama", "position held", "President of the USA"*) hold in the time frame *"2009-2017"*. In contrast to their work, we focus on KGs that ignore the temporal validity of a triple. Implicitly, if a triple is in the KG at the time of observation, it is considered valid.

In practice, KGs exhibited promising potentials by serving as information bases for practical applications like question answering (Bordes et al., 2014), social network analysis (Wang et al., 2018), medical diagnostic reasoning systems (Rotmensch et al., 2017) or QA systems (Cui et al., 2017). Through the announcement of the Google KG in 2012, which serves as a base for the company's search engine, KGs gained public attention[3]. Besides, further Instances of KGs have been realized and applied in the industry. These are located at IBM (Devarajan, 2017), Facebook (Noy et al., 2019) and Amazon (Krishnan, 2018), for example.

Amazon set up a KG by mapping information about the products distributed in its online shop. As a result, visitors of the website are offered with improved user experience. First, product searches are simplified. The product graph enables the search engine to understand entered user key words. By supplying the keywords *„bath suit"* for example, products are listed which are semantically related, i.e. yield-

[3] `https://www.blog.google/products/search/introducing-knowledge-graph-things-not/` last retrieved September 20, 2020.

ing in a variety of swimsuits (Krishnan, 2018). Additionally, Amazon uses its KG to identify hidden patterns in the behavior of vendors and customers. The obtained insights can then be used to classify customers to provide improved product recommendations (Krishnan, 2018).

Despite commercial graphs, there exist several publicly accessible KGs which are widely used in academia. Freebase, Wikidata, WordNet and YAGO are information sources that are ranked among the most popular ones. Freebase is a KG that stores relational information about people, locations, media and other things of the real world (Toutanova et al., 2015). The graph was created in 2007 and later acquired by Google (Färber et al., 2018). Its complementary online services ended in 2016, whereas its stored data has been integrated into the Wikidata KG (Tanon et al., 2016). However, the latest data acquired by Freebase is still available yet. Wikidata, in turn, is a graph established in 2012. Besides the type of information represented in the Freebase graph, it also stores the knowledge sources of facts to evaluate their truthfulness (Färber et al., 2018).

In contrast, Wordnet is a lexical KG and models information about the English language (Miller, 1994). It was created in 1985 by the Princeton University, New Jersey, in the U.S., following the objective of realizing an online database for the English language that can be processed by machines (Miller, 1994). Accordingly, it is composed of English nouns, verbs, adjectives, and adverbs interconnected by semantic relation types (Miller, 1994). The YAGO graph sources contained facts from Wikipedia and WordNet to merge them uniformly into a database (Suchanek et al., 2007). Accordingly, it stores relational information about real-world objects, the same alike represented by Freebase and Wikidata. These include people, organizations, locations and books. (Suchanek et al., 2007).

KGs can be constructed in different ways. Some are created manually by human actors. This can be done by employing knowledge engineers who act as human experts in their field of knowledge and curate the KG by adding or removing facts (Hogan et al., 2020). Other KGs such as Wikidata develop their knowledge base through crowd-sourcing (Vrandecic and Krötzsch, 2014). In addition to manual maintenance, KGs can also be created automatically by extracting information from text sources such as books, newspapers and scientific articles using Natural Language Processing techniques. Both options further can be combined into a semi-automatic approach by having extracted facts verified by human actors (Hogan et al., 2020).

Typically KGs evolve over time. In the domains they represent, new knowledge is continuously accumulated, or existing insights are rejected (Wu et al., 2019). In this respect, triples are either inserted or removed from the graph's knowledge base. Real-world scenarios underline such dynamics: Wikidata, for example, stored 5.3 million entities in 2016, whereas in 2019 it counted 11.3 million entities[4]. DBpedia, in turn, daily integrates the update stream of Wikipedia into its knowledge base

[4] `https://wikitech.wikimedia.org/wiki/WMDE/Wikidata/Growth#Wikidata` last retrieved June 28, 2020.

(Wu et al., 2019). Amazon, as an industrial example, updates its KG with respect to its changing product range (Wu et al., 2019).

We refer to a changing graph as evolving KG, while we call an unchanging graph static KG. Defined by a set of facts, which changes along a time stream of insert and delete operations, evolving KGs form the thematic core of this work as they are used in our designed evaluation framework to assess incrementally learned embeddings over time.

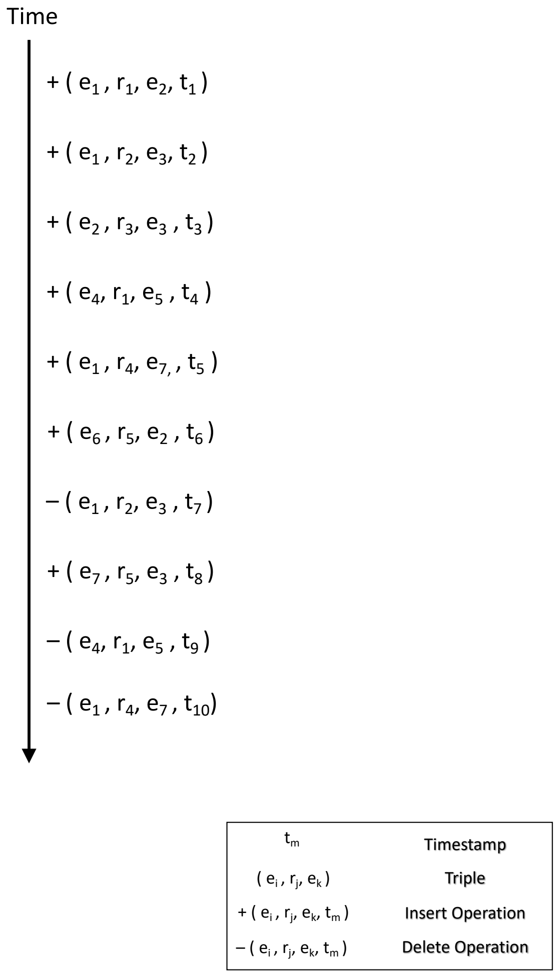

Time
+ (e_1 , r_1, e_2, t_1)
+ (e_1 , r_2, e_3, t_2)
+ (e_2 , r_3, e_3 , t_3)
+ (e_4, r_1, e_5 , t_4)
+ (e_1 , r_4, $e_{7,}$, t_5)
+ (e_6 , r_5, e_2 , t_6)
– (e_1 , r_2, e_3 , t_7)
+ (e_7 , r_5, e_3 , t_8)
– (e_4, r_1, e_5 , t_9)
– (e_1 , r_4, e_7 , t_{10})
t_m Timestamp
(e_i , r_j, e_k) Triple
+ (e_i , r_j, e_k, t_m) Insert Operation
– (e_i , r_j, e_k, t_m) Delete Operation

Figure 2: Evolving KG represented by a sequence of triple operations.

Accordingly, we encode an evolving KG by a sequence of triple operations, as they are depicted in figure 2. A triple operation specifies an operation type, i.e. an insertion or deletion expressed by a "+" or "–" sign respectively, a triple (h, r, t) and a timestamp t and. Given the triple operation – (e_1, r_4, e_7, t_5), it indicates that the triple (e_1, r_4, e_7) has been removed from the KG at timestamp t_5. Note that triple operations same like quadruples in the context of temporal KGs, include temporal meta information. However, temporal KGs specify the time frame in which a triple hold. In contrast, triple operations are related to the development of the KG and specify when a triple has been added or removed from its knowledge base. The quadruple (*"Gerhard Schroeder"*,*"position held"*, *"Federal Chancellor of Germany"*, *"1998-2005"*) e.g. expresses that the person *"Gerhard Schroeder"* held the position *"Federal Chancellor of Germany"* from 1998 until 2005. The triple operation + (*"Gerhard Schroeder"*,*"position held"*, *"Federal Chancellor of Germany"*, *"2015-05-20 12:45:08"*), in turn, documents that the corresponding triple was inserted into the KG at *"2015-05-20 12:45:08"*.

2.2. Fundamentals of Knowledge Graph Embedding

In the Machine Learning literature, the discipline of (statistical) relational learning focuses on the information extraction and inference out of relational data (Nickel et al., 2016; Rossi et al., 2012). Applications, which reason over information contained in a KG, often involve numerical algorithms (Wang et al., 2017). Since a KG is a symbolic representation, relying on such a data structure makes underlying computations resource-intensive (Jia et al., 2016; Wang et al., 2017). The assignment of numerical values to a knowledge network's entities and relations is a sufficient approach, as it approximately inherits their global relational information (Wang et al., 2017). These representations encode KG elements semantically with regard to their relational nature and empower the efficient operation of downstream tasks. (Jia et al., 2016; Wang et al., 2017; Xiao et al., 2016).

The research direction of KG embedding emphasizes this numerical transformation (Nickel et al., 2016; Wang et al., 2017). Related embedding models are designed according to the following scheme: First, they specify representations for KG entities and relations, second, they formulate a scoring function expressing the plausibility of a triple, and lastly, they learn embeddings for entities and relations by minimizing a loss and simultaneously maximizing the plausibility of facts observed in the KG (Wang et al., 2017).

In most cases, entities are represented as vectors onto a continuous vector space. In various other methods, entities are constituted, for example, as multivariate Gaussian distributions (He et al., 2015). A specific entity may either act as a subject (head entity) or object (tail entity) within a relation (Bordes et al., 2013). Embedding techniques take this aspect into account. They learn the vector embeddings by occupying relational information of entities, distributed in the entire KG (Jia et al., 2016; Wang et al., 2017).

Embeddings of relation types are realized differently in the literature (Wang et al., 2017). In some works, they are manifested as vectors that translate between the vector representations of entities (Bordes et al., 2013; Jia et al., 2016). Other representations lead from matrices, tensors, multivariate Gaussian distributions to mixtures of these types (Wang et al., 2017).

In general, graph elements are embedded by sequentially minimizing a predefined loss function (Bordes et al., 2013; Jia et al., 2016). The loss term incorporates the plausibility of a triple expressed by a score function f. It assigns a plausibility score also called energy score denoted as $f_r(h, t)$, to a triple (h, r, t) (Bordes et al., 2013). The learning procedure's objective is to learn entity and relation embeddings by maximizing the plausibility of facts, i.e. triples which hold in the KG (Wang et al., 2017).

The formulation of the energy score (i.e. plausibility score) varies along with embedding techniques. Some methods characterize the degree of plausibility by a probability for the existence of a triple (Nickel and Tresp, 2013). Other methods implement it as a distance in the continuous vector space (Bordes et al., 2013; Jia et al., 2016).

2.3. Training of Knowledge Graph Embedding Techniques

KG embedding models are typically trained under two alternative assumptions: the closed-world and the open-world assumption (Wang et al., 2017).

The closed-world assumption states that the information depicted in a KG is complete, implying that semantic relations between entities that are not observed in the KG are considered to be false (Wang et al., 2017). For training under the closed-world assumption, representations for KG elements can be learned by minimizing the squared loss, for example, formulated as follows:

$$L = \sum_{h,t2E,r2R} (y_{(h,r,t)} \quad f_r(h,t))^2.$$

Using the notation from section 2.1, E and R are sets of KG entities and relations, $f_r(h, t)$ is the energy score of the triple (h, r, t) and $y_{(h,r,t)}$ is the truth value of (h, r, t) equal to 1 if the triple holds or 0 otherwise. By seeking to attach plausibilities towards 1 to facts and vice-versa towards 0 for negative triples, embeddings of KG elements are learned (Wang et al., 2017).

For negative examples, all triples that are not observed in the graph are gathered. Given a sparse KG with an immense number of entities and relations, a training procedure under the closed-world assumption is enormously resource-intensive (Wang et al., 2017). Furthermore, since KGs are often incomplete in the real world, i.e. not all facts are observed, respectively learned embeddings cannot generalize well in such environments (Shi and Weninger, 2018).

Contrary to the closed-world assumption, the open-world assumption states that a triple must not be necessarily false if it is not observed in the KG (Wang et al., 2017). Related training procedures form KG embeddings by minimizing a pairwise ranking loss, for instance, stated as follows:

$$L = \sum_{(h,r,t)2T,(h^0,r,t^0)2T^0}^{n} max(0, \gamma + f_r(h,t) \quad f_r(h^0,t^0)).$$

Here, (h, r, t) is a positive and (h^0, r, t^0) is a negative example.

In contrast to the closed-world assumption, not all false triples from the KG are incorporated as negative examples for training. Typically pairs of positive and negative examples are used as depicted in the ranking loss. This makes the training procedure more scalable to KGs with a large number of entities and relation (Bordes et al., 2013; Tay et al., 2017; Wang et al., 2014). To form a negative counterpart, a fact's head or tail entity is replaced by a random entity from the KG (Bordes et al., 2013; Wang et al., 2014).

Compared to the closed-world assumption, the loss term here is already satisfied if the energy scores of the positive example $f_r(h, t)$ and the negative example (h^0, r, t^0) differ by the margin value γ. To provide an illustration, it can be assumed that the energy scores for the positive example („*Rome*“, „*is capital of*“, „*Italy*“) and a negative example („*Turin*“, „*is capital of*“, „*Italy*“) should have a minimum distance of γ. In this sense, negative examples are not considered implausible but relatively more implausible than positive examples, which makes learned embeddings generalize better in downstream tasks (Wang et al., 2017). For this reason, we focus on training procedures that are carried out under the open-world assumption in this thesis.

The training procedure conducted under the open-world assumption is typically conducted in minibatch mode (Bordes et al., 2013). Minibatches are subsets of the training set of the KG. The model first captures all entities and relation of the KG and initializes their representations with random values (Bordes et al., 2013; Nickel et al., 2011). Then the mini-batches of positive training examples are iterated whereby for each positive triple (h, r, t) a negative counterpart is generated by either replacing h or t with a random entity from the KG (Bordes et al., 2013; Tay et al., 2017). Based on these training examples, the model optimizes its embeddings by minimizing the loss, which implies to maximize plausibilities for facts and minimize them for negative examples (Wang et al., 2017).

2.4. Evaluation Tasks

To evaluate the quality of learned representations, most KG embedding works in the literature focus on the tasks of link prediction and triple classification, which are outlined in the following passages. The general approach is to exploit embeddings and calculate energy scores for positive and negative test examples of the KG in order to check whether positive facts are considered as more plausible by the model (Bordes et al., 2013).

Triple Classification Triple classification is a binary classification task and has been conceptualized in the work of Socher et al. (2013). Basically, a model is used to classify a series of test examples as true or false. For each test triple, a negative example is constructed by replacing the head or tail entity with another KG entity.

For all these examples, their energy scores are calculated by using the evaluated model. A test example (h, r, t) is then classified as true or false, depending on whether the associated energy score $f_r(h, t)$ is lower or higher than a relation-specific threshold T_r or a universal classifier T (Han et al., 2018; Socher et al., 2013). The threshold value is obtained by calculating the energy scores for validation or test triples and maximizing the model accuracy (Han et al., 2018; Socher et al., 2013).

Link Prediction The basic workflow in the link prediction is conceptualized as follows. Starting from a test triple (h, r, t) we form a set of corrupted triples by exchanging the head entity h and the tail entity t to (h, r, t^0) or (h^0, r, t) (Bordes et al., 2011). The head and tail entities are exchanged by all entities which are present in the KG (Bordes et al., 2011).

Given a positive test example and the set of negative examples, KG embeddings are used to calculate their plausibility scores. In the next step, all candidates are sorted according to these energies to determine the rank of the test triple (Bordes et al., 2013).

By performing this ranking task for all test examples and averaging the obtained ranks, the first evaluation measure of the Mean Rank is calculated. Further, the ratio of test examples ranked among the top n is measured and expressed in the Hits@n (typically Hits@10) metric. In conclusion, a higher Hits@n value altogether with a lower Mean Rank score expresses a high model quality.

It is possible that by substituting the head or tail of a triple to generate negative examples, false negatives are produced (Bordes et al., 2013). In the ranking task, such examples could then be ranked higher than the actual test triple (Bordes et al., 2013). This would diminish the performance of the model. To deal with this issue the link prediction task differentiates the calculation of the related evaluation metrics in two settings: The first is the raw setting which takes false negatives in the ranking tasks into account, followed by the filtered score where the test triple is solely ranked among true negatives (Bordes et al., 2013).

For our evaluation framework, we use both triple classification and link prediction as test scenarios. However, both tasks are mainly used in the literature in connection with a static KG. As we explain in section 5, we had to specify them for their application in the context of an evolving KG.

2.5. Benchmark Datasets

In the following, subsets of real-world KGs are presented which have been widely used in the domain of KG embedding. These benchmark datasets are extracted from the sources of Freebase, WordNet and YAGO. Each consists of a training, validation and test subset. Table 1 documents statistics about the number of their entities, relation types, training and evaluation triples.

	Freebase		WordNet		YAGO
	FB15k	FB15k-237	WN18	WN18RR	YAGO3-10
Entity Types	14,951	14,541	40,943	40.943	123,182
Relation Types	1,345	237	18	11	37
Train Ex.	483,142	272,115	141,442	86,835	1,079,040
Valid Ex.	50,000	17,535	5,000	3,034	5000
Test Ex.	59,071	20,466	5,000	3,134	5000

Table 1: Statistics about benchmark datasets for the evaluation of static embedding models.

Freebase15k and Freebase-15k-237 The two datasets are subsets of the Freebase KG. The former dataset was introduced by Bordes et al. (2013). Freebase was created and maintained collaboratively (Bollacker et al., 2008). It describes people, locations, media and other things of the real world. Toutanova et al. (2015) introduced a subset of FB15k called FB15k-237. This subset was extracted to provide a more realistic knowledge base, due to the high number of redundant facts in form of inverse relations (Akrami et al., 2018; Toutanova et al., 2015). Akrami et al. (2018) outline that these redundant triples contained in the FB15k dataset are not realistic and falsely improve the performance of KG embedding techniques.

WordNet-18 and WordNet-18-RR WordNet18 (WN18) is a dataset extracted by Bordes et al. (2013) from the WordNet graph. The database was created in 1985 by the Princeton University, New Jersey in the U.S. (Miller, 1994). The development object was to provide a lexical online database for the English language which is able to be processed by machines (Miller, 1994). Accordingly, it is composed of English nouns, verbs, adjectives and adverbs which are interconnected by semantic relation types (Miller, 1994). Like the FB15k dataset, the WN18 dataset includes highly redundant facts with relations which are are included twice in the test dataset in form of inverse counterparts (Dettmers et al., 2018). Hence, the results of underlying KG embedding models may be artificially improved. For this reason, Dettmers et al. (2018) excluded inverse relations from the WN18 subset and published the results by releasing the WN18RR dataset.

YAGO3-10 The YAGO3-10 dataset is a subset of the YAGO3 database created by the Max-Planck-Institute in Germany. Its knowledge is sourced from Wikipedia and WordNet to merge them uniformly into a database (Suchanek et al., 2007). As well as Freebase, it stores relational information about real world objects. These include people, organisation, locations, books, etc. (Suchanek et al., 2007). In their work

Mahdisoltani et al. (2015) extracted the dataset which is used in the KG literature for evaluation purposes. The creator states that it possesses a minimum degree of complexity: specifically, each of the contained entities participates in a minimum number of 10 different relation types (Dettmers et al., 2018; Mahdisoltani et al., 2015).

2.6. TransE

To give a more detailed insight into KG embedding discipline, we review some popular KG embedding techniques in the following sections and describe how they represent entities and relations, how their loss functions are constructed and reflect on their strengths and weaknesses. In contrast to our work's core, which is devoted to evolving KGs and incremental KG embedding, these models are mainly designed for static KGs. However, because the majority of incremental models are based on them, it is essential to have a basic understanding of their concepts.

Beginning with the work of Bordes et al. (2013), the authors designed an embedding method called TransE, which represents KG elements by mapping them onto a continuous vector space (Bordes et al., 2013). For each entity and relation a vector with k dimensions is learned by considering the energy function $f_r(h,t)$ which expresses the plausibility of the fact (h,r,t). $f_r(h,t)$ is stated as follows:

$$f_r(h,t) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{1/2},$$

where $\mathbf{h}$, $\mathbf{t}$, $\mathbf{r} \in \mathbb{R}^d$ manifest the vector embeddings of the KG elements h, t and r, and $\|.\|_{1/2}$ either describes the L1 or L2 norm. Note that we follow this notation in the rest of this thesis and depict the vector representations of KG entities and relations as boldface letters.

Basically, TransE forms its embeddings by maximizing the plausibility expressed by the energy function $f_r(h,t)$ in case, (h,r,t) is a triple observed in the KG. In other words, the embedding of h plus the embedding of r should nearly result in the embedding of t, i.e. $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ (Bordes et al., 2013). Therefore, a relation is referred to as a vector translation in the k-dimensional space.

Enforced by the pair-wise ranking-loss of TransE, which incorporates the energy function f, the model tends to maximize energies of facts while minimizing energies of negative triples. The loss term is defined as:

$$L = \sum_{(h,r,t)\in S,(h',r,t')\in S'}^{n} max(0, \gamma + f_r(h,t) - f_r(h',t')).$$

Here, the expression $max(0,x)$ sets x to 0 in case $x < 0$. S is the training set of the KG T, and contains positive triples, whereas S' includes negative triples based on corrupted triples from the KG. Following the open-world assumption, the margin γ is used to define a minimum distance between the energies of positive and negative facts. The model is trained by using stochastic gradient descent in minibatch mode as it iterates the training examples and optimizes its embeddings by minimizing the loss. Bordes et al. (2013) argue that the motivation behind TransE is based upon its efficiency for KGs with a considerable amount of elements as the complexity linearly grows with the number of entities, relations and predefined dimensions. Simultaneously, it should be capable of sustaining an adequate explanatory power manifested by the resulting embeddings (Bordes et al., 2013). Thereby, the trade-off between the complexity of the model and its predictive power is tackled (Bordes et al., 2013).

The authors state that TransE performs especially well for KGs with 1:1 cardinalities. Nevertheless, a KG typically also includes relations with mapping properties of 1:n, n:1 or n:n (Wang et al., 2014). TransE is not suited to map such cases adequately (Wang et al., 2014). It tends to assign similar vector representations to entities that, for instance, act as head entities and are connected to the same tail entity in the same type of relation (Nickel et al., 2011; Wang et al., 2017).

Given an example for such a n:1 relation, soccer players would be involved in the same semantic relation to their club according to the following facts.

(*"Luis Suarez", "Member of sports team", "FC Barcelona"*)
(*"Lionel Messi", "Member of sports team", "FC Barcelona"*)
(*"Marc-Andre Tersteegen", "Member of sports team", "FC Barcelona"*)

Suppose these soccer players possess similar embeddings and are involved in facts with other type of relations and entities. In that case, the model's predictive power suffers. Continuing the illustration, we adopt the test example (*"Lionel Messi", "is father of", "Thiago Messi"*), and subject the model to the task of link prediction. It shall predict the missing entity for the incomplete triple (?, *"is father of", "Thiago Messi"*). Since the soccer players possess similar embeddings, the model could falsely complete the tuple with any teammate of *"Lionel Messi"*, namely *"Marc-Andre Tersteegen"* or *"Luis Suarez"*.

2.7. TransH

TransH is a knowledge embedding technique developed by (Wang et al., 2014) and learns the same as TransE vector representations for KG elements. It is mainly designed to address the flaws of TransE in dealing with reflexive, 1:n and n:1 relations as we have discussed them at the end of section 2.6.

To tackle these issues TransH, allows entities to have different roles, i.e., different vectors in different type of relations (Wang et al., 2014). Each relation r in the KG is therefore represented by a hyperplane $\mathbf{w}_r$ and a perpendicular vector $\mathbf{r}$, where $\mathbf{w}_r, \mathbf{r}$ *2* R^d (Wang et al., 2014). Rather than calculating the plausibility of a fact (h, r, t) directly in the vector space, as approached by TransE, entities are first mapped onto the relation-specific hyperplane (Wang et al., 2014). Integrating these entity projections into the energy function f of TransE, results in:

$$f_r(h,t) = \quad k(\mathbf{h} \quad \mathbf{w}_r^T\mathbf{h}\mathbf{w}_r) + \mathbf{r} \quad (\mathbf{t} \quad \mathbf{w}_r^T\mathbf{t}\mathbf{w}_r)k_2^2.$$

Next, TransH follows the same training procedure as TransE and optimizes its embeddings using stochastic gradient-descent in mini-batch mode to minimize a pair-wise ranking loss. Besides, a soft constraint regularizes the model (Wang et al., 2014). It comprises the normalization of the entity embeddings, the relation embeddings, and the normal vectors of the relation-specific hyperplanes $\mathbf{w}_r$. Supplementary, it enforces the hyperplane $\mathbf{w}_r$ and $\mathbf{r}$ to remain perpendicular towards each other. Although TransH doubles the number of parameters in TransE due to hyperplane projections, it possesses the same space and time complexity (Wang et al., 2014). After evaluating the benchmarks of FB15k and WN18, the author showed that their model significantly performed better for 1:n and n:1 relations than TransE (Wang et al., 2014).

2.8. RESCAL

In contrast to the introduced methods of TransE and TransH, which are based on a translation in the vector space, RESCAL learns entity and relation embeddings through the conduction of tensor decomposition. This model has been developed by Nickel et al. (2011). A three-way tensor X models the relations of a KG, where the first two dimensions set out the existence of a fact between two entities i and j and the third mode k represents the specific relation types (Nickel and Tresp, 2013). If i and j are connected by relation k in the KG and compose a fact, the corresponding tensor entry X_{ijk} should be 1 (otherwise 0). The decomposition of a tensor into its factors is sufficient because KGs are often high-dimensional and sparse (Nickel and Tresp, 2013). RESCAL comes up with a rank-r factorization of the tensor where each tensor slice X_{ijk} is factorized into the following product:

$$X_k = AR_kA^T.$$

Here, A is a $n \quad d$ matrix and R_k is $d \quad d$ matrix. The factorization implicitly learns unique entity representations included as row vectors in the matrix A in d dimensions (Nickel et al., 2011). The tensor factorization approach captures the co-occurrences of entities into their representations, characterised as latent components. The matrix R accordingly differentiate these interactions along each relation type k of the KG (Nickel et al., 2011). A major difference to the explained models of TransE and TransH can be recognized: whereas these two models ground plausibility on distances in the vector space, RESCAL is characterised as a semantic matching model by deriving relational semantics through tensor factorization (Wang et al., 2017). The underlying loss function is drafted as follows:

$$L = \frac{1}{2}\sum_k (kX_k \quad AR_kA^T k_F^2) \tag{1}$$

The term measures the squared error of the predicted value for a specific fact and its existence. Converted to an element-wise loss, 1 is reformulated as:

$$L = \frac{1}{2}\sum_k (X_{ijk} \quad a_i^T R_k a_j)^2 \tag{2}$$

where a_i, a_j 2 R^d. At this point the nature of relational semantics can be explained. The calculation of the embedding a_i of entity i relates to a linear regression problem by supposing that R_k and a_j are constant values. Thus, a_i depends on the values of a_j, R_k and the existence of the underlying triple (i, k, j) (Nickel et al., 2011). For the reason that a_j is derived from the relations of entity j, indirect relations of i are incorporated into a_i.

The input tensor X_{ijk} is approximated by solving a multi-linear optimization problem (Nickel et al., 2011). As a result, the matrix A and the slices of the tensor R can be computed directly in alternation (Nickel et al., 2011). The capturing of relational semantics of entities enables potential downstream tasks like the classification of entities concerning their latent components (Nickel et al., 2011). The complexity of the model grows linearly with respect to the number of entities and dimensions. Although relations are depicted as d d matrices, the underlying representations face a quadratic growth of parameters (Nickel et al., 2011).

3. Related Work

In this section, we cover the research direction of incremental KG embedding. There are only three publications in which related techniques have been introduced to the best of our knowledge. To distinguish incremental KG embedding models, we perform a classification of embedding techniques in the first place.

Afterwards, we review these related works. Systematically, we describe their concepts and examine the strategy used to enable incremental learning. Of particular interest for our evaluation framework is how the authors evaluate their techniques. Therefore we complete each review and analyze the strengths and weaknesses of their evaluation settings. As we will see in section 5, we used our analysis results to identify requirements for our evaluation framework.

3.1. Classification of Knowledge Graph Embedding Techniques

To classify embedding techniques, we take two aspects into account. First, we classify KG embedding techniques, whether they consider temporal qualifiers on facts in their representations. As we described in 2.1, there are temporal KGs, which specify when a triple has been valid in time (Lacroix et al., 2020). Related embedding techniques refer to the domain of temporal KG embedding and manifest these constraints in their embeddings (García-Durán et al., 2018). On the other hand, there are methods based on non-temporal KGs, as we have treated them so far. These methods ignore the time restriction of a fact and consider a triple as valid if it exists in the KG. Contrary to the temporal KG embedding, we refer to such procedures as non-temporal KG embedding models.

Next, we differentiate whether a method can optimize its learned embeddings incrementally, which in the literature is referred to online embedding learning (Tay et al., 2017; Wu et al., 2019). A KG, by its nature, evolves. With triple insertions or deletions, its collection of facts changes, and new entities or relations may emerge or disappear from the graph (Jia et al., 2016; Wu et al., 2019).

The majority of models, including the representatives of TransE, TransH, and RESCAL, are designed for static KGs. Here the set of facts, entities, and relation types is fixed (Wu et al., 2019). If these methods are applied to a changing KG, they cannot efficiently update their embeddings (Tay et al., 2017).

We illustrate this issue by taking the model of TransE as an example. At the beginning of the training procedure, the model indexes all entities and relations present in the KG and initializes their vector representations. In the training procedure the representations are then learned by following the constraint that $\mathbf{h}+\mathbf{r}$ $\mathbf{t}$ applies for each fact (h, r, t) in the KG. Suppose that after the optimal embeddings are found, a new triple (e_1, r_1, r_2) is added to the KG, where e_2 is a newly inserted entity. First, the model initializes a vector representation $\mathbf{e_2}$. The embeddings $\mathbf{e_1}$, $\mathbf{r_2}$ and $\mathbf{e_2}$ then have to be optimized to fit $\mathbf{e_1} + \mathbf{r_2}$ $\mathbf{e_2}$ (Wu et al., 2019). If these previously learned vectors are modified now, the general constraint $\mathbf{h} + \mathbf{r}$ $\mathbf{t}$ does not apply any more for other facts in which e_1 and r_2 are included (Jia et al., 2016; Wu et al., 2019). In conclusion, a chain reaction along the complete KG, can make it necessary to learn the entire KG again (Tay et al., 2017). By scaling this procedure to KG with a large number of elements and a high frequency of updates it requires a large amount of time and memory which is costly and inefficient (Jia et al., 2016).

Several methods address these issues and can be applied to a frequently changing KG, by learning embeddings incrementally (Tay et al., 2017; Wu et al., 2019). With the insertion or deletion of facts, they create embeddings for unseen entities or relations, by simultaneously updating and maintaining representations of existing ones (Wu et al., 2019). While methods designed on static graphs are referred to static or transductive KG embedding in the literature, those based on a changing set of facts are related to incremental or inductive KG embedding. We adopt these terms in our terminology and describe such techniques as static or incremental KG embedding models in the remainder of this thesis.

Besides, another area associated with KG embedding is called Graph representational learning. It includes techniques like GraphSAGE (Monti et al., 2017) for embedding continuously changing graphs, CTDNE, which learns temporal graph patterns, or TGAT covering both capabilities (Monti et al., 2017; Nguyen et al., 2018; Xu et al., 2020). Nevertheless, they are only applicable to undirected graphs. Unlike KGs, these graphs do represent interactions between entities rather than semantic relations. Hence, they neglect relational semantics and solely learn representation for entities but not for relations (Wu et al., 2019). We, therefore, exclude them in this classification and define the first restriction of related works exclusively to KG embedding methods.

Table 2 lists existing KG embedding techniques and classifies them according to the outlined characteristics. The model of TransE, for instance, is conceptualized for static data and ignores temporal validity for triples. Know-Evolve and TA-TransE (García-Durán et al., 2018; Trivedi et al., 2017) are also based on static KGs but incor-

	Incorporate temporal qualifiers on facts?	Support online learning on evolving knowledge graphs?
RESCAL (Nickel et al., 2011)		
TransE (Bordes et al., 2013)		
TransH (Wang et al., 2014)		
(i)TransA (Jia et al., 2016)		X
PuTransE (Tay et al., 2017)		X
Know-Evolve (Trivedi et al., 2017)	X	
TA-TransE (García-Durán et al., 2018)	X	
DKGE (Wu et al., 2019)		X
This thesis (2020)		X
Potential future work	X	X

Table 2: Classification of KG embedding techniques.

porate temporal constraints of facts. In line with the works of Jia et al. (2016), Tay et al. (2017) and Wu et al. (2019) we devote this thesis to incremental, non-temporal KG embedding. Implicitly we ignore temporal KG embedding but cover static KG embedding partially as we compare related techniques with incremental ones in our evaluation framework. To our best knowledge, only three incremental concepts are existing in the literature. These include iTransA, PuTransE and DKGE which we review in the upcoming sections (Jia et al., 2016; Tay et al., 2017; Wu et al., 2019).

3.2. PuTransE

Parallel Universe TransE, called PuTransE, is a technique developed by Tay et al. (2017). It supports online embedding learning and refers to the concepts of TransE and TransH in deriving KG embeddings by maximizing the plausibility of a fact through translations in the vector space.

Contrary to these models, PuTransE is not restricted to a single embedding space. Instead, the KG is divided into multiple randomly constructed training sets, each embedded into separate vector spaces (Tay et al., 2017). The authors associate their methodology to the concept of random forests used in a classification task, as they are composed of multiple decision trees, whose individual calculation outcomes contribute to an overall prediction (Tay et al., 2017).

A training set denoted X_i, is constructed so that the contained triples are gathered from a common semantic context in the KG. The formation of an embedding space (called universe) 4_i 2 4, where 4 is the set of all embedding spaces, follows a certain scheme: The first step is the sampling of a semantic focus, i.e., a relation r_i. Then all triples (h, r_i, t) in which r_i occurs are retrieved from the KG (Tay et al., 2017). Starting from the entities of these triples, a bidirectional random walk traverses the KG while collecting a predefined number of training examples for the compilation of X_i. In this way, the corresponding triple selection scheme considers semantic and structural properties of the KG (Tay et al., 2017).

PuTransE forms an embedding space 4_i by following the approach of TransE. So, every entity and relation involved in X_i is assigned to a local vector representation, learned by maximizing the plausibility $f_r(h,t) = \quad k\mathbf{h} + \mathbf{r} \quad \mathbf{t}k$ for each $(h,r,t)\ 2\ X_i$ (Tay et al., 2017). Basically, also other techniques can be applied to create 4_i (Tay et al., 2017). TransH, for instance, would map entities to vectors and relations to both vectors and hyperplanes. Accordingly, the model could be termed PuTransH. Likewise, TransE and TransH, PuTransE generates corrupted triples but replaces a head or tail entities with an entity of the same embedding space. Accordingly, the loss function is defined as:

$$L = \sum_{(h,r,t) 2\, 4_i} \; \sum_{(h^0,r^0,t^0) \not 2\, 4_i} max(0, \gamma + f_r(h,t) \quad f_r(h^0,t^0)).$$

The number of triples in X_i restricts to a randomly chosen hyperparameter β_i, which is significantly smaller than the total number of triples in the KG (Tay et al., 2017). Hence, a multitude of embedding spaces has to be formed for the appropriate representation of a KG. In an experiment, for instance, Tay et al. (2017) applied their model to the WN18 benchmark (see section 2.5) and created 5000 embedding spaces.

PuTransE predicts the plausibility of a triple by exploiting the global setting of embedding spaces. Referring to a global energy score, denoted as $G_r(h,t)$ it is calculated by using the formula:

$$G_r(h,t) = \max_{(h,r,t) 2} \quad k\mathbf{h} + \mathbf{r} \quad \mathbf{t}k_{1/2},$$

where Φ is the set of all embedding spaces in which h, r and t coexist, and $\mathbf{h}$, $\mathbf{r}$, $\mathbf{t}\ 2\ \mathsf{R}^d$ are the respective local representations in an embedding space $4_i\ 2\ \Phi$. In other words, for calculating the plausibility $G_r(h,t)$ of a triple (h,r,t) all embedding spaces are iterated in which h, r and t occur, to calculate the local energies of (h,r,t) and adopt their maximum value.

The model leverages the power of randomness ranging from the selection of conjunctive training sets to the configuration of hyperparameters for each embedding space (Tay et al., 2017). The representation of a KG is delegated to multiple embedding spaces learned with randomly chosen hyperparameters. In this way, the tuning of hyperparameters is not necessary, as the authors state (Tay et al., 2017).

Additional advantages are the ability to learn universes in parallel, enhancing the efficiency of the embedding process. PuTransE addresses the flaws of TransE to cope with 1:n or n:1 cardinalities. In section 2.6, we described the tendency of TransE to learn similar vector representations for entities, which are connected to the same entity by the same relation (Tay et al., 2017; Wang et al., 2017, 2014). As the authors argue, PuTransE prevents this spatial accumulation by separating KG embeddings into multiple, multiple universes (Tay et al., 2017).

In this way, also online learning is enabled. Given the situation that PuTransE already learned representations and a KG faces modifications afterwards, the model creates new embedding spaces based on the updated state of the graph. In this way, also newly added facts can be captured (Tay et al., 2017). The newly formed spaces are added to an accumulated set of embedding spaces and consequently contribute to the calculation of plausibility scores (Tay et al., 2017).

For the unlearning of a deleted fact, the authors propose two options: The first is to add new embedding spaces, as described above. This approach aims at reducing the influence of embedding spaces, created before the triple deletion, on the global energy score (Tay et al., 2017). The second option is to deprecate embedding spaces in which entities and relations of the triple coexist.

Tay et al. (2017) tested their model by using the GeoSocial26k benchmark and simulated a travel prediction scenario based on link prediction. GeoSocial26k comprises three successive time states of a KG, called snapshots. It stores social media data that record semantic relations between users, hashtags and locations. The authors trained PuTransE on a snapshot and afterwards gathered test examples, which were facts of the next snapshot, documenting which locations users have visited. By applying the task of link prediction and measuring the Hits@1 score, the model was basically instructed to predict these facts, i.e. places a user visited correctly.

In conclusion, the authors formulate an authentic setting to evaluate their model. They used an evolving KG and based on the contained social media data, derive a real-world test scenario to apply incrementally learned embeddings. However, despite the test case's authenticity, we consider this scenario too specific to be utilized as a benchmark for evaluating incremental models. Furthermore, this scenario is mainly tailored to the data stored by GeoSocial26k. Therefore, the question remains on how to transfer the approach of learning a model on a snapshot and testing it on a subset of the next snapshot to other evolving KGs.

3.3. (Incremental) TransA

The embedding technique of TransA extends the concept of TransE. It similarly learns vector representations for KG entities and relations. In their work Jia et al. (2016) argue that the optimal margin in the training process should consider the locality of a KG. The authors conduct an experiment in which they extract five subsets of the Freebase dataset FB15k and show that these artificially constructed sets are optimally trained with different margins, which indicates that the margin value influences a model's performance (Wang et al., 2014). For TransE and TransH, the optimal margin is found during training by performing hyperparameter tuning and used globally for all training examples.

In contrast, given a positive training example (h, r, t), the method of TransA is locally-adaptive, meaning that it finds the optimal margin during training time and calculates it for a entity-relation pair, i.e h, r or r, t (Jia et al., 2016). Denoted as M_{opt}, the optimal margin is composed of a entity-specific margin M_{ent} (based on h) and a relation-specific margin M_{rel} (based on h and r). As shown in the following formula, these two margins are weighted with a factor μ and summed into

$$M_{\text{opt}} = \mu M_{\text{ent}} + (1 \quad \mu) M_{\text{rel}} \ ,$$

where μ is a hyperparameter and set to $0 \leq \mu \leq 1$. In the following passages, we assume the entity-relation pair h and r to explain M_{opt}.

The entity-specific margin M_{ent} is based on h and the set of relations R_h in which the entity is involved (Jia et al., 2016). For h and a relation $r_i \in R_h$ all positive and negative tail entities of h in respect to r_i are identified (Jia et al., 2016). Whereas positive tail entities occur with r_i and h in a fact of the KG, negative tail entities are connected with h by another relation r', i.e. $r' \neq r_i$ and $r_i, r' \in R_h$. The entity-specific margin of h concerning r_i is then obtained by the distance of the farthest positive tail entity from h and the nearest negative tail entity in the vector space (Jia et al., 2016). If we accordingly determine the margin values for h and all other relations in R_h, average their values, we receive M_{ent}. The explained procedure is expressed in the following formula:

$$M_{\text{ent}} = \frac{\sum_{r_i \in R_h} \min_{t,t'} \sigma(\|\mathbf{h} - \mathbf{t}'\| - \|\mathbf{h} - \mathbf{t}\|)}{nr_h},$$

where the term $\sigma(x)$ returns the absolute value of x (Jia et al., 2016), nr_h is the number of relations in R_h, $\mathbf{t} \in \mathbb{R}^d$ is the embedding of a positive positive tail entity of h concerning r_i and $\mathbf{t}' \in \mathbb{R}^d$ respectively the embedding of a negative tail entity. Considering the numerator, it receives the minimum value, for the farthest positive tail entity t and the nearest negative tail entity t' of h.

The relation-specific margin M_{rel} is determined similarly. For r and h of the training example (h, r, t), the relation r is compared with all relations $r' \in R_h$, whereas $r' \neq r$, by considering their similarity. The similarity in turn is based on the length of the relation embeddings. With respect to r, the relations in R_h, are separated into similar relations $r_j \in R_h$ with $\|\mathbf{r_j}\| < \|\mathbf{r}\|$ which are smaller than $\|\mathbf{r}\|$ and dissimilar relations $r_k \in R_h$ with $\|\mathbf{r}\| \leq \|\mathbf{r_k}\|$ which are larger or equal to $\|\mathbf{r}\|$. The relation-specific margin M_{rel} is lastly obtained by:

$$M_{\text{rel}} = \min_{r_k \in R_{h,r}} (\|\mathbf{r_k}\| - \|\mathbf{r}\|),$$

where $\mathbf{r}_k, \mathbf{r} \in \mathbb{R}^d$ and $R_{h,r}$ constitutes the set of relations in R_h other than r, i.e. $R_h \setminus \{r\}$.

TransA follows a similar learning procedure like TransE and TransH. The main difference is that the balance parameter μ is configured before starting the training process to calculate the optimal margin M_{opt} for each positive training triple (h, r, t) at runtime (Jia et al., 2016).

Jia et al. (2016) provide an extended version of TransA which is called incremental TransA (iTransA). The authors examine which influence newly added triples have on previously calculated entity and relation-specific margins. Based on their analysis, they derived solutions on how to adjust the margin values incrementally after each KG update. Basically, iTransA stores its learned embeddings and the entity and relation-specific margins after a training process finished. Suppose a triple (h, r, t) is added to the KG afterwards. In that case, the model determines which margin values have to be adjusted and optimizes them by using the preserved values.

For the adaption of entity-specific margins, it is observed for which existing entities, h and t appear as positive or negative entities after the KG change (Jia et al., 2016). Likewise, the model identifies relations for which r emerges as dissimilar and changes their margins Jia et al. (2016). We will not detail the formulas to optimize margin values at this point, as their explanation would exceed this paper's scope. For a detailed explanation, we refer to Jia et al. (2016).

The updated margins are then used to continue to learn the embeddings on the modified KG. According to the authors, the training procedure converges faster than if embeddings were reinitialized and learned, due to the optimized margin values.

In conclusion, iTransA has two disadvantages in dealing with evolving KGs. First, it cannot update the margins after deletions of facts and is therefore limited to triple insertions. As we argue in section 5, forgetting facts is essential for an incremental KG embedding method, since KGs like Wikidata frequently receive triple deletions. On the other hand, the optimization scheme after a KG change is time and memory intensive, especially for large KGs. First, all entities and relations are scanned for a necessary change in their margins. Second, the complete set of facts is learned again in the subsequent training phase to update KG embeddings.

The method of iTransA was tested against the static FB15k dataset. To reproduce an evolving KG setting, two subsets based on distinctive relation types have been derived. One subset served as a base set and was used to learn the embeddings of iTransA. The second subset was added to the base set in the next step, whereas the model optimized its embedding incrementally. Along the merge process, 269 new relations and 772 new entities have been added.

We believe that the underlying evaluation scenario is not suitable for evaluating incremental models. The authors fail in reproducing an authentic, evolving KG. From the statistics in table 1, it can be concluded that the evolution of a KG typically adds more new entities than relations. An insertion of only 772 entities in 269 relations, as performed here, does not reflect this. A more severe flaw of the evaluation design is the neglect of triple deletions. Since iTransA cannot process them, they are omitted. However, a suitable evaluation scenario should include deletions to test whether a model can unlearn facts which have been reinforced before in its embeddings.

3.4. Dynamic Knowledge Graph Embedding

In their work Wu et al. (2019) introduced the incremental KG embedding technique DKGE ("Dynamic Knowledge Graph Embedding"), which is an extension of TransE. Likewise, DKGE learns vector representations and expresses the plausibility of a fact through translations in vector space. However, in contrast to TransE, the elements of a KG are represented by joint embeddings (Wu et al., 2019).

Specifically, each representation $\mathbf{o}$ assigned to a KG element o, is composed of two parts: a knowledge embedding $\mathbf{o}^k$ and a contextual subgraph embedding $\mathbf{sg}(o)$, where $\mathbf{o}^k, \mathbf{sg}(o)$ 2 R^d (Wu et al., 2019). Whereas the knowledge embedding $\mathbf{o}^k$ is the actual embedding of o, the contextual subgraph embedding $\mathbf{sg}(o)$ encodes information about its semantic context (Wu et al., 2019). The dimensions of both embeddings are weighted by a vector $\mathbf{g}$ *2* R^d and summarized to $\mathbf{o}$, as stated in the following formula:

$$\mathbf{o} \; = \mathbf{g} \quad \mathbf{o}^k + (1 \quad \mathbf{g}) \quad \mathbf{sg}(o).$$

Unlike $\mathbf{o}^k$, DKGE does not learn $\mathbf{sg}(o)$. Instead, it learns an entity-specific and a relation-specific Attentive Graph Convolutional Network (AGCN) to calculate $\mathbf{sg}(o)$.

An AGCN is an architecture that constitutes a Graph Convolutional Network (GCN) and adds an attentive layer to summarize the network's output into a vector (Wu et al., 2019). The GCN, in turn, obtains an undirected graph, represented by an adjacency matrix and a feature matrix (Kipf and Welling, 2016). The adjacency matrix records interactions between the nodes in the graph. The vectors of the feature matrix, on the other side, depict the semantic features of nodes (Kipf and Welling, 2016; Wu et al., 2019). Both matrices are processed by the GCN to produce a trained feature matrix at node-level (Kipf and Welling, 2016). The attentive layer then weights the respective feature vectors and sums them to a vector, which finally encodes the input graph (Wu et al., 2019).

DKGE uses this concept to encode the context of an element o into the contextual subgraph embedding $\mathbf{sg}(o)$ (Wu et al., 2019). The context is expressed by an undirected graph $sg(o)$, termed subgraph of o. It is conceptualized differently for KG entities and relations. We illustrate the construction of $sg(o)$ using the KG depicted in figure 3.

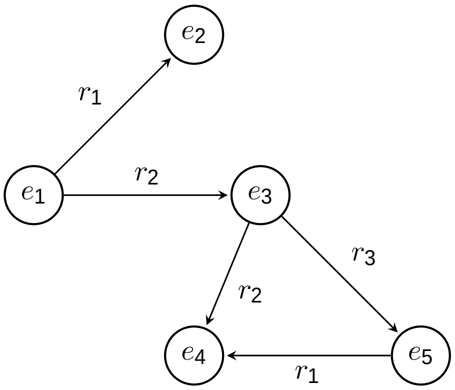


Figure 3: Illustrative KG.

If we equate the element o with entity e_1, then the subgraph $sg(e_1)$ consists of its one-hop neighbors (Wu et al., 2019). These are entities directly connected to e_1, i.e. e_2 and e_3. In the subgraph $sg(e_1)$, e_1 is then connected to e_2 and e_3 by undirected edges as shown in figure 4.

If we equate the element o to relation r_2 in contrast, the sub-graph $sg(r_2)$ is composed of relation paths. In this case, these are sequences of relations, which connect

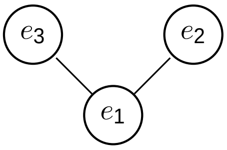


Figure 4: Contextual subgraph of entity e_1.

the same entities as r_2 in the same direction. Related to figure 3, (r_3, r_1) is a relation path of r_2, since the path likewise leads from entity e_3 (via entity e_5) to entity e_4. To construct $sg(r_2)$, the relation r_2 is connected with (r_3, r_1) by an undirected edge as illustrated in figure 5.

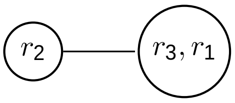


Figure 5: Contextual subgraph of relation r_2.

Based on $sg(o)$, an adjacency matrix is created and submitted to the GCN combined with a feature matrix. As the nodes in $sg(o)$ represent KG entities or relations, the vectors in the input feature matrix express contextual embeddings of KG elements (Wu et al., 2019). They are initialized together with the knowledge embeddings of elements and learned by DKGE (Wu et al., 2019). After the GCN returns the output feature matrix, the attentive layer weights the corresponding vectors and sums them into $\mathbf{sg}(o)$.

Thus, DKGE trains two AGCNs and learns a knowledge embedding and contextual embedding for each KG element. The latter representations participate in the contexts of other elements and influence their subgraph embeddings. By placing joint embeddings into the energy function $f_r(h, t)$ of TransE, we obtain:

$$f_r(h,t) = \quad k\mathbf{h} \quad + \mathbf{r} \qquad \mathbf{t} \;\; k_{1/2}.$$

Through the influence of contextual embeddings on the subgraph embeddings of other elements, DKGE supports to efficiently update its KG embeddings, as explained in the following example. Let us assume that DKGE was trained on a specific KG. So, the KG embeddings satisfy the general constraint that $\mathbf{h} \quad + \mathbf{r} \qquad \mathbf{t}$ applies for all facts (h, r, t) in the KG (Wu et al., 2019).

Now the KG is changed by an update afterwards. Accordingly, the contexts of KG elements can change with triple insertions or deletions. Concerning entities, neighboring entities emerge or disappear from their contexts. Similarly, the same applies to relations and relation paths.

We continue the example and consider a fact (e_1, r_1, e_2) that already existed before the KG update. Next, we perform a case distinction to explain how changes in the graph affect previously learned embeddings of DKGE.

In the first case, we assume that the context of the elements e_1, r_1, and e_2 has remained unchanged. Accordingly, the subgraphs $sg(e_1)$, $sg(r_1)$, and $sg(e_2)$, and the contextual subgraph embeddings $\mathbf{sg(e_1)}$, $\mathbf{sg(r_1)}$, and $\mathbf{sg(e_2)}$ remain unchanged (Wu et al., 2019). As we already outlined these subgraph embeddings combined with the knowledge embeddings $\mathbf{e_1}^k$, $\mathbf{r_1}^k$, $\mathbf{e_2}^k$ result in the joint embeddings $\mathbf{e_1}$, $\mathbf{r_1}$, $\mathbf{e_2}$ (Wu et al., 2019). Since the knowledge embeddings are conceptually context-independent, both the knowledge embeddings and the higher-level joint embeddings remain unaltered. Finally, the vectors $\mathbf{e_1}$, $\mathbf{r_1}$, $\mathbf{e_2}$ still respect the constraint $\mathbf{e_1} + \mathbf{r_1} \quad \mathbf{e_2}$, and DKGE is not required to adapt its parameters for the fact (e_1, r_1, e_2) again (Wu et al., 2019).

In the second case, suppose that from the elements in (e_1, r_1, e_2) the context of e_1 has changed. Conducting the thought experiment formulated above, if $sg(e_1)$ changes, the consecutive subgraph embedding $\mathbf{sg(e_1)}$ changes, leading to a modification of the joint embedding $\mathbf{e_1}$ (Wu et al., 2019). Hence, the constraint $\mathbf{e_1} + \mathbf{r_1}$ $\mathbf{e_2}$ no longer holds for the fact (e_1, r_1, e_2). For this reason, the model needs to optimize its parameters by learning (e_1, r_1, e_2) (Wu et al., 2019).

In deduction, DKGE not relearns the whole KG after it was modified. It instead updates its embeddings by learning new facts and only existing facts that include at least one element with an affected context.

The procedure to inductively optimize KG embeddings is designed as follows. First, DKGE determines which entities and relations have been added or removed from the KG. For removed KG elements, their contextual and knowledge embeddings are deleted. In contrast, for emerging elements, their embeddings are initialized. Following our thought experiments, the model identifies facts for which the global constraint $\mathbf{h} + \mathbf{r} \quad \mathbf{t}$ does not hold anymore. Using this triples as training data, existing and new KG elements are optimized differently in the training phase. For existing KG elements, contextual element representations and the parameters of the GCNs are fixed while only optimizing their knowledge representations. For newly emerging KG elements, in contrast, both their knowledge and contextual embeddings are learned.

Concerning the evaluation setting used in this work, the authors compile four datasets, namely YAGO-3SP, IMDB-30SP, IMDB-13-3SP. They are extracted from real-world knowledge bases and consist of multiple snapshots, i.e. time states of a KG. YAGO-3SP is a subset of YAGO and counts three snapshots. IMDB-30SP and IMDB-13-3SP, containing 30 and 13 snapshots, are sourced from the Internet Movie Database. It records semantic relations between movies, TV series, actors, directors (Wu et al., 2019). DBpedia-3SP counts three snapshots and contains facts from the DBpedia knowledge base (Wu et al., 2019).

As evaluation scenarios, the authors applied the tasks of link prediction, question answering (QA) and a scalability test.

In the context of the link prediction task, DKGE is compared with baselines like the incremental method PuTransE, and static methods like TransE, ComplEx, ConvE. Here, the datasets YAGO-3SP and IMDB-30SP are exploited. Both datasets hold a training, a validation and a test dataset for each snapshot. While the training data alternates with the snapshots, the sets of validation and test triples remain unchanged.

At each snapshot, the authors measure the metrics of Mean Rank and Hits@10 (Wu et al., 2019). At the first snapshot, all methods are learned from scratch. Whereas PuTransE is learned incrementally, in the remaining snapshots, the static methods learn their embeddings from scratch on the respective datasets. As for DKGE, the authors run their technique in an incremental and a static learning procedure. The results indicate that the static learning procedure of DKGE performed best in all snapshots, followed by the incremental version of the model (Wu et al., 2019).

For the QA task, the authors solely apply DKGE to DBpedia-3SP and optimized the model's embeddings incrementally. The task prepares ten questions, whose answers can change along with the three snapshots (Wu et al., 2019). An example is a question: *"Who is the coach of the Los Angeles Lakers?"*. The questions are converted into to form $(h, r, ?)$ (or $(?, r, t)$) to subject the model to return answers by predicting the missing entity (Wu et al., 2019). Remarkably, the tasks of QA and link prediction are similar as the model has to consider all existing entities in its answer.

The difference to link prediction is that the questions are derived from training and validation examples, rather than from test triples. So, the DKGE explicitly learned these triples. Measured by the Hits@1 score, the model's performance indicates the correctly answered ratio to the total number of questions (Wu et al., 2019).

The last evaluation task is a scalability test. It examines whether DKGE is sufficiently applicable for a large-scale evolving KG like the IMDB-13-3SP dataset (Wu et al., 2019). It counts three snapshots with averagely 8 million training triples in the related training datasets. Therefore the authors trained DKGE on the first snapshot from scratch, which took around a week to complete the training phase (Wu et al., 2019). In the other snapshots, the model optimized its embeddings incrementally. Here, in contrast, the training process took only around two hours on average (Wu et al., 2019).

Comparing the underlying evaluation scenario with the evaluation settings of iTransA and PuTransE, Wu et al. (2019) notably formulate a more comprehensive scheme for the evaluation of incremental procedures.

The authors ground their evaluation scenarios on evolving datasets extracted from real-world KGs like YAGO or DBpedia and therefore utilized accurate data. Moreover, we primarily consider the described setting used for link prediction as reasonable for evaluating incremental methods. Here, incremental methods are tested over time and compared with other incremental and static models. As we will see in section 5, we adapted this approach for our constructed evaluation scheme.

Besides, we also identified some drawbacks in their evaluation. First, the authors used datasets, including the same test examples in their different snapshots. Hence, their evaluation scenario only considers test examples that remain true throughout the evolving KG. We believe that it is more reasonable to use test data that evolves together with the KG. While it is plausible to check how a model performs on permanently valid triples, it is more interesting to check how accurate it can learn or unlearn triples. This aspect was ignored entirely by the authors.

Second, the data used to train incremental models embodies KG time states represented by static sets of facts. However, an incremental model updates its embeddings by capturing changes of a KG. These changes are not explicitly visible in snapshots, although they can be reproduced by comparing two consecutive snapshots. Nevertheless, we find it more reasonable to supply incremental models directly with KG updates. By doing so, the training process is more transparent and comprehensible.

As we will see in section 5, we considered the discussed strengths and weaknesses to derive general requirements for our developed evaluation scheme.

4. Implementing Parallel Universes in the OpenKE Framework to Enable Incremental Knowledge Graph Embedding

In this work, we have implemented the model of PuTransE and evaluated it by using our devised evaluation framework. The incremental strategy of PuTransE is based on the parallel universe methodology. Here, the static model of TransE is utilized to create local embedding spaces. These are then combined to predict the plausibility of a triple. We followed this approach, but instead of only using TransE, we further adopted the static models TransH and TransD and created the incremental models of PuTransE, PuTransH and PuTransD.

Since the authors did not publish the source code of PuTransE, we had to develop the model and the parallel universe methodology by ourselves. In the following sections, we first motivate our implementation. After that, we recap the concept of parallel universes and its strategy to enable incremental learning. In the next step, we expose essential implementation details. As described at the end of this section, we systematically evaluated our implementation to ensure its correctness.

4.1. Motivation

Besides PuTransE, we could have implemented the rest of the incremental methods discussed so far, namely DKGE and iTransA. However, we excluded them mainly because their implementation did not fit for this work's time frame. For iTransA, likewise, for PuTransE, no open-source code exists. Regarding DKGE, source code is freely available[5]. Nevertheless, we would have to perform extensive adjustments in order to use the implementation for our purposes.

The decision to implement PuTransE, PuTransH and PuTransD in our work is beneficial for the research direction of incremental KG embedding. We have based our implementation on the OpenKE framework[6]. OpenKE is an open-source frame-

[5] Accessible at `https://github.com/lienwc/DKGE/`.

[6] Accessible at `https://github.com/thunlp/OpenKE/`.

work developed in the programming languages Python and C++. Organized by the Natural Language Processing Group at the Department of Computer Science and Technology, Tsinghua University, it provides data structures to perform KG embedding. These include a range of static KG embedding models and procedures for their training and evaluation. The results of Han et al. (2018) indicate that the OpenKE implementations of TransE and TransH even exceeded the results of the original papers (Bordes et al., 2013; Han et al., 2018; Wang et al., 2014).

The integration of the parallel universe configuration into OpenKE is especially of symbiotic nature: On the one hand, we could use the well-performing and readily-made models of OpenKE to increase efficiency in developing the three incremental models. On the other hand, our implementation of parallel universes is not only compatible with TransE but also with the other OpenKE instances of TransH, TransD, TransR and DistMult. Thus, additional incremental models can be created using OpenKE to contribute to the research direction of incremental KG embedding.

Furthermore, since OpenKE only supports static KG embedding, we had to adjust the framework. Now OpenKE includes data structures that enable the training and evaluation of incremental models. With this in mind, we have created a foundation to integrate further incremental models into the OpenKE framework.

4.2. Recap of the Parallel Universe Methodology

Starting from a KG $T =$ f(h, r, t)g, the parallel universe configuration divides it into multiple subsets X_i 2 T. A static embedding method is then used to learn these subsets into separate vector spaces 4$_i$ 2 4, called universes, where 4 denotes the set of all vector spaces.

These universes are then combined to determine a global prediction $G_r(h, t)$, expressing the plausibility of a triple (h, r, t). $G_r(h, t)$ is calculated as follows:

$$G_r(h,t) = \max_{(h,r,t)2} \quad k\mathbf{h} + \mathbf{r} \quad \mathbf{t}k_{1/2}.$$

Here, Φ represents the set of all embedding spaces in which h, r and t coexist. Based on the embedding spaces 4$_i$ 2 Φ the local energy scores of (h, r, t) are calculated. The maximum of these scores is finally adopted for $G_r(h, t)$.

The combination of the parallel universe methodology with a static embedding model supports the incremental optimization of KG embeddings. In case a KG was modified, the random triple selection process described in 3.2 traverses the KG and gather triples used to create new local embedding spaces. In this way, triples can be captured that emerge with a KG update.

The same strategy is pursued concerning the unlearning of deleted facts. With an increasing number of new spaces, the impact of previously trained spaces on the global energy score is reduced (Tay et al., 2017). Another strategy to handle deletions is to deprecate spaces where the triple existed and exclude them from the global energy estimation (Tay et al., 2017). The new universes are finally combined with the set of universes created before the KG modification and contribute to the calculation of global predictions.

4.3. Basic Implementation Structure

We implemented the parallel universe methodology and the three incremental models PuTransE, PuTransH and PuTransD, by utilizing the programming languages Python and C++. Extensive operations executed in the training and evaluation procedures of incremental models have been realized in C++ due to the high performance it achieves. These include the processing of triple operations, generation of negative triples, sampling of training and test examples, and the calculation of the evaluation metrics. The entity and relation embeddings, their optimization and the parallel universes have been realized in Python by using the machine learning library PyTorch[7] with version 1.5.0.

4.4. Treatment of Missing Energy Scores

In their work (Tay et al., 2017), the authors of PuTransE neglect an important aspect of their parallel universe methodology. As mentioned before, $G_r(h,t)$ is predicted by calculating the local energies of the triple (h,r,t) in all embedding spaces in which h, r and t are present to adopt the maximum value. However, the authors do not specify how $G_r(h,t)$ is estimated in case h, r, and t are not captured in a shared embedding space.

During the evaluation of PuTransE, we noticed that no global predictions could be calculated for numerous triples. For this reason, we have implemented two techniques for the substitution of missing energy scores.

4.4.1. Negative Infinity Score Substitution

The first technique is to replace missing triple energy scores with the lowest possible value, which refers to the negative infinity value, i.e. 1 . In case the model cannot predict a triple, the key idea is to rate it as implausible as possible.

Concerning link prediction and triple classification, affected test examples are attached to the worst rank, respectively predicted as false.

4.4.2. Null Vector Substitution

Another approach we elaborated is inspired by the treatment of missing values as it is partially applied in the domain of word embeddings (Cordeiro et al., 2016). In contrast to the previous option, this technique assumes that for a triple (h,r,t) with

[7] `https://github.com/pytorch/pytorch/`.

a missing energy score $G_r(h,t)$, at least one of its entities, h or r, and the corresponding relation r share a common embedding space X_i, i.e.

$$9X_i : h, r\ 2\ X_i\ \_\ r, t\ 2\ X_i.$$

As sketched in algorithm 1, iterating all embedding spaces in which h and r (or r and t) are present, the local energy scores of (h,r,t) are determined using $kh + r \quad tk$. Since t (respectively h) does not occur in these spaces, its missing vector representation is substituted by the null vector $\vec{0}$ to calculate the local energy $kh + r \quad \vec{0}k$. Finally, for $G_r(h,t)$ the highest local score is adopted.

Algorithmus 1 : Global Energy Estimation with Null Vector Handling

```
Input  : Tuple $(h, r)$, set of embedding spaces $\Phi$ in which $h$ and $r$ are
         represented
Output : Global Prediction $G_r(h,t)$ for $(h,r,t)$
$G \quad ?$ // Initialize empty List of global energy scores
for $\Phi_i\ 2\ \Phi$ do
    $L[h,r,t] \quad kh + r \quad \vec{0}k$ // Local Score
    $G[h,r,t] \quad max(G[h,r,t], L[h,r,t])$
```

Note that this technique does not eliminate the negative infinity score approach. Given the sets of embedding spaces, Φ_h, Φ_r and Φ_t in which h, r and t are represented, this approach is not applicable, if these sets are disjunctive, i.e.

$$\Phi_h\ @\ \Phi_r\ @\ \Phi_t.$$

In such cases, we set $G_r(h,t) = \quad 1$. Logically, it is more likely that a score can be determined for a tuple (h,r) and (r,t) than for a triple (h,r,t). Contrary to the negative infinity score approach, the core idea is to provide a fair assessment here. Instead of assigning $G_r(h,t)$ to the worst possible plausibility, its replacement rather depends on the predictable elements of (h,r,t).

4.5. Enabling Incremental Learning in OpenKE

Before our implementation, OpenKE was only applicable to static KG embedding. Accordingly, the framework's training and testing procedures were based on a static KG, i.e. unchanging set of facts. For this reason, we first had to create a compatible framework environment before implementing the incremental models of PuTransE, PuTransH and PuTransD.

First, we created procedures for loading an evolving KG represented by triple operations. Second, we adapted the legacy system's training and evaluation procedures to source data from an evolving KG. Lastly, we implemented the parallel universe methodology by utilizing these procedures. We modularized our artifacts. They allow to combine the parallel universe methodology with most of the static embedding models of OpenKE. Furthermore, additional incremental models can be seamlessly integrated into OpenKE and applied in the context of an evolving KG in the future.

4.6. Novel Incremental Embedding Approaches: PuTransH and PuTransD

On top of the implemented parallel universes, we utilized the OpenKE instances of TransE, TransH and TransD to create the incremental models PuTransE, PuTransH and PuTransD. Their illustrative executions on the WN18 benchmark have been published at `https://github.com/rlafraie/OpenKE/tree/OpenKE-PyTorch/experiments/`.

4.7. Systematic Evaluation of the Implemented Embedding Techniques

To ensure the correctness of our implementation, we proceeded as follows. First, we evaluated the OpenKE instances of TransE and TransH, as they serve as fundamental elements for the incremental models of PuTransE and PuTransH[8]. We compared the evaluation results with those published in the original papers, namely Bordes et al. (2013) and Wang et al. (2014). Likewise, we evaluated the implementations of PuTransE, PuTransH and PuTransD in a next step and compared their outcomes with the results of the original PuTransE instance published in Tay et al. (2017). In the following sections, we specify the experimental setups of both steps and confront the achieved results with those of the authors.

4.7.1. Evaluation of OpenKE Modules

Bordes et al. (2013) and Wang et al. (2014) both used the benchmarks datasets WN18 and FB15k in their evaluation contexts. Regarding the evaluation metrics, they applied a link prediction task and measured the raw and filtered Mean Rank and Hits@10 score. Accordingly, we followed this evaluation protocol.

For experiments with the TransE and TransH implementations of the OpenKE framework, we selected the hyperparameters for both models among various candidates and performed a grid search to attain the best performing model. The margin γ was selected among $1, 2, 5, 10$. For the stochastic gradient in the training phase, we choose the learning rate among $0.1, 0.01, 0.001$. We configured the number of dimensions to represent entities and relations among $20, 50, 100$ and calculated the energy scores by either taking the L1 or L2 norm. We limited the number of training epochs to 1000 and conducted early stopping on the validation dataset by observing the filtered Hits@10 score.

The model of TransE performed best on the WN18 benchmark using the L1 norm, a margin γ of 5, a learning rate of 0.1 and 50 dimensions. On the FB15k dataset, in turn, it achieved the best results with the L1 norm, a margin equals 1, a learning rate of 0.1 and also 20 dimensions.

[8] The application of PuTransD shall only illustrate that the implemented parallel universes are compatible with the majority of the OpenKE models. For this reason, we excluded TransD and restricted the evaluation of the OpenKE modules to TransE and TransH.

	WN18				FB15k			
	Mean Rank		Hits@10 (%)		Mean Rank		Hits@10 (%)	
	Raw	*Filtered*	*Raw*	*Filtered*	*Raw*	*Filtered*	*Raw*	*Filtered*
TransE (OpenKE Implementation)	**190**	**178**	**78.1**	**91.6**	**217**	**102**	34	**48.11**
TransE (Bordes et al. (2013))	263	251	75.4	89.2	243	152	**34.9**	47.1
TransH (OpenKE Implementation)	**303**	**290**	**78.11**	**89.81**	**187**	**71**	**46.20**	**64.41**
TransH (Wang et al. (2014))	318	303	75.4	86.7	211	84	42.5	58.5

Table 3: Link prediction results of the OpenKE instances TransE and TransH on the WN18 and FB15 benchmarks.

On the WN18 benchmark, the optimal hyperparameters for TransH were the L1 norm, a margin γ of 2, a learning rate of 0.1 and 20 dimensions. Concerning the FB15k dataset, the best parameter candidates have been the L1 norm, a margin equals 2 a learning rate of 0.1 and 50 dimensions. We open-sourced our experiments at `https://github.com/rlafraie/OpenKE/tree/OpenKE-PyTorch/experiments/`.

As depicted in table 3, the OpenKE implementations outperformed the original models in all settings, i.e. raw and filtered and further in nearly all evaluation metrics. Although the TransE instance of Bordes et al. (2013) achieved better outcomes in the filtered Hits@10 score on the FB15k benchmark, the OpenKE instance of TransE performed significantly better in the remaining metrics and overall. The instance of TransH, in contrast, exceeded the results stated in Wang et al. (2014) in all measures.

4.7.2. Evaluation of PuTransE, PuTransH and PuTransD

After assessing the soundness of the OpenKE modules, we evaluated the implementations of PuTransE, PuTransH and PuTransD. Likewise conducted in the first step, we followed the evaluation protocol of the authors, i.e. Tay et al. (2017).

Specifically, we used the WN18 benchmark to apply PuTransE, PuTransH and PuTransH to link prediction by measuring the Mean Rank and Hits@10 scores (raw and filtered). For the substitution of missing energy scores, we followed the approaches described in section 4.4.

Tay et al. (2017) state the parallel universe configuration is non-parametric, meaning that due to creating each embedding space with randomly chosen hyperparameters, parameter tuning is eliminated.

For the ranges among to choose parameter values, we adopted the values stated in the original paper (Tay et al., 2017). We relied on the L1 norm and selected the margin γ among the range of $[1, 4]$, the learning rate among $[0.001, 0.1]$ and the number of training examples to form an embedding space among $[500, 2000]$. The number of dimensions for the embeddings of entities and relations has been set to 20. Also,

	Mean Rank		Hits@10 (%)	
	Raw	*Filtered*	*Raw*	*Filtered*
PuTransE (Tay et al. (2017)	**39**	**29**	**88.1**	**94.9**
Infinity Score Handling				
PuTransE	485	468	71.20	83.66
PuTransH	532	515	71.18	83.08
PuTransD	642	625	63.38	73.50
Null Vector Handling				
PuTransE	702	685	71.70	83.62
PuTransH	736	719	71.14	83.02
PuTransD	2347	2330	52.10	54.93

Table 4: Link prediction results of the implemented models PuTransE, PuTransH and PuTransD on the WN18 benchmark.

the number of epochs to form an embedding space has been generated randomly among the range $[500, 2000]$. For each experiment, we trained 6000 spaces and applied early stopping on the validation dataset by measuring the filtered Hits@10 score.

Table 4 shows that among our implementations, the model PuTransE performed best in all test measures, followed by PuTransH and PuTransD. Remarkably the models achieved better results by replacing missing global predictions with the negative infinity score, especially for the Mean Rank score.

Nevertheless, none of our instances achieved the results published in Tay et al. (2017). During the development phase, we made sure to implement the concept formulated in Tay et al. (2017) as accurately as possible. As the authors did not publish their source code and the parallel universes are mainly based on randomness, we could not identify any points for improvement, nor could we verify their published results.

In contradiction to their statement that the methodology is non-parametric, there was still an opportunity to tune the parameter ranges and the number of dimensions to improve evaluation results. Additionally, we recognized that the optimal model was obtained in every experiment by reaching the configured limit of 6000 embedding spaces. Hence, we could consider raising the number of embedding spaces. However, we were unable to comply with both options due to limited time resources.

5. An Evaluation Methodology for Comparing Incremental and Static Embedding Techniques

In addition to the implemented models PuTransE, PuTransH and PuTransD, we designed an evaluation framework for the thorough assessment of incremental embedding methods that we present in this section.

Considering the discussion of related studies conducted in section 3, we first review the strengths and weaknesses of existing evaluation approaches. From these weaknesses, we deduce objectives that had to be accomplished by our evaluation framework. Based on these requirements, we will then describe our evaluation framework.

5.1. Reviewing Drawbacks of Existing Evaluation Strategies

In the literature there are only three works related to incremental KG embeddings. All of them set up incremental embedding techniques and describe schemes for their evaluation. As discussed in section 3, these evaluation approaches indicate weaknesses.

Concerning the work of Jia et al. (2016), the authors divided the FB15k benchmark into two subsets. First, iTransA was applied and evaluated to a base set. The second subset then has been merged into the base set, resulting in the complete FB15k dataset, to simulate the growth of a KG. Subsequently, the model incrementally updated its embeddings and has been evaluated again. Here, the authors failed to reproduce an authentic evolving KG for two reasons: First, their approach includes only triple insertions, but ignores triple deletions completely. Second, the simulated graph evolution does not reflect authentic KG properties.

In contrast, Tay et al. (2017) and Wu et al. (2019) use real-world evolving KGs in their evaluation settings. Tay et al. (2017) exploited the GeoSocial26k dataset, which is composed of three snapshots and stores relational social media information between users, hashtags and locations. In their evaluation, the authors train an incremental model on the facts contained in one snapshot to predict which locations a user will visit in the next snapshot. The accuracy of the model is then assessed, by comparing its predictions with the truth values, i.e. with the places that a user actually visited. To be used as a benchmark for incremental methods, we consider this test case as too specific.

Solely, the work of Wu et al. (2019) describes a comprehensive evaluation strategy that inspired our framework's design. The authors divide their evaluation into three tasks: A link prediction task, a QA task and a scalability test. We summarized these tasks in section 3.4. In particular, the concept of comparing both static and incremental methods in the context of an evolving KG pursued in the first task, we consider to be useful for our evaluation framework.

In the link prediction task, the YAGO-3SP dataset is used. Extracted from YAGO, it depicts an evolving KG at three time states. Each of these snapshots contains a test, a validation and a training set of triples. The authors proceed and learn a model in each snapshot on the respective training dataset and evaluate it by using the related test dataset.

A central ambiguity we recognized in their work is that it is not transparent how they trained their incremental models. An incremental model, in general, adjusts its embeddings for the modifications made in a KG. Since a snapshot is a static dataset, KG changes are not directly indicated. Although changes could be reproduced by comparing two subsequent snapshots, we do not find any indications for such a procedure after reviewing the corresponding implementation at `https://github.com/lienwc/DKGE/`.

5.2. Goals

Encouraged by these evaluation settings' strengths and weaknesses, we derived the following goals to construct a sound evaluation framework for incremental KG embeddings:

First, following the work of Wu et al. (2019), incremental embedding methods should be evaluated and compared with static relatives in the context of an evolving KG in different evaluation tasks. It had to be analyzed how accurately the learned representations of both types capture the elements of a changing KG over time.

In addition to static and incremental learning methods, we tested another learning variant called pseudo-incremental. The literature suggests to learn static methods from scratch after a KG modification in order to accurately capture the new state of the graph (Jia et al., 2016; Wu et al., 2019). In contrast, we considered to preserve already formed embeddings of a static model and to optimize them only for graph increments, i.e. inserted facts, throughout an evolving KG. Such an approach is useful for cases where eminently more triples are inserted than deleted. In this way, one could increase the efficiency of the learning process by accepting a decrease in the accuracy of KG embeddings. Accordingly, we used a static model and learned it across an evolving KG applying the pseudo-incremental learning procedure. Subsequently, we investigated whether and to what extent it has performed worse than a static procedure that repeatedly learned its embeddings from scratch.

Second, we noticed that using snapshots for learning an incremental model leads to ambiguities, as discussed in section 3. By relying on triple operations instead, changes between snapshots of a KG are more comprehensible. Moreover, they can be explicitly communicated to an incremental model in the training process.

Following this approach also allowed us to apply static models in the course of evolving KGs. Specifically, we have trained them on the snapshots of the KG. Since a snapshot is a KG time state, we produced it by resolving the triple operations that occurred until then. Implicitly, we could freely choose the number of snapshots making our evaluation scheme more flexible than the related works, which utilize datasets with a fixed number of snapshots.

Lastly, the main objective of our evaluation framework was the comprehensive investigation of incremental models. The evaluation framework described in Wu et al. (2019) uses the same test dataset at all KG snapshots. Accordingly, these are test triplets that remain positive throughout the evolution of the KG. For the evaluation of static KG embeddings, such an approach is sufficient. Incremental embedding models, in contrast, retain and optimize their representations over time. Therefore, it is more reasonable to involve facts that have been deleted in the evolution of a KG. In this way, it shall be investigated whether an incremental model can unlearn such examples. Wu et al. (2019), in contrast, wholly ignored this aspect. To the best of our knowledge, no work yet considered such an approach.

The unlearning of facts is of particular relevance. Wikidata, for instance, maintains an archive which documents numerous deletion requests for entities and triples since the creation of the graph[9].

In our evaluation framework, we do not only consider inserted respectively valid and deleted respectively invalid triples. Instead, we observe the history of a fact in the evolution of a KG. For example, a fact can be added, deleted, and added again. Along with these operations, the truth value of the triple alternates. We categorized such cases and demanded an incremental model to predict the current truth value of a triple correctly to measure its performance.

5.3. Design

As stated, our framework targets the comprehensive evaluation of incremental models and their comparison with static and pseudo-incremental models in the evolution of a KG. We devised a methodology that ensures comparability between the different model types by supplying each model with the same training and evaluation data.

To feed our evaluation framework and support the extensive investigation of incremental models, we have constructed a new benchmark dataset. It was created by assigning the data emerging in an evolving KG to training, validation and test datasets. The compilation process will be outlined in section 6. In this section, we focus on the evaluation framework's methodology, explaining the procedures for training and evaluating models.

Starting from an evolving KG manifested by a sequence of triple operations, we partitioned it into n equally sized intervals, such that each one includes the same number of triple operations as shown in figure 6. Note that in contrast to figure 2, we do not consider timestamps in the following figures, since for our evaluation framework, only the chronological order of the triple operations is relevant. An interval results in a snapshot representing the evolving KG's time state, i.e. it contains all triples that hold then. We followed the general approach and applied a model by conducting a training procedure for each interval and performing an evaluation at the subsequent snapshot to measure the model's performance over time.

[9] `https://www.wikidata.org/wiki/Wikidata:Requests_for_deletions/Archive` last retrieved April 20, 2020.

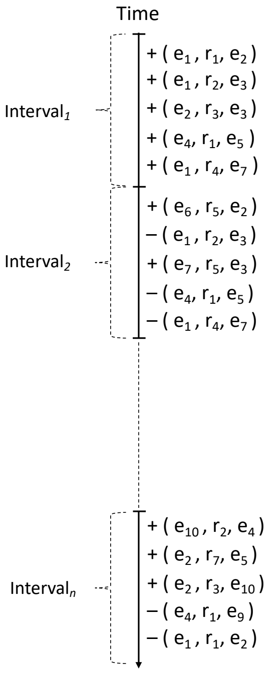

Time
Interval$_1$
+ (e$_1$, r$_1$, e$_2$)
+ (e$_1$, r$_2$, e$_3$)
+ (e$_2$, r$_3$, e$_3$)
+ (e$_4$, r$_1$, e$_5$)
+ (e$_1$, r$_4$, e$_7$)
Interval$_2$
+ (e$_6$, r$_5$, e$_2$)
− (e$_1$, r$_2$, e$_3$)
+ (e$_7$, r$_5$, e$_3$)
− (e$_4$, r$_1$, e$_5$)
− (e$_1$, r$_4$, e$_7$)
Interval$_n$
+ (e$_{10}$, r$_2$, e$_4$)
+ (e$_2$, r$_7$, e$_5$)
+ (e$_2$, r$_3$, e$_{10}$)
− (e$_4$, r$_1$, e$_9$)
− (e$_1$, r$_1$, e$_2$)

Figure 6: Division of an evolving KG into n equally sized intervals.

5.3.1. Training Procedures

Concerning training, we supply the different model types (i.e. static, incremental and pseudo-incremental models) with the same training data emerging in the KG. However, we run for each type a distinct learning procedure, which requires the training data in distinct formats.

Static embedding methods are mainly designed for a fixed set of triples. Accordingly, we allocate datasets of training triples at each snapshot, as depicted in figure 7. We call them training snapshots because they are not related to the entire graph, but depict time states of the separated training data evolving with the KG. A static model learns each training snapshot from scratch. Implicitly it only captures representations for entities and relations that are currently present at the corresponding point in time.

In contrast, we run incremental models by preserving their embeddings in the KG evolution to optimize them incrementally. Precisely, we inform a model about triple operations occurring within an interval, as shown in figure 8. Since the underlying graph is altered with an interval, we call its attached sequence of triple operations an update.

Similar to the snapshots in figure 7, the updates are not related to the entire KG but

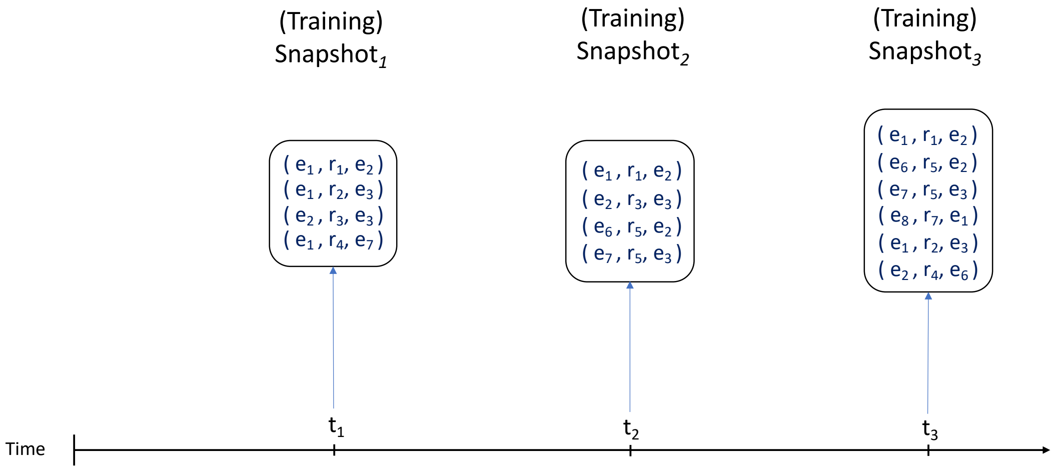


Figure 7: Snapshots used to train static KG embedding methods.

the evolving training data. An update thus explicitly indicates the changes made to the training data. Accordingly, the updates shown in figure 8 result in the training snapshots in figure 7.

An update potentially includes multiple operations for a single triple. If the number of these operations is even, they do not affect the subsequent snapshot. Assuming a triple (h, r, t) exists at a snapshot and the next update contains a delete and an insert operation for the triple, (h, r, t) will be still present in the next snapshot. An odd number of operations, on the other hand, would affect the next snapshot of the graph. Unlike a static model, an incremental might be irritated in both cases because it has to process the alternating insertions and deletions to adjust its embeddings in the same update repeatedly. As we aim at a fair evaluation for static and incremental model types, we restrict each update to one operation per triple. So we remove all related operations with an even number. If the number is odd, we only keep the first operation.

Besides, we conduct a pseudo-incremental learning procedure. Here we use a static model. However, instead of relearning an entire snapshot from scratch, we preserve its embeddings to optimize them for KG increments. They are shown in figure 9.

Given two consecutive training snapshots S_t and S_{t+1} an increment I_{t+1} is the set of triples emerging at S_{t+1}. So they were inserted with the update between the two snapshots. An increment can therefore be formalized as:

$$I_{t+1} = S_{t+1} \, n \, S_t = f(h, r, t) j (h, r, t) \; 2 \; S_{t+1} \wedge (h, r, t) \; \not 2 \; S_t g.$$

Note that an increment does not capture triple deletions. However, if a deleted triple is inserted to the graph again, it will emerge in the next increment. In figure

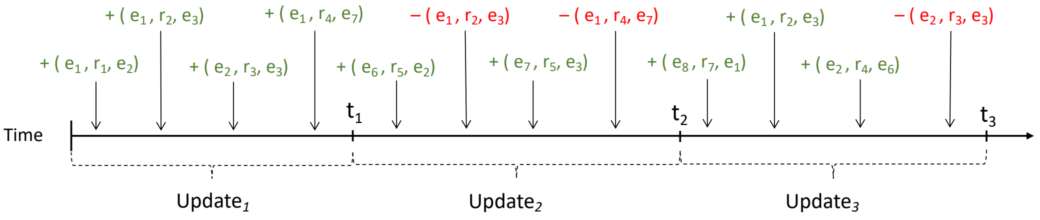

+ (e_1 , r_2, e_3)
+ (e_1 , r_4, e_7)
− (e_1 , r_2, e_3)
− (e_1 , r_4, e_7)
+ (e_1 , r_2, e_3)
− (e_2 , r_3, e_3)
+ (e_1 , r_1, e_2)
+ (e_2 , r_3, e_3)
+ (e_6 , r_5, e_2)
+ (e_7 , r_5, e_3)
+ (e_8 , r_7, e_1)
+ (e_2 , r_4, e_6)
Time
t_1
t_2
t_3
$Update_1$
$Update_2$
$Update_3$

Figure 8: Updates used to train incremental KG embedding methods.

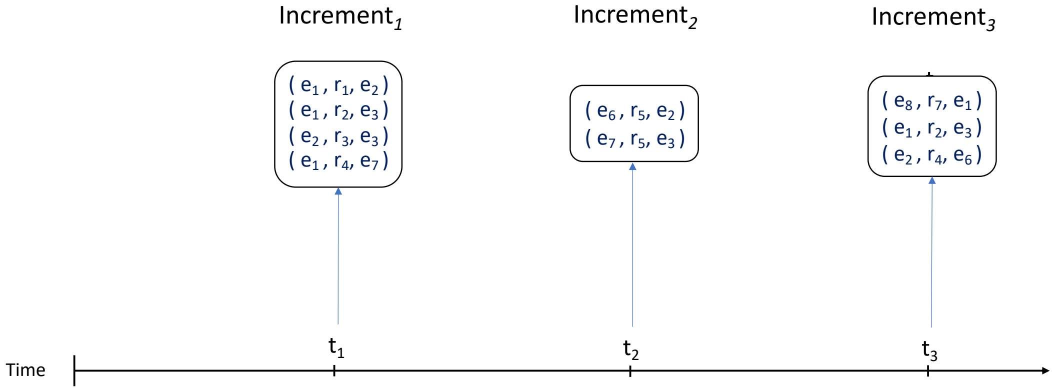


Figure 9: Increments used to train pseudo-incrementally learned KG embedding methods.

9, this is exemplified by the triple (e_1, e_2, e_3). It is initially added to the graph with $Update_1$, deleted in $Update_2$ and subsequently inserted again with $Update_3$. Hence, it is assigned to $Increment_1$ and $Increment_3$ but not to $Increment_2$.

Static models assign fixed indexes to entities and relations before learning their embeddings (Wu et al., 2019). Therefore, we inform static models about all entities and relations appearing in the graph evolution to facilitate their execution in our pseudo-incremental learning scheme. In this way, the model is capable of dealing with emerging KG elements.

5.3.2. Evaluation Scenarios

At each snapshot, we execute an evaluation procedure applied to all models using a withhold test dataset. Here, we assess a model in the tasks of link prediction and triple classification. We already reviewed the tasks in section 2.4 in the context of a fixed KG. Concerning their execution throughout an evolving KG, however, we had to specify them as outlined in the following sections.

Besides, we designed a new evaluation task, which we will outline at the end of this section. It focuses on the comprehensive evaluation of incrementally learned embeddings and is called negative triple classification. We conduct this task at each snapshot after finishing the tasks of link prediction and triple classification. To distinguish between negative triple classification and triple classification, we refer to the latter task in the remainder of this section as general triple classification.

Link Prediction In the link prediction task described in section 2.4, an embedding model ranks a positive test triple among negative triples, concerning their plausibility scores (Bordes et al., 2011). Basically, the higher a model ranks a test example, the better the model performs.

In the context of a fixed KG, the generation of negative ranking candidates is trivial. The entities h and t of the test triple (h, r, t) are substituted by all entities of the KG (Bordes et al., 2013). Since the number of KG entities remains unchanged, the number of candidates also remains unchanged.

In the context of a dynamic environment, as it is the case for an evolving KG, the number of entities might vary at each snapshot. Thus, we had to clarify the construction of negative examples according to the following identified options:

First, negative examples can be obtained by replacing the head and tail entities of test examples by all entities present at a snapshot. Implicitly, with the number of entities, the number of candidates to rank the test triple fluctuates along with the snapshots. Assume two ranking tasks with a different number of candidates and a model with fixed embedding vectors. Logically, the model would perform better in the setting where fewer possibilities are available to rank a test triple. Consequently, the evaluation results across the snapshots are not comparable.

Another option is to fix the number of candidates in the link prediction tasks at all snapshots, for example, by incorporating all entities ever occurring in the evolving KG. Accordingly, head and tail entities of test examples are also substituted with entities that have been removed or not emerged so far. This option's advantage is that the link prediction task's evaluation results are comparable between snapshots compared to the first variant. However, this would lead to problems in applying our designed static learning variant as it only learns embeddings for entities and relations present at a snapshot and do not capture unseen entities.

We wanted to keep our framework compatible with the defined training procedures. So we decided to implement the first approach, i.e. to solely include present entities in constructing negative rank candidates for link prediction.

Concerning the evaluation metrics, we performed the link prediction task on each snapshot to measure the Mean Rank and Hits@10 score (raw and filtered). Finally, we macro-averaged the results for each metric to summarize a model's performance throughout the KG evolution.

Triple Classification Following the evaluation protocol of Han et al. (2018), a model classifies a set of positive and negative examples with respect to a classifier T in this task. Based on the learned KG embeddings the model calculates the energy score $f_r(h,t)$ of a triple (h,r,t) and classifies it as true in case $f_r(h,t) > T$ applies, otherwise as false.

Here, we use triples from a snapshot's test dataset and create a negative counterpart by either corrupting their head or tail entities with a random entity from the KG. It is essential to prevent the gathering of negative examples, which once were valid in the KG evolution. Static models are learned from scratch, and would probably have no problems to forget such facts in graph evolution. However, since we do not know whether an incremental model can forget deleted facts at the time of testing, the static model may have an advantage in this aspect. We exclude such cases in this evaluation task to ensure a fair evaluation of static and incremental methods. However, we cover them for an extensive test frame tailored for incremental methods.

Concerning the evaluation metric, we measure the model's accuracy on the described test examples. Given the frequencies of different classification outcomes, i.e. true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), the accuracy is calculated with the formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

After measuring the accuracy scores for all snapshots, we calculate the micro and macro-average accuracy of a model to summarize its performance for the KG evolution.

The macro-averaged accuracy (MAA) equally weighs the received accuracy scores and describes the mean accuracy (Figueiredo et al., 2011). It is calculated by using the formula:

$$MAA = \frac{\sum_{i=1}^{n}(Accuracy_{(i)})}{n},$$

where $Accuracy_{(i)}$ is the accuracy measured at snapshot i and n is the number of snapshots.

The micro-averaged accuracy MIA in turn treats every classification decision made throughout the KG evolution equally (Figueiredo et al., 2011). It is determined as follows:

$$MIA = \frac{\sum_{i=1}^{n}(TP_{(i)} + TN_{(i)})}{\sum_{i=1}^{n}(TP_{(i)} + TN_{(i)} + FP_{(i)} + FN_{(i)})},$$

where i represents the index of the respective snapshot.

By reflecting on the formula, MIA emphasizes results obtained in classification tasks with a higher number of samples. Given the example that at three snapshots classification tasks are performed, where the first two snapshots each contain 100 examples and the third snapshot contains 1000 examples, the micro-averaging accuracy would favor classification results measured in the last snapshot. By combining the MIA with the MAA, we balance out such situations and provide a more comprehensive view of the model's performance.

Negative Triple Classification for Incremental Embedding Models Naturally, KGs like YAGO and Wikidata evolve, meaning that their knowledge base change over time. New facts can be added to the graph and last there. However, if they are no longer considered valid, they can be deleted from the graph. All in all, with every triple operation, the truth value of the related triple changes. By investigating the Wikidata KG history, we found cases of recurring insertions and deletions of the same triples, which caused their truth values to oscillate over time.

That said, an incremental model should optimize its embeddings by assigning plausibilities to triples, with respect to their current truth values. We adopted this scenario in our evaluation framework and designed an evaluation task tailored for the comprehensive evaluation of incrementally learned embeddings. Based on the Wikidata history examination, we defined four classes that categorize triples according to their operation history and current truth value. These are:

1. Inserted triples
2. Deleted triples
3. Positive oscillated triples
4. Negative oscillated triples

The first category consists of newly inserted triples, i.e. facts that emerged in the KG for the first time. The second category describes triples that have been deleted after their first insertion. Precisely, they are related to a single insert operation and a single delete operation. The third category collects facts that have been reinserted after their deletion, i.e. they faced at least two insertions and an interim deletion. The fourth category, in turn, explains triples for which at least two insertions and two deletions occurred.

With any operation affecting a triple's truth value, a triple can alternate between these categories, as shown in the state diagram in figure 10. If a triple appears in the graph for the first time, we assign it to the first category. When the triple is removed for the first time from the KG, we collect it for the second category. It prevails for the remaining snapshots unless there is no operation left in the KG evolution changing its truth value again. However, as soon as the fact enters the graph again, it is assigned to the third category. With additional insertions or deletions, a fact alternates between the third and fourth categories.

We observe the operation histories of facts retrospectively at each snapshot and attach them accordingly to these categories. Rather than only considering triples of the test dataset, we cover the entire KG, i.e. by also involving training and validation examples.

Since we restrict updates to one operation per triple (see 5.3.1), the truth value of a triple can only change per snapshot. Implicitly, instances of the second category are only available from the second snapshot. Likewise, third and fourth class instances can only be captured from the third, respectively, the fourth snapshot.

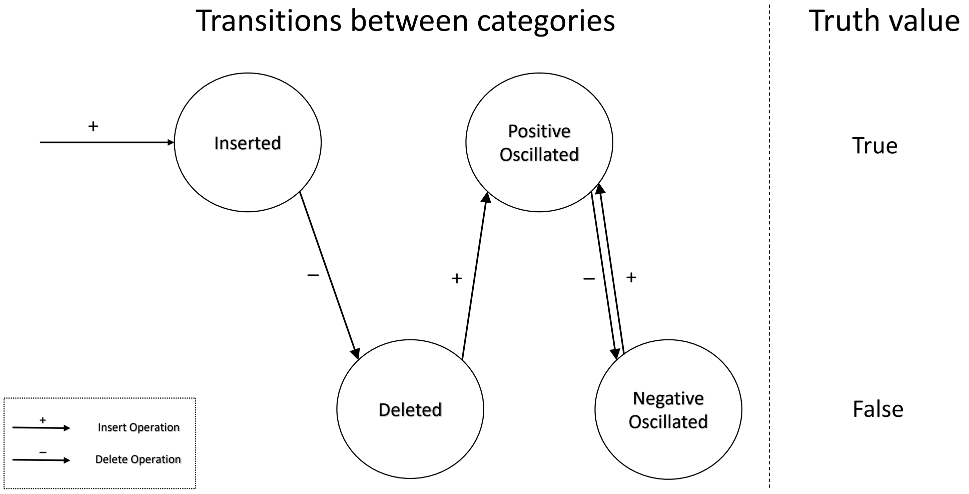


Figure 10: State diagram depicting paths between the four categories.

Since triples of the first category are already part of the snapshots' test datasets and therefore used in the general triple classification task, we exclude them from this task. However, we have captured them in our concept description for the sake of clarity. Characteristically, we designate this evaluation task as *negative triple classification*, since we classify facts which once have been deleted from the KG.

For the negative triple classification we use the threshold T determined in the basic triple classification procedure. Then we exploit the incrementally learned embeddings of a model to predict the plausibility $f_r(h,t)$ for a test example (h,r,t). If $f_r(h,t) > T$ applies, (h,r,t) is classified as true otherwise as false.

Based on the classification outcomes, we measure the accuracy $Accuracy_{(c)}$ for all categories separately, following the formula:

$$Accuracy_{(c)} = \frac{TP_{(c)} + TN_{(c)}}{TP_{(c)} + TN_{(c)} + FP_{(c)} + FN_{(c)}},$$

where $TP_{(c)}$,$TN_{(c)}$,$FP_{(c)}$ and $FN_{(c)}$ are outcomes for category c.

Note that a category consists of either true or false triples. The second and fourth categories, for instance, collect exclusively negative triples. As a result, true positives and false negatives cannot occur in these cases. So, the accuracy expresses the measure of specificity, i.e.

$$Accuracy_{(2/4)} = Specificity_{(2/4)} = \frac{TN_{(2/4)}}{TN_{(2/4)} + FP_{(2/4)}}.$$

Conversely, the third category, only capturing positive examples, eliminates true negatives and false positives. In this case, the accuracy score expresses the recall, defined as:

$$Accuracy_{(3)} = Recall_{(3)} = \frac{TP_{(3)}}{TP_{(3)} + FN_{(3)}}.$$

However, for the sake of simplicity, we refer to the accuracy score for the metrics of all classes. Finally, we summarize the models' performance for each category by micro-averaging and macro-averaging the multiple accuracy results. The macro-averaged accuracy $MAA_{(c)}$ for category c is composed as follows:

$$MAA_{(c)} = \frac{\sum_{i=1}^{n}(Accuracy_{(ic)})}{n},$$

where $Accuracy_{(ic)}$ is the accuracy measured for category c at snapshot i and n is the number of snapshots.

The micro-averaged accuracy $MIA_{(c)}$ in contrast is described as:

$$MIA_{(c)} = \frac{\sum_{i=1}^{n}(TP_{(ic)} + TN_{(ic)})}{\sum_{i=1}^{n}(TP_{(ic)} + TN_{(ic)} + FP_{(ic)} + FN_{(ic)})},$$

where $TP_{(ic)}$, $TN_{(ic)}$, $FP_{(ic)}$ and $FN_{(ic)}$ are classification outcomes obtained for class c at snapshot i.

6. WikidataEvolve - A Benchmark Extracted from Wikidata for Evaluating Incremental Knowledge Graph Embeddings

In section 5, we discussed why it is reasonable to ground our evaluation framework on an evolving KG represented by a time stream of triple operations. Since no such dataset has been published in the literature yet, we had to compile it ourselves.

There are a bunch of publicly available KGs from which we could extract data. However, most of them are inappropriate for our purposes. They release their data in chronological versions instead of documenting the change history of the graph. Hence, we chose Wikidata as a data source. It is one of the most famous representatives of KGs. In contrast to other KGs, it publishes its complete revision history up to a point in time at `https://dumps.wikimedia.org/wikidatawiki/`.

In this section, we first outline the methodology we followed in the data extraction process. Here, we traversed the complete history of the KG to transform it into an evolving KG, represented by a sequence of 9 million triple operations. We call this sequence *Wikidata9M*[10].

In the next step, starting from the Wikidata9M dataset, we compiled a new benchmark called *WikidataEvolve*. It is aligned with the methodology of our evaluation framework described in section 5. Consequently, it contributes to the domain of (incremental) KG embedding by providing a first benchmark tailored for the thorough assessment of incremental KG embedding models and their comparison with static and pseudo-incremental embedding methods throughout an evolving KG.

6.1. Extracting Wikidata9M - a high-quality Triple Operations Stream from Wikidata

To compile Wikidata9M, we downloaded the complete history of Wikidata accumulated until May 1, 2020. The Wikidata history is divided into downloadable dumps. Each dump contains the complete revision histories of Wikidata Items, i.e., KG entities. In turn, each revision states a modification of the graph and contains a timestamp indicating when the change took place. Further, it contains a list of claims. Claims represent semantic relations between Wikidata items, i.e., triples in which the respective item is the subject[11]. The list of claims attached to a revision, specifically documents which triples (of the respective entity) still existed after the KG change.

Based on these lists, we synthesized triple operations. Thereby, we iterated the chronologically ordered revisions. By determining the differences between claim lists of two consecutive revisions, we detected which triples have been added or deleted with a revision. Finally, we encoded the extracted operations by collecting the involved triple, the underlying operation type, i.e., insertion or deletion, and its timestamp. The triple operation

(*“Cristiano Ronaldo”, “Member of sports team”, “Real Madrid“, “+”, “2018-11-28 07:56:51”*),

for instance, states that the triple (*“Cristiano Ronaldo”, “Member of sports team”, “Real Madrid“*) emerged in the KG at the specified timestamp. In contrast, the triple operation

(*“Cristiano Ronaldo”, “Member of sports team”, “Real Madrid“, “–”, “2020-01-09 02:30:07”*)

documents the triple’s deletion. Note, that we do not encode the names of entities and relations in a triple operation but their Wikidata identifiers.

Each Wikidata Item possesses a web page, primarily displaying all its relations in the Wikidata knowledge base in which the entity at hand participates as the subject.

[10] Published at `https://github.com/rlafraie/OpenKE/blob/OpenKE-PyTorch/benchmarks/Wikidata/final_extracted_data/Wikidata9M.txt.bz2`.

[11] `https://www.wikidata.org/wiki/Wikidata:Glossary` last retrieved June 16, 2020.

As we noticed during the extraction process, the Wikidata history tracks redirects between entity web pages. Redirects take the visitor of an URL promptly to a different URL (Kwon et al., 2017). Like Wikidata points out, redirects are reasonable for items that refer to the same real-world object but have been maintained in parallel by the Wikidata community[12]. Such duplicates are resolved by merging them and their contained information into a newly created item (and related web page). Given that these duplicates emerged as objects in relationships with other items before, their deletion would lead to inconsistencies in the Wikidata graph. By assigning redirects to the duplicates' web pages leading to the newly created item, Wikidata avoids such inconsistencies.

In the extraction process, we dealt with such cases in two steps: First, we substituted redirected entities in our set of extracted triple operations with their target entities. As a result, we obtained numerous duplicates characterized by insertions or deletions that successively add or remove the same triple to the KG. For instance, assume that the triple (*"Angela Merkel"*, *"born in"*, *"Eimsbüttel"*) was added twice with two consecutive insert operations. Second, we solved these inconsistencies by only keeping the operation with the earliest timestamp.

In the next step, we removed triple operations with reflexive triples. These are facts in which an entity acts as the subject and object. Lastly, we applied a filter defined by Lacroix et al. (2020). In this work, the authors extracted a static dataset from the Wikidata knowledge base. As they state, many items in the Wikidata KG represent proteins, categories, scholarly articles, and numerical values that are unsuitable for the application in a link prediction or triple classification task (Lacroix et al., 2020). For this reason, they conducted a filtering process and published a list of appropriate KG elements available at `https://github.com/facebookresearch/tkbc`, in which such cases are excluded. Similarly, we used their artifact as a filter and obtained a sequence of 9 million triple operations.

6.2. From a Triple Operations Stream to an Evolving Knowledge Graph Benchmark

To support the introduced evaluation framework, we compiled the WikidataEvolve benchmark and closely oriented the construction process on the structures described in section 5.

WikidataEvolve comprises three datasets – tailored for the different model types and their learning procedures. That said, we transformed Wikidata9M to supply static models with the snapshots, pseudo-incremental models with the increments and incremental models with the updates of the evolving KG.

Since we began the compilation process from a sequence of triple operations, we had to figure out how to separate the evolving KG data into training, validation, and test data. The division procedure in a static KG environment is quite simple: A fixed KG is randomly split into a training, validation, and test dataset (Bordes

[12] `https://www.wikidata.org/wiki/Help:Redirects` last retrieved June 20, 2020.

et al., 2013). A static model is then learned using the training and validation dataset and applied to the test dataset. In contrast, our case was more complicated as the compilation process had to carry out two divisions. On the one hand, a division into time intervals had to be conducted. On the other hand, training and evaluation data had to be synthesized. Several alternative courses of action to create snapshots, updates, increments, plus respective training and evaluation datasets opened up.

According to our evaluation framework's design, we intuitively considered a course of action in which we first divide Wikidata9M into n updates. In the next step, we calculated the resulting snapshots. The idea was to perform the train, validation, and test split on each snapshot. Doing so, we would have achieved to synthesize data compatible with the application of static models.

Before proceeding to analyze how we could derive the updates and increments from the training datasets to feed incremental, respectively, pseudo-incremental learning procedures, we identified fundamental flaws in this approach.

As we would repeatedly perform a random split on each snapshot, triples may jump between the training, validation, and test datasets. Given a triple (h, r, t), which exists at all snapshots, it could first emerge as a training example at snapshot t. However, after performing another split at snapshot $t + 1$, the triple could be attached to the validation or test datasets. An incremental model would learn the triple in t but is applied to it in the evaluation procedure at snapshot $t + 1$, leading to distorted performance results.

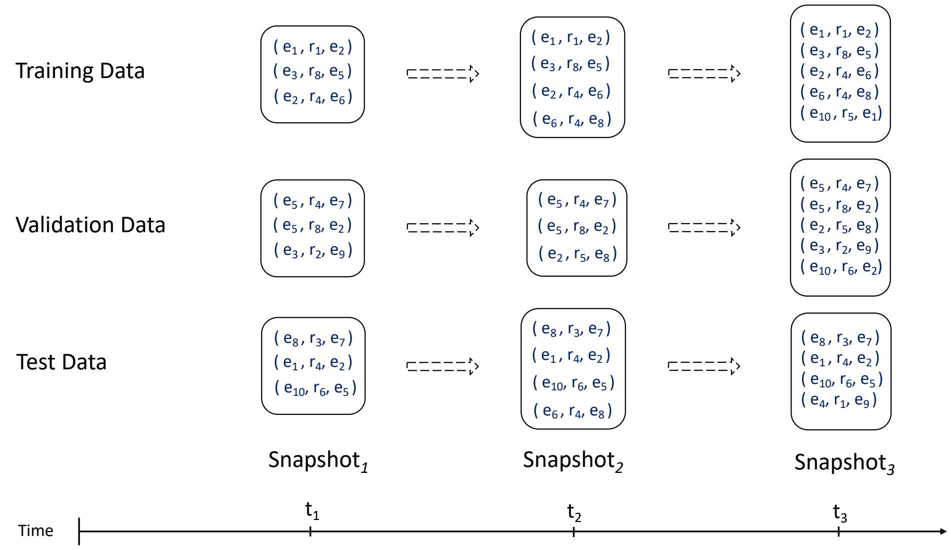

Training Data
Validation Data
Test Data
(e_1, r_1, e_2) (e_3, r_8, e_5) (e_2, r_4, e_6)
(e_1, r_1, e_2) (e_3, r_8, e_5) (e_2, r_4, e_6) (e_6, r_4, e_8)
(e_1, r_1, e_2) (e_3, r_8, e_5) (e_2, r_4, e_6) (e_6, r_4, e_8) (e_{10}, r_5, e_1)
(e_5, r_4, e_7) (e_5, r_8, e_2) (e_3, r_2, e_9)
(e_5, r_4, e_7) (e_5, r_8, e_2) (e_2, r_5, e_8)
(e_5, r_4, e_7) (e_5, r_8, e_2) (e_2, r_5, e_8) (e_3, r_2, e_9) (e_{10}, r_6, e_2)
(e_8, r_3, e_7) (e_1, r_4, e_2) (e_{10}, r_6, e_5)
(e_8, r_3, e_7) (e_1, r_4, e_2) (e_{10}, r_6, e_5) (e_6, r_4, e_8)
(e_8, r_3, e_7) (e_1, r_4, e_2) (e_{10}, r_6, e_5) (e_4, r_1, e_9)
Snapshot$_1$
Snapshot$_2$
Snapshot$_3$
Time
t_1
t_2
t_3

Figure 11: Organically developing training, validation and test data throughout an evolving KG.

Therefore, we rejected this approach to aim for a permanent separation of training and test data. Figure 11 illustrates our objective. Here the training, validation, and test data grow organically along with the entire graph evolution.

To comply with this scheme, we developed a procedure that takes the graph's evolution in the construction of the data sets into account: First, we divided Wikidata9M into four updates with an equal number of triple operations. Since we evaluated at each snapshot, an increasing number of updates and respective snapshots would led to an increasing number of results. Therefore we intended to keep a comprehensive overview of the results. On the other hand, we granted each triple one operation per interval, as we have already explained. Thus, the truth value of a triple could only change per snapshot. Inherently for the task of negative triple classification, a triple needs a minimum of four transitions of its truth value to reach category four. Hence it is not reasonable to set the number of snapshots too low. We consider our decision of four updates and respective snapshots to be appropriate because we allowed the KG triples enough time to spread over the categories.

Afterwards, we initialized three sets used to cross-temporally collect training, validation, and test examples. Then we iterated the sequence of updates by following the procedure sketched in algorithm 2. We first determined which triples have been inserted and deleted with the update (lines 3 - 9). For inserted triples, we first checked if they have already been part of the cross-temporally training, validation, or test dataset to attach them respectively (lines 10 - 19). In this way, we prevented the mixture of training and test examples. On the remaining triples, which logically emerged in the KG for the first time, we performed a train, validation, and test split to assign them to their corresponding dataset (lines 20 - 23). Concerning the split proportions, we drew on the benchmarks from section 2.5 and labeled 90% of newly inserted triples as training examples and 5% each as validation and test examples. Next, we handled deleted triples by deleting them from the corresponding datasets (lines 24 - 31).

Lastly, we stored these datasets' current states, representing a snapshot of the KG structured into training, validation, and test data (lines 32 - 34). After such an iteration, we preserved the cross-temporal datasets and applied the same procedure for the remaining updates.

Algorithmus 2 : Compilation of training, validation, and test datasets from an evolving KG.

```
Input   : Sequence of KG updates $U$
Output  : Training, validation and test dataset $Train_i, Valid_i, Test_i$ for snapshot $i$
$K_{train}, K_{valid}, K_{test}$   ? // Sets for cross-temporal collection of training, valid and test triples
for $U_i$ 2 $U$ do
    /* Determine inserted and deleted triples in $U_i$ */
    $I_i$   ? // Set to track inserted triples in Update $U_i$
    $D_i$   ? // Set to track deleted triples in Update $U_i$
    for $operation$ 2 $U_i$ do
        if $operation.type == insert$ then
            $I_i$   $I_i$ [ f$operation.triple$g
        if $operation.type == delete$ then
            $D_i$   $D_i$ [ f$operation.triple$g
    /* If triple existed once in the KG it returns to its corresponding set */
    for $triple$ 2 $I_i$ do
        if $triple$ was once in $K_{train}$ then
            $K_{train}$   $K_{train}$ [ f$triple$g
            $I_i$   $I_i$ n f$triple$g
        if $triple$ was once in $K_{valid}$ then
            $K_{valid}$   $K_{valid}$ [ f$triple$g
            $I_i$   $I_i$ n f$triple$g
        if $triple$ was once in $K_{test}$ then
            $K_{test}$   $K_{test}$ [ f$triple$g
            $I_i$   $I_i$ n f$triple$g
    /* Perform train, validation, test split on remaining newly inserted triples */
    $I_{train}, I_{valid}, I_{test}$   $I_i$
    /* Attach splits to cross-temporal sets */
    $K_{train}$   $K_{train}$ [ $I_{train}$
    $K_{valid}$   $K_{valid}$ [ $I_{valid}$
    $K_{test}$   $K_{test}$ [ $I_{test}$
    /* Remove deleted triples in $D_i$ from the sets */
    for $triple$ 2 $D_i$ do
        if $triple$ 2 $K_{train}$ then
            $K_{train}$   $K_{train}$ n f$triple$g
        else if $triple$ 2 $K_{valid}$ then
            $K_{valid}$   $K_{valid}$ n f$triple$g
        else if $triple$ 2 $K_{test}$ then
            $K_{test}$   $K_{test}$ n f$triple$g
        $D_i$   $D_i$ n f$triple$g
    /* Store training, validation, test datasets for snapshot $i$ */
    $Train_i$   $K_{train}$
    $Valid_i$   $K_{valid}$
    $Test_i$   $K_{test}$
```

After finishing the procedure, we obtained four snapshots, each structured into training, validation, and test snapshots. Moreover, the datasets were permanently disjunctive and grew organically over time, as illustrated in figure 11.

Nevertheless, these datasets were only suitable for executing static embedding models in our evaluation framework. Therefore we proceeded and processed the training snapshots to derive updates and increments for the training of incremental and pseudo-incremental methods.

Algorithmus 3 : Compilation of Training Updates and Increments.

```
  Input  : Set of chronologically ordered training snapshots $T$, KG update $U_i$
  Output : Training update $B_i$, training increment $I_i$ for training snapshot $T_i$
1 for $T_i$ 2 $T$ do
      /* Identify inserted and deleted triples                          */
2     $I_i$    $T_i$ n $T_{i}$ 1
3     $D_i$    $T_i$ 1 n $T_i$
      /* Compile training update $B_i$                                  */
4     $B_i$    ?
5     for $operation$ 2 $U_i$ do
6         if $operation.triple$ 2 $I_i$ then
7             $B_i$    $B_i$ [ f$operation$g
8         if $operation.triple$ 2 $D_i$ then
9             $B_i$    $B_i$ [ f$operation$g
```

As drawn in the algorithm 3, we compared two consecutive training snapshots to determine which triples have been added or deleted in between (lines 2 and 3). Added triples have been summarized into an increment. Next, we matched the added and deleted triples to their operations in the respective KG update (lines 5-9). By chronologically sorting these insertions and deletions, we compiled a training update.

Lastly, we gathered examples for the negative triple classification task to complete the WikidataEvolve benchmark. In this process, we traversed the KG updates, which served as input to create the training and evaluation datasets (see algorithm 2). While tracking the operation history for all triples, we assigned them to the four test categories at each snapshot. Here, we stored the examples for every category into an individual dataset.

In line with our devised evaluation framework, WikidataEvolve supplies the same training data in different formats compatible with the different model types. So, we supplied each model type with a different dataset. Further, for all types the same evaluation data was attached. Only for incremental models, we included examples gathered for negative triple classification.

6.3. Dataset Statistics

We summarized statistics about the WikidataEvolve dataset in table 5. In the first two rows, we provided the average number of training triples contained in the different training snapshots (#Training Triples (Snapshots)) and increments (#Training Triples (Increments)). The next three rows depict the average number of training operations (#Training Operations (Updates)), and related triple insertions and deletions for the various updates. Besides, we recorded the average number of triples for the multiple validation (#Valid Triples) and test datasets (#Test Triples) as well as the number of present entities (#Entities) and relations (#Relations) at the different snapshots.

To give a complete picture, we extended the table by the standard deviation and the minimum and maximum values. They indicate how individual frequencies scatter throughout the evolving KG.

On average we tracked $405,218$ entities and 324 relations for the different snapshots, $4,458,796$ triples in the training snapshots, $1,745,577$ triples in their increments and $1,815,377$ triple operations in the related updates. Remarkably, the number of insert operations is significantly higher, likely because in the described algorithm in section 6.2 we only add triples to the KG until the first snapshot. For the validation and test datasets, we counted $247,696$ and respectively $247,678$ triples on average.

The KG continuously grows over time, meaning that the lowest frequencies are counted in the first update and increase steadily until the fourth snapshot to reach their maximum.

	Avg.	Std. Dev.	Min.	Max.
#Training Triples (Snapshots)	4,458,796	2,072,112	1,949,569	6,703,109
#Training Triples (Increments)	1,745,577	238,812	1,399,986	1,949,569
#Training Operations (Updates)	1,815,377	176,091	1,557,798	1,949,569
#Insertions	1,745,577	238,812	1,399,986	1,949,569
#Deletions	69,800	67,222	0	157,812
#Valid Triples	247,696	115,132	108,309	372,450
#Test Triples	247,678	115,072	108,310	372,253
#Entities	405,218	30,926	361,696	432,353
#Relations	324	84	216	407

Table 5: Details of WikidataEvolve.

In table 6, we documented the number of test examples gathered for the categories of negative triple classification for each snapshot. As we explained in section 5.3.2, we restrict a KG update to one operation per triple. In this way, the empty cells in table 6 are explained. We can only collect examples for deleted triples (category 2) from the second snapshot, for positive oscillated triples (category 3) from the third snapshot and negative oscillated triples (category 4) from the fourth snapshot.

Notably, the second category counts the most examples in all snapshots. It starts at approximately $39k$ samples in the second snapshot, increases its size more than seven times up to the fourth snapshot. For the third category, we could gather $2,119$ examples in the third snapshot and nearly $10k$ in the fourth snapshot. The fourth category has the fewest instances as we could only gather 437 examples at the fourth snapshot.

	Snapshot 1	Snapshot 2	Snapshot 3	Snapshot 4
Category 2 (Deleted Triples)	-	45,663	132,818	299,574
Category 3 (Pos. Oscillated Triples)	-	-	2,119	9,886
Category 4 (Neg. Oscillated Triples)	-	-	-	437

Table 6: Snapshot frequencies for the three test categories of negative triple classification. Since category 1 is not considered in this evaluation task, we excluded related frequencies.

Compared with benchmark datasets used for static KG embedding introduced in 2.5, the size of the WikidataEvolve dataset is much larger. As we count approximately 4.5 million training examples for the different snapshots, the sizes of the training datasets of the other benchmark datasets range from $86,835$ to about 1 million training triples.

Bordes et al. (2013) extracted data from the KGs of Freebase and WordNet to form the benchmarks of FB15k and WN18. Thereby, the authors filtered entities and relation appearing in at least 100 triples, to provide models with an appropriate amount of relational information to learn representations for KG elements (Bordes et al., 2013).

Inspired by Bordes et al. (2013), we transferred the author's approach to our setting and conducted snapshot-based filtering of entities and relations. We targeted entities and relations complying with a minimum volume of triple appearances T_{ent} and T_{rel} after they emerged in the evolving KG. Thus, before configuring the datasets according to algorithm 2 we adopted the KG updates and determined their resulting snapshots. Next, we determined the frequencies for all entities and relations at all snapshots and filtered KG elements appearing more frequently than T_{ent} and respectively T_{rel}. Lastly, we iterated the KG updates to collect the triple operations of the filtered KG elements.

On the one hand, we wanted to select the thresholds such that for entities and relations at all snapshots, an appropriate amount of relational information is provided. However, most important was to still collect a sufficient number of samples

	Avg.	Std. Dev.	Min.	Max.
#Training Triples (Snapshots)	487,875	154,416	304,647	667,224
#Training Triples (Increments)	175,642	86,650	117,788	304,647
#Training Operations (Updates)	184,478	81,140	127,991	304,647
#Insertions	175,642	86,650	117,788	304,647
#Deletions	8,836	7,409	0	17,902
#Valid Triples	27,103	8,563	16,925	37,038
#Test Triples	27,075	8,537	16,925	36,977
#Entities	47,161	10,136	35,804	59,391
#Relations	119	28	87	150

Table 7: Details of the filtered version of WikidataEvolve.

to feed the task of negative triple classification. After some experimentation, we considered a value for $T_{ent} = 15$ and $T_{ent} = 70$ to be appropriate.

Table 7 depicts details of the filtered version of WikidataEvolve. By comparing the statistics of the filtered version and the raw version (see 5), it is noticeable that the size of the filtered version is significantly smaller in every aspect. The average number of training triples for the different training snapshots decreased from approximately 4.5 million in the raw version to approximately $487k$ in the filtered version. Accordingly, the size of the training updates, increments and evaluation datasets decreased. Concerning entities, we counted approximately $405k$ in the raw version and around $47k$ in the filtered version on average for the different snapshots. The number of relations has decreased from 324 to 119. Considering the statistics, the filtered version of WikidataEvolve aligns with the benchmark datasets presented in 2.5. For detailed statistics about both versions see tables 12 and 13 in appendix A.

Logically, the filtering procedure has reduced the number of test examples used for the negative triple classification. However, we still consider the frequencies of the three categories shown in table 8 to be sufficient to assess an incremental model in its capability of learning and unlearning facts. Just as in the raw variant, most instances are assigned to the second group of deleted triples. They range from almost $8k$ examples in the second to $37k$ instances in the last snapshot. The third category counts 495 samples in the third snapshot and approximately $2k$ in the last one. Like the raw version, with a count of 135 samples, the fourth group has the fewest instances.

	Snapshot 1	Snapshot 2	Snapshot 3	Snapshot 4
Category 2 (Deleted Triples)	-	7,999	18,931	37,020
Category 3 (Pos. Oscillated Triples)	-	-	495	2,087
Category 4 (Neg. Oscillated Triples)	-	-	-	135

Table 8: Snapshot frequencies for the three test categories in the filtered WikidataEvolve version.

The last two categories have the fewest samples in both WikidataEvolve versions. Nevertheless, comparing both versions' statistics, the third and fourth categories shrunk nearly by a factor of 4, whereas the training and evaluation datasets of the raw version decreased by a factor of 10. So, we reached an acceptable trade-off between preserving a sufficient number of samples for negative triple classification and eliminating rarely occurring KG elements. We published the filtered version of WikidataEvolve at `https://github.com/rlafraie/OpenKE/tree/OpenKE-PyTorch/benchmarks/Wikidata/WikidataEvolve`.

7. Critical Discussion

This section conclusively discusses our created artifacts, including the devised evaluations framework, the WikidataEvolve benchmark, and the Wikidata9M sequence of triple operations. As we outlined at the beginning of section 5, the literature offers no uniform procedure for the evaluation of incremental models. Further, existing evaluation approaches fail in addressing the specificity of incremental methods in their test settings.

First of all, the combination of WikidataEvolve and the evaluation framework allows to evaluate and compare different types of embedding models over time. These include both static and incremental methods. For static KG embedding methods, we have defined two learning variants in our framework. While a static learning procedure learns the entire KG repeatedly from scratch, the pseudo-incremental variant only learns added facts, i.e. KG increments. With the pseudo-incremental learning variant, we provide an interpolation between static and incremental KG embedding models. The literature suggests that if a KG was modified, static models have to learn the complete graph from scratch (Jia et al., 2016; Wu et al., 2019). Due to the new learning variant, this statement can be corroborated now. For this purpose, a particular model must be applied to a static and pseudo-incremental training procedure using our evaluation framework. By analyzing the evaluation results, it can finally be determined if and to what extent the pseudo-incremental procedure performed worst than the static variant. Furthermore, the results of an incremental model can be included for an overall comparison of the different model types in order to examine which model achieved the best evaluation results and how stable their embeddings performed over time.

For practical purposes, decisions can be consulted accordingly. For example, one might consider to apply a static model to a pseudo-incremental learning procedure if it achieves a reasonable trade-off between the accuracy of static methods and the efficiency of incremental methods. Alternatively, one could prefer using an incremental method to achieve better efficiency than a pseudo-incremental variant and a higher accuracy than static models.

More importantly, our evaluation framework is tailored for the sound evaluation of incremental models. In contrast to static methods, incremental methods maintain and optimize their embeddings in the KG development. Indeed, a reasonable evaluation of incremental methods should emphasize the temporal change of the KG. For this reason, we developed the task of negative triple classification. Here, we do not exclusively draw on triples labeled as test examples. Instead, we observe the operation history of all triples in the evolution of the KG. Accordingly, we subsume triples by observing their operation history into 3 test categories. The categories cover deleted triples, and triples that have been repeatedly deleted and inserted during the evolution. Based on these examples, we test an incremental model's ability to predict their current truth values throughout the KG evolution. By differentiating the test results achieved in the three categories, we can assess whether a model performed better in learning or unlearning facts to consult its strengths and weaknesses. All in all, we developed a test framework that enables to analyze and compare incremental models in-depth and therefore constitutes a first benchmark in the domain of incremental KG embedding.

Reflecting on the design of our framework, it is noticeable that its elements are flexibly configurable. On the one hand, we retrieve data to support our framework from the Wikidata9M dataset. Based on this time stream of triple operations, the number of KG updates and resulting snapshots in our framework are freely configurable. For our purposes, we have divided the time stream into four updates to compile our WikidataEvolve dataset. The higher the number of updates is selected, the less triple operations each update contains logically. Hence, the frequency of triple operations per update is controllable for individual evaluation purposes.

Besides, the general approach of learning and evaluating models up to a snapshot ensures comparability of performance results between models. Regarding the evaluation tasks, we apply link prediction and triple classification in addition to negative triple classification. However, suppose further benchmark tasks for evaluating static and incremental KG embedding models emerge in the future. In that case, they can be seamlessly integrated into our framework.

Nevertheless, our framework is not free of limitations. In section 5.1 we criticized that Wu et al. (2019) fix the test datasets in their evaluation setting throughout an evolving KG. We consider such a procedure to be inappropriate, to simulate an authentic evolving KG. The test data should instead evolve in the same way like the training data of a KG as we approached in the compilation of WikidataEvolve. However, this has the drawback that the results between the snapshots are not fully comparable. In this respect, the temporal development of test results in our framework must be treated with caution.

Considering the negative triple classification task, we sampled not only triples from the test data set of the evolving KG but from the entire KG, including the training and validation data. Such an approach contradicts the convention in machine learning, to separate training and test data to evaluate predictive algorithms (Reitermanova, 2010).

Looking at the task description of negative triple classification, we demand a model to predict the alternating truth values of facts at a given snapshot correctly. Due to our framework's described methodology, an incremental model only receives triple operations related to the training data of the KG. So, it is not informed about when a test triple is deleted from the KG. For this reason, we consider it more difficult for the model to predict the current truth values of test triples correctly. Therefore we consciously included training triples as test cases to check whether the model at first is capable of unlearning deleted training facts.

Ultimately, our contribution to the research direction of (incremental) KG embedding is two-fold: (i) We created an evaluation framework that can be used to assess incremental KG embeddings in-depth and compare them to other types of KG embedding models in the context of an evolving KG. (ii) We compiled the WikidataEvolve dataset, which represents an authentic evolving KG and forms the first benchmark for incremental KG embedding.

8. A Comparative Study of Incremental and Static Knowledge Graph Embedding Approaches

In the scope of this work, we evaluated a selection of KG models in an experiment by applying our devised evaluation framework. The corresponding source code has been published at `https://github.com/rlafraie/OpenKE/tree/OpenKE-PyTorch/experiments/`. In this section, we first describe the experimental setup and afterwards present and discuss the experimental results.

The training and evaluation procedures have been executed on a Ryzen Threadripper 1950X CPU, with 128GB RAM and 2 NVIDIA Titan RTX GPUs.

8.1. Experimental Setup

In this experiment, we exploited the filtered version of the WikidataEvolve benchmark created in this work (see section 6.3 for dataset statistics). It represents an evolving KG and provides training data for the different model types presented in section 5.3. Basically, we trained models on this training data and evaluated them at each snapshot.

As a static KG embedding method, we applied TransE and subjected it to both a static learning procedure and a pseudo-incremental learning procedure. Here, we utilized the implementations of the OpenKE framework. As an incremental KG embedding method, in turn, we selected the PuTransE method implemented by ourselves.

Regarding the hyperparameters, we performed a grid search for TransE on the first training snapshot. Here, the model performed best with a L1 norm, a margin of 5, a learning rate of 0.1 and a number of 100 dimensions. The determined hyperparameters were then adopted throughout the evolving KG for both the static and the pseudo-incremental learning procedure.

For PuTransE, we adopted the ranges from the evaluation setting described in section 4.7.2. Here, we used the L1 norm, a randomly selected margin γ among $[1, 4]$, the learning rate from the range $[0.001, 0.1]$ and the number of triples per embedding space among $[500, 2000]$. The number of dimensions was set to 20. The limit of training epochs per universe was chosen randomly from the range $[500, 2000]$.

We conducted early stopping on the validation data sets in each training phase by measuring the Hits@10 score on the validation dataset. For TransE, we defined a limit of 1000 epochs per training phase in both learning variants. For PuTransE, we restricted the number of embedding spaces to 6000. Concerning the treatment of missing global energy scores, we employed the technique described in 4.4.1. Accordingly, we equated the energy scores of triples for which no global prediction could be calculated in the evaluation process to the negative infinity score.

On the subject of evaluation, we tested all models in link prediction and triple classification tasks. For a detailed description of the tasks, we refer to section 2.4 and our specifications outlined in 5.3.2. For both tasks, we utilized the implemented procedures of the OpenKE framework. Concerning the latter task, the developers determined the threshold T directly on the test dataset by calculating a model's maximum accuracy. To measure a more realistic value for accuracy, the classifier should be first obtained on the validation dataset and afterwards applied to classify the test data. Nevertheless, due to lack of time, we could not adjust the OpenKE procedures accordingly and relied on their scheme. After a triple classification task finished, we adopted the threshold T and evaluated an incremental model, i.e. PuTransE, in the self-developed negative triple classification task.

Besides, we had to accelerate our experiment's evaluation procedure due to the mentioned lack of time. For this purpose, we did not use the complete test datasets of WikidataEvolve. Instead, we sampled 6000 test triples from the snapshots' test datasets. Note that our limitation only relates to triple classification and link prediction. For negative triple classification, conversely, we have considered all examples withhold for the three categories at each snapshot.

8.2. Experimental Results

In the following section, we present the results of our experiment. We will discuss them for the respective tasks in individual sections. In the last section, we summarize them and reflect on the whole experiment.

8.2.1. Link Prediction

Table 9 presents the results of the three models achieved in the link prediction tasks throughout WikidataEvolve. While the upper half of the table lists the Mean Rank values, the lower half of the table displays the Hits@10 scores. In turn, the columns illustrate the results obtained at the different snapshots in the raw and filtered setting. The last column finally summarizes the performance results for the entire evolving KG by measuring their macro-averages.

	Snapshot 1		Snapshot 2		Snapshot 3		Snapshot 4		Macro average	
	raw	*filtered*	*raw*	*filtered*	*raw*	*filtered*	*raw*	*filtered*	*raw*	*filtered*
Mean Rank										
TransE (static)	1,374	415	1,133	**438**	**1,131**	**428**	**1,166**	**453**	**1,201**	**434**
TransE (pseudo-incremental)	**831**	**394**	**999**	762	1,329	1,059	1,691	1,401	1,213	904
PuTransE	12,538	11,644	12,396	11,712	12,508	11,787	13,784	13,058	12,806	12,050
Hits@10(%)										
TransE (static)	29.35	37.48	**30.67**	**38.67**	**32.89**	**42.74**	**33.61**	**43.66**	**31.63**	**40.64**
TransE (pseudo-incremental)	**35.14**	**40.16**	28.33	31.14	29.94	33.76	29.69	34.04	30.78	34.78
PuTransE	15.83	17.20	18.27	20.63	20.73	25.72	21.87	26.80	19.17	22.59

Table 9: Evaluation results of link prediction.

Up to the first snapshot, only triples were added to the KG. Accordingly, the first training snapshot and increment contained the same training triples. Thus, at this point, the static and pseudo-incremental learning variants of TransE learned their embeddings on the same training data. Although both learning variants also used the same hyperparameters configuration, the pseudo-incremental variant achieved a better overall result at the first snapshot. We assume that this happened by chance due to the random sampling of training examples and the stochastic gradient descent performed in the training procedure.

At the remaining snapshots and in the KG's entire evolution, the static learning procedure performed best in link prediction. This also becomes clear by comparing the macro-average values of the three models. The good performance is probably because the model learned the KG's training datasets repeatedly from scratch and can represent the KG elements more accurately. The filtered link prediction results of the static procedure at the different snapshots further indicate its robustness over time. Whereas the respective Mean Rank values fluctuated only slightly, the filtered Hits@10 scores even improved over time.

In contrast, the performance results for the pseudo-incremental procedure of TransE performed worse with each snapshot. Compared to the static learning procedure, it performed worse throughout the evolution of the KG except at the first snapshot. The results indicate that static methods have to relearn their embeddings after a KG is modified in order to sustain their predictive power.

The incremental KG embedding method PuTransE was stable in its performance throughout the KG. While the model achieved similar Mean Rank results on all snapshots, the Hits@10 score increased from the first until the last snapshot.

The statistics of the filtered WikidataEvolve dataset (see section 6.3) express that there have been $47k$ entities on average at the different snapshots. Thus, PuTransE had to rank a test triple among the same number of negative ranking candidates on average. Consequently, the achieved macro-averaged Mean Rank of approximately $12k$ indicates that the incremental model has not positioned the test triple on the top ranks but, on average, higher than most negative examples.

Nevertheless, table 9 reveals that PuTransE performed significantly worse than the other evaluation objects in link prediction. As we argued in the evaluation setting of 4.7.2, the underlying KG embedding concept of PuTransE, is based to a great extent on randomness. As the authors further did not publish their source code, we could not find any improvement points.

Only we could try to achieve better performances for PuTransE in the context of our experiment by adjusting the hyperparameter ranges or allowing the model to produce more embedding spaces per training update. Nevertheless, due to time constraints, we could not comply with both options.

8.2.2. Triple Classification

Table 10 displays the results of triple classification. For each model, the snapshot-wise accuracy scores are listed, and the overall performance is expressed by the micro-averaged (MIA) and macro-averaged accuracy (MAA). The respective formulas can be reviewed in section 5.3.2. We further have disclosed the detailed classification outcomes for each model in table 14 in appendix B.

As already explained, we sampled 6000 test triples from the corresponding test datasets at each snapshot. Thus, we always used the same number of classification examples in the evolution of the KG. Consequently, measured accuracy scores have been equally weighted in the micro-average calculation and led to identical macro and micro-average values.

	Accuracy(%)				MIA(%)	MAA(%)
	Snapshot 1	Snapshot 2	Snapshot 3	Snapshot 4		
TransE (static)	**93.05**	**94.81**	**95.47**	**95.61**	**94.73**	**94.73**
TransE (pseudo-incremental)	92.79	90.71	90.51	91.06	91.27	91.27
PuTransE	52.68	52.46	52.71	52.53	52.59	52.59

Table 10: Evaluation results of triple classification.

The results show that, just as in the link prediction task, the static learning procedure of TransE scored best on all snapshots and in total. Also the pseudo-incremental learning variant achieved high accuracy scores at all snapshots, which did not deteriorate as in the link prediction task but remained stable.

PuTransE, in contrast, achieved poor accuracy results on all snapshots and in the entire KG evolution, as reflected in the related micro-averaged or macro-averaged accuracy scores. Notably, the model predicted the majority of the positive and negative test examples as false (see table 14 in appendix B). We suspect that it failed to map the elements of these triples into common universes. Therefore, it could not calculate energy values for these examples following the parallel universe methodology. As we stated, we replaced missing plausibility scores with the negative infinity score. Accordingly, affected test examples would be predicted as false in the triple classification. This would explain the high number of true negatives, false negatives and poor predictive performance.

8.2.3. Negative Triple Classification

Table 11 finally reveals the results PuTransE has achieved in negative triple classification. Here the accuracy scores measured for the three categories of deleted, positive oscillated and negative oscillated triples at the different snapshots are listed. As in triple classification, we employ the micro-averaged and macro-averaged accuracy to summarize the performances achieved for each category. Furthermore, we have listed the detailed classification outcomes in table 15 in appendix C.

	Accuracy(%)				MIA(%)	MAA(%)
	Snapshot 1	Snapshot 2	Snapshot 3	Snapshot 4		
Category 2 (Deleted Triples)	-	83.34	88.60	87.54	87.33	86.49
Category 3 (Positive Oscillated Triples)	-	-	17.58	10.59	11.92	14.08
Category 4 (Negative Oscillated Triples)	-	-	-	78.52	78.52	78.52

Table 11: Measured classification accuracy scores for PuTransE in the negative triple classification.

In the third category, triples were considered, which reemerged in the KG after their deletion. So the model had to learn them again and classify them as true. In contrast to the second and fourth categories, the evaluated model achieved the lowest performance here. While we measured an accuracy value of 17.58% in the third snapshot, the model's performance even decreased in the fourth snapshot and reached a 10.59% accuracy. About the macro-average accuracy, the model achieved an overall accuracy of only 14.08%. Since there are was a higher number of examples classified in the fourth snapshot than in in the third snapshot, the worse result of 10.59% had been weighted more, which explains the lower micro-averaged accuracy of 11.92%.

In contrast, the second and fourth categories contained solely false triples. Whereas triples in the second category have been deleted precisely once in their history in the KG, triples of the fourth category received at least two deletions. So, in both cases, the model had to unlearn these facts. In general, PuTransE has achieved higher classification performances for both categories than for the true, reinserted facts from the third category.

For the second category, we could collect test instances from the second to the last snapshot. The accuracy scores measured in this range were stable and permanently above 80%. Expressed by the micro and macro-averages, PuTransE achieved its best performance in this group.

For the fourth category, we could only gather examples in the fourth snapshot. Therefore the measured accuracy score is equal to the corresponding micro and macro-averages. Here the model achieved its second-best performance with 78.52%. However, to obtain a more expressive result for the classification performance of PuTransE in this class, a higher number of measurement points, i.e. updates or snapshots, would have to be selected in the construction of the WikidataEvolve dataset. As outlined in section 7, our evaluation framework is flexibly configurable and allows future research to follow such an approach. For our work, we relied on four snapshots in order to keep the results of our experiment comprehensive.

Since examples for negative triple classification are not only sourced from the test and validation data but also from the training data, the classification results indicate that PuTransE had difficulties in relearning facts in the KG evolution, i.e. assigning them a high plausibility.

As previously discussed for the results of triple classification, we suspect that the model failed to produce common universes for most of these examples. So we attached their plausibilities to the negative infinity value, making the model predict them as false.

Reviewing the description of negative triple classification explained in section 5.3.2, a fact is assigned to the second category when it was firstly deleted from the KG, to the third category when it was inserted again and to the fourth category when it was deleted again. In this way, the accuracy values listed in the diagonal of the table 11 express the model's performance for triples whose truth values alternated with each snapshot in the evolving KG.

Here PuTransE was able to unlearn these triples adequately after their deletions, expressed by an accuracy score of 83.34% in the second category of the second snapshot and 78.52% in the fourth category of the fourth snapshot. The low value of 17.58% in the third category indicates, however, as stated above, that the model had problems to relearn these alternated cases after their insertion.

8.2.4. Summary of Results

Reflecting our experiment results in the overall picture, we conclude that TransE with both learning procedures performed significantly better in link prediction and triple classification than PuTransE.

The static learning method of TransE, which learned its embeddings repeatedly from scratch, performed best in total. In link prediction, it ranked positive test examples on the higher ranks on average. Furthermore, it achieved the highest accuracy values in the triple classification task.

Also, the pseudo-incremental learning variant of TransE achieved relatively good results, even though it learned only the increments of the KG. Although its performance decreased over time in link prediction, it remained stable in triple classification with high accuracy values at all snapshots.

In the end, PuTransE achieved the worst results compared to the learning variants mentioned above. The model's performance in the link prediction was relatively stable during the KG evolution and was not subject to any fluctuations. Besides, it placed positive test triples averagely on higher ranks than the majority of negative examples. Nevertheless, it performed significantly worse than TransE.

In the triple classification, the incremental model also achieved significantly worse results. Noticeable was that it predicted most of the positive as well as negative test examples as false. Explained with the parallel universes methodology, we assume that the model failed to map the test triples' elements into a common universe such that no plausibility value could be calculated for such cases. As we equalized their plausibility scores with the lowest possible value, i.e. the negative infinity value, the model has classified them as false.

This behavior is also indicated by the results of PuTransE in the negative triple classification. For deleted facts that were negative at the time of evaluation, the model achieved its best performances by correctly predicting the majority of these examples as false. For facts that were added to the KG again after their deletion, we measured the poorest accuracy. Thus, PuTransE had difficulties in learning them.

This reflects also on the performance of PuTransE for triples whose truth values have changed with each snapshot. In case these alternating triples had been removed from the KG and became false, the model has correctly classified the majority of these examples as false. However, if these alternated triples have been reinserted and became valid, most of them were also predicted as false.

In conclusion, PuTransE achieved a better performance in unlearning than in learning facts. However, considering the results of PuTransE in the other evaluation tasks, it has to be questioned whether the model has even managed to learn the majority of KG facts, i.e. assigning them to a high plausibility in the KG evolution.

From an overall perspective, our experiment demonstrates how, using our evaluation framework, incremental KG embedding methods can be adequately evaluated and compared with static relatives, and more importantly, how the strengths and weaknesses of an incremental model can be consulted in the context of an evolving KG.

9. Conclusion & Future Work

The research direction of Incremental KGE is not very advanced yet. Up to now, only three works exist which describe incremental KG embedding models. However, since these models have been evaluated in different scenarios, their results are not comparable. Furthermore, these evaluation scenarios show shortcomings regarding the evaluation of incremental methods.

In the scope of this thesis, we have developed an evaluation framework for the assessment of incremental methods. For this purpose, these three existing works have been examined regarding their incremental concepts and the strengths and weaknesses of their evaluation strategies. The analysis revealed that the authors failed in designing an evaluation scenario that reflects an authentic evolving KG. Furthermore, they neglected the characteristics of incrementally learned embeddings.

Based on the identified strengths and drawbacks, we elaborated on requirements, which we took into account in the development of our evaluation framework. Ultimately, it is designed to evaluate and compare incremental KG embedding models throughout the evolution of a KG, and further allows to include static KG embedding models. Besides, it formulates the newly conceived task of negative triple classification, which explicitly assesses an incremental model's ability to learn and unlearn facts.

To feed our evaluation framework with factual data, we have compiled the WikidataEvolve dataset. Representing an evolving KG, it provides the first adequate benchmark for the research field of incremental KG embedding and the evaluation of incremental models. Apart from incremental models, it is also compatible with static methods and allows them to be applied in two different learning procedures. Besides a static learning procedure trained on the KG's snapshots throughout its evolution, we designed a variant that we characterize as pseudo-incremental. The key idea here was to conserve a static model's embeddings and optimize them on the KG increments. We investigated how such a procedure performs in contrast to the static learning procedure. The literature suggests that static models have to be learned from scratch after a graph has been modified. No work has yet examined to what extent this statement applies. Using the pseudo-incremental learning variant, we have made this statement verifiable.

Furthermore, we implemented the incremental embedding method PuTransE, and based on its concept, created the new incremental methods PuTransD and PuTransH. Finally, we applied our evaluation framework and the WikidataEvolve benchmark to evaluate the incremental method PuTransE and compared it with the static and pseudo-incremental learning variant of the method TransE.

Here both learning variants of TransE performed significantly better than PuTransE. Furthermore, the experiment results indicated that the pseudo-incremental learning variant of TransE diminished in its performance in link prediction over time, but remained stable for triple classification. In addition, we examined PuTransE concerning its ability to learn and unlearn facts in the new task of negative triple classification. Notably, the model had weaknesses in learning facts reinserted into the KG after their deletion. On the other hand, the model could classify most of the deleted facts correctly as false. It strengthened the suspicion that PuTransE had general difficulties in learning facts, respectively assigning them a high plausibility. Consequently, our artifacts, including our evaluation framework, the WikidataEvolve benchmark and the newly designed models of PuTransH and PuTransD, lay a foundation for developing the research direction of incremental KG embedding.

In case new incremental embedding models emerge in the future, we propose to evaluate them by using our framework. We urge to do so for the existing incremental models of iTransA, DKGE and PuTransE, and the newly created relatives PuTransH and PuTransD by exploiting the artifacts developed in this work.

As we outlined, the elements of our framework are flexibly configurable. First, based on the sequence of triple operations Wikidata9M, the number of snapshots can be freely adjusted and a modified version of the WikidataEvolve dataset can be compiled. Second, the tasks of triple classification and link prediction are seamlessly exchangeable with other evaluation tasks. We suggest considering these possibilities for future use of our framework. Our experiment evaluated the models at four measurement points, respectively snapshots during the KG evolution. However, future works might increase this number. In this way, more test examples are probably collectible for the test categories of negative triple classification.

With the task of negative triple classification, we considered incremental embeddings' characteristics to maintain and adapt their representations over time. In this respect, learning and unlearning are essential capabilities of incremental embedding methods. Finally, we encourage future works to elaborate on additional evaluation scenarios for the measurement of both capabilities.

Acknowledgements

First of all, I want to express special thanks to Lukas Schmelzeisen. Without his guidance and his advice, this thesis, in its result, would not have been possible.

Next, I want to thank Steffen Staab and Zeyd Boukhers, who were very tolerant of the difficulties I encountered during this thesis.

Finally, I thank my family, who gave me continuous support not only during this thesis, but throughout my life.

References

Farahnaz Akrami, Lingbing Guo, Wei Hu, and Chengkai Li. Re-evaluating embedding-based knowledge graph completion methods. In *CIKM*, pages 1779–1782. ACM, 2018.

Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250. ACM, 2008.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *AAAI*. AAAI Press, 2011.

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

Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. In *EMNLP*, pages 615–620. ACL, 2014.

Silvio Cordeiro, Carlos Ramisch, and Aline Villavicencio. mwetoolkit+sem: Integrating word embeddings in the mwetoolkit for semantic MWE processing. In *LREC*. European Language Resources Association (ELRA), 2016.

Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, and Wei Wang. KBQA: learning question answering over QA corpora and knowledge bases. *PVLDB*, 10(5):565–576, 2017.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, pages 1811–1818. AAAI Press, 2018.

Deepika Devarajan. Ibm cloud blog, 2017. URL `https://www.ibm.com/blogs/bluemix/ 2017/12/happy-birthday-watson-discovery/`.

Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. In *SEMANTiCS (Posters, Demos, SuCCESS)*, volume 1695 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and YAGO. *Semantic Web*, 9 (1):77–129, 2018.

Fábio Figueiredo, Leonardo C. da Rocha, Thierson Couto, Thiago Salles, Marcos André Gonçalves, and Wagner Meira Jr. Word co-occurrence features for text classification. *Inf. Syst.*, 36(5):843–858, 2011.

Alberto García-Durán, Sebastijan Dumancic, and Mathias Niepert. Learning sequence encoders for temporal knowledge graph completion. In *EMNLP*, pages 4816–4821. Association for Computational Linguistics, 2018.

Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. Openke: An open toolkit for knowledge embedding. In *Proceedings of EMNLP*, 2018.

Shizhu He, Kang Liu, Guoliang Ji, and Jun Zhao. Learning to represent knowledge graphs with gaussian embedding. In *CIKM*, pages 623–632. ACM, 2015.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *CoRR*, abs/2003.02320, 2020.

Yantao Jia, Yuanzhuo Wang, Hailun Lin, Xiaolong Jin, and Xueqi Cheng. Locally adaptive translation for knowledge graph embedding. In *AAAI*, pages 992–998. AAAI Press, 2016.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Arun Krishnan. Making search easier: How amazon's product graph is helping customers find products more easily, 2018. URL `https://blog.aboutamazon.com/innovation/making-search-easier/`.

M. Kroetsch and G. Weikum. Special issue on knowledge graphs, 2016. URL `http://www.websemanticsjournal.org/index.php/ps/announcement/view/19`.

Heeyoung Kwon, Mirza Basim Baig, and Leman Akoglu. A domain-agnostic approach to spam-url detection via redirects. In *PAKDD (2)*, volume 10235 of *Lecture Notes in Computer Science*, pages 220–232, 2017.

Timothée Lacroix, Guillaume Obozinski, and Nicolas Usunier. Tensor decompositions for temporal knowledge base completion. In *ICLR*. OpenReview.net, 2020.

Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. YAGO3: A knowledge base from multilingual wikipedias. In *CIDR*. www.cidrdb.org, 2015.

George A. Miller. WORDNET: A lexical database for english. In *HLT*. Morgan Kaufmann, 1994.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *CVPR*, pages 5425–5434. IEEE Computer Society, 2017.

Giang Hoang Nguyen, John Boaz Lee, Ryan A. Rossi, Nesreen K. Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *WWW (Companion Volume)*, pages 969–976. ACM, 2018.

Maximilian Nickel and Volker Tresp. Tensor factorization for multi-relational learning. In *ECML/PKDD (3)*, volume 8190 of *Lecture Notes in Computer Science*, pages 617–621. Springer, 2013.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816. Omnipress, 2011.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

Natasha F. Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: Lessons and challenges. *ACM Queue*, 17(2):20, 2019.

Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8(3):489–508, 2017.

Zuzana Reitermanova. Data splitting. In *WDS*, volume 10, pages 31–36, 2010.

Ryan A. Rossi, Luke K. McDowell, David William Aha, and Jennifer Neville. Transforming graph data for statistical relational learning. *J. Artif. Intell. Res.*, 45:363–441, 2012.

Maya Rotmensch, Yoni Halpern, Abdulhakim Tlimat, Steven Horng, and David Sontag. Learning a health knowledge graph from electronic medical records. *Scientific reports*, 7(1):5994, 2017.

Baoxu Shi and Tim Weninger. Open-world knowledge graph completion. In *AAAI*, pages 1957–1964. AAAI Press, 2018.

Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, pages 926–934, 2013.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706. ACM, 2007.

Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. Recurrent knowledge graph embedding for effective recommendation. In *RecSys*, pages 297–305. ACM, 2018.

Thomas Pellissier Tanon, Denny Vrandecic, Sebastian Schaffert, Thomas Steiner, and Lydia Pintscher. From freebase to wikidata: The great migration. In *WWW*, pages 1419–1428. ACM, 2016.

Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. Non-parametric estimation of multiple embeddings for link prediction on dynamic knowledge graphs. In *AAAI*, pages 1243–1249. AAAI Press, 2017.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *EMNLP*, pages 1499–1509. The Association for Computational Linguistics, 2015.

Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3462–3471. PMLR, 2017.

Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Trans. Knowl. Data Eng.*, 29(12): 2724–2743, 2017.

Xiang Wang, Dingxian Wang, Canran Xu, Xiangnan He, Yixin Cao, and Tat-Seng Chua. Explainable reasoning over knowledge graphs for recommendation. In *AAAI*, pages 5329–5336. AAAI Press, 2019.

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

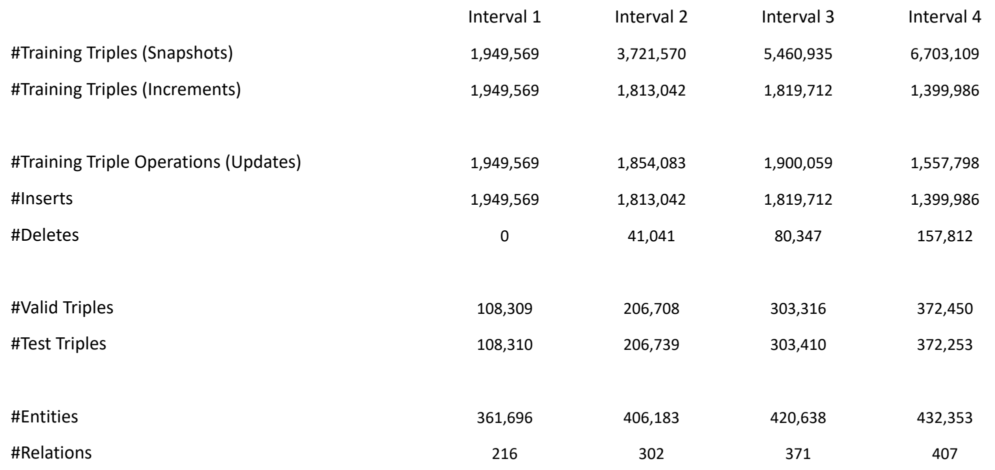
Zhouxia Wang, Tianshui Chen, Jimmy S. J. Ren, Weihao Yu, Hui Cheng, and Liang Lin. Deep reasoning with knowledge graph for social relationship understanding. In *IJCAI*, pages 1021–1028. ijcai.org, 2018.

Tianxing Wu, Arijit Khan, Huan Gao, and Cheng Li. Efficiently embedding dynamic knowledge graphs. *CoRR*, abs/1910.06708, 2019.

Han Xiao, Minlie Huang, and Xiaoyan Zhu. Transg : A generative model for knowledge graph embedding. In *ACL (1)*. The Association for Computer Linguistics, 2016.

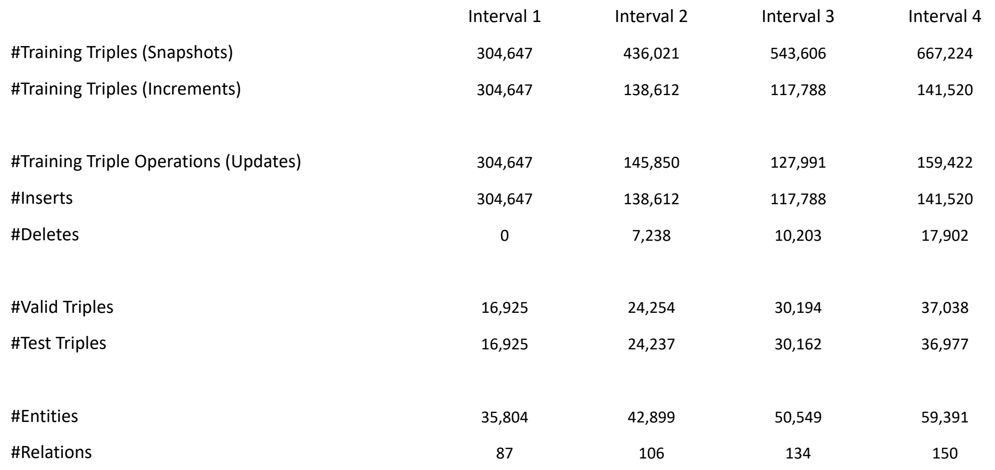
Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *ICLR*. OpenReview.net, 2020.

A. Detailed Statistics about WikidataEvolve



	Interval 1	Interval 2	Interval 3	Interval 4
#Training Triples (Snapshots)	1,949,569	3,721,570	5,460,935	6,703,109
#Training Triples (Increments)	1,949,569	1,813,042	1,819,712	1,399,986
#Training Triple Operations (Updates)	1,949,569	1,854,083	1,900,059	1,557,798
#Inserts	1,949,569	1,813,042	1,819,712	1,399,986
#Deletes	0	41,041	80,347	157,812
#Valid Triples	108,309	206,708	303,316	372,450
#Test Triples	108,310	206,739	303,410	372,253
#Entities	361,696	406,183	420,638	432,353
#Relations	216	302	371	407

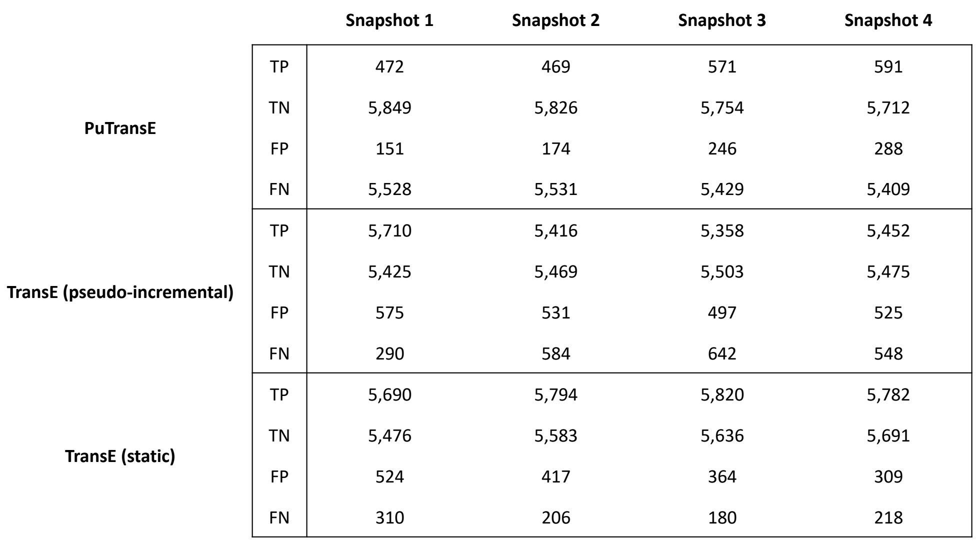
Table 12: Details about the raw version of WikidataEvolve.



	Interval 1	Interval 2	Interval 3	Interval 4
#Training Triples (Snapshots)	304,647	436,021	543,606	667,224
#Training Triples (Increments)	304,647	138,612	117,788	141,520
#Training Triple Operations (Updates)	304,647	145,850	127,991	159,422
#Inserts	304,647	138,612	117,788	141,520
#Deletes	0	7,238	10,203	17,902
#Valid Triples	16,925	24,254	30,194	37,038
#Test Triples	16,925	24,237	30,162	36,977
#Entities	35,804	42,899	50,549	59,391
#Relations	87	106	134	150

Table 13: Details about the filtered version of WikidataEvolve.

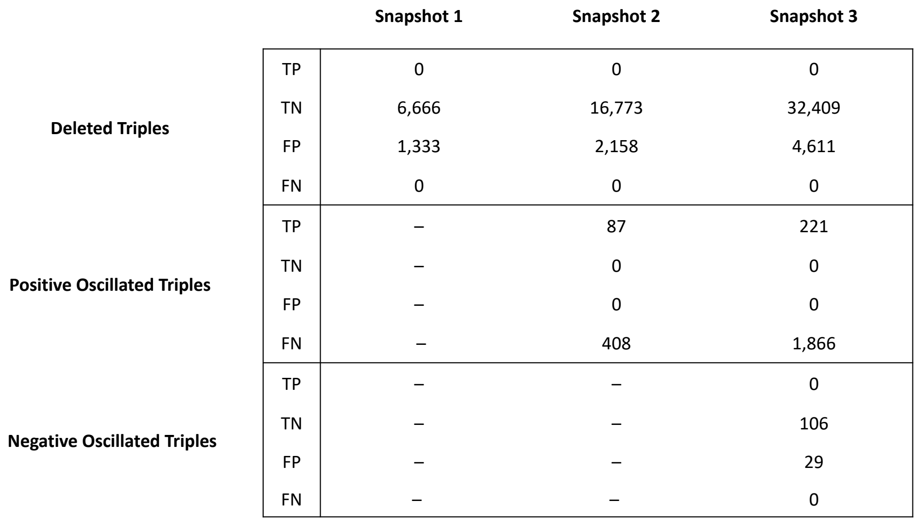
B. Classification Outcomes of Triple Classification



		Snapshot 1	Snapshot 2	Snapshot 3	Snapshot 4
PuTransE	TP	472	469	571	591
	TN	5,849	5,826	5,754	5,712
	FP	151	174	246	288
	FN	5,528	5,531	5,429	5,409
TransE (pseudo-incremental)	TP	5,710	5,416	5,358	5,452
	TN	5,425	5,469	5,503	5,475
	FP	575	531	497	525
	FN	290	584	642	548
TransE (static)	TP	5,690	5,794	5,820	5,782
	TN	5,476	5,583	5,636	5,691
	FP	524	417	364	309
	FN	310	206	180	218

Table 14: Classification outcomes of triple classification.

C. Classification Outcomes of Negative Triple Classification



		Snapshot 1	Snapshot 2	Snapshot 3
Deleted Triples	TP	0	0	0
	TN	6,666	16,773	32,409
	FP	1,333	2,158	4,611
	FN	0	0	0
Positive Oscillated Triples	TP	–	87	221
	TN	–	0	0
	FP	–	0	0
	FN	–	408	1,866
Negative Oscillated Triples	TP	–	–	0
	TN	–	–	106
	FP	–	–	29
	FN	–	–	0

Table 15: Classification outcomes for the incremental knowledge graph embedding method PuTransE in negative triple classification.