Learning Triple Embeddings from Knowledge Graphs

Valeria Fionda,[1] **Giuseppe Pirrò**[2*]
[1]Department of Mathematics and Computer Science, University of Calabria
Via Pietro Bucci 30B, 87036, Rende (CS), Italy
[2]Department of Computer Science, Sapienza University of Rome
Via Salaria 113, 00198, Rome, Italy
fionda@mat.unical.it, pirro@di.uniroma1.it

Abstract

Graph embedding techniques allow to learn high-quality feature vectors from graph structures and are useful in a variety of tasks, from node classification to clustering. Existing approaches have only focused on learning feature vectors for the nodes and predicates in a knowledge graph. To the best of our knowledge, none of them has tackled the problem of directly learning triple embeddings. The approaches that are closer to this task have focused on homogeneous graphs involving only one type of edge and obtain edge embeddings by applying some operation (e.g., average) on the embeddings of the endpoint nodes. The goal of this paper is to introduce Triple2Vec, a new technique to directly embed knowledge graph triples. We leverage the idea of line graph of a graph and extend it to the context of knowledge graphs. We introduce an edge weighting mechanism for the line graph based on semantic proximity. Embeddings are finally generated by adopting the SkipGram model, where sentences are replaced with graph walks. We evaluate our approach on different real-world knowledge graphs and compared it with related work. We also show an application of triple embeddings in the context of user-item recommendations.

1 Introduction

In the last years, learning graph representations using low-dimensional vectors has received attention as viable support to various (machine) learning tasks, from node classification to clustering (Cai, Zheng, and Chang 2018). Approaches like DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), node2vec (Grover and Leskovec 2016) and their variants strive to find node embeddings that preserve structural relations in the embedding space. These approaches only focus on *homogeneous graphs* including only one type of edge. Another strand of research focused on embedding nodes in knowledge graphs (aka heterogeneous information networks) characterized by several distinct types of nodes and edges (Wang et al. 2017). Notable approaches are RDF2Vec (Ristoski and Paulheim 2016), metapath2vec (Dong, Chawla, and Swami 2017), and JUST (Hussein, Yang, and Cudré-Mauroux 2018). One common denominator of both homogeneous and knowledge graph embedding approaches is the usage of language model techniques. The idea is to consider sequences of nodes in a graph (i.e., random walks) as analogous to sentences in a document; then, the node sequences are fed into models like Skip-gram (Mikolov et al. 2013) to learn the final node embeddings. There have been previous attempts considering edge embeddings in homogeneous graphs. In particular, node2vec construct edge embeddings by applying some operator (e.g., average, Hadamard product) to the embeddings of the endpoint nodes of an edge. However, this will be problematic in the context of KGs since it is not clear how to behave when nodes are linked by more than an edge as typically happens in real-world KGs like DBpedia. Other approaches like ComplexE (Trouillon et al. 2016), ConvE (Dettmers et al. 2018), TransG (Xiao, Huang, and Zhu 2016), and RotatE (Sun et al. 2019) learn embeddings of both entities and predicate types for link prediction. These approaches could compute triple embeddings by aggregating node and predicates embeddings. This approach is sub-optimal as it will consider the same predicate and node embeddings in all triples where these appear.

The goal of this paper is to devise a novel technique to directly learn triple embeddings from knowledge graphs. This sets three main challenges. *The first is about how to go from node embeddings to triple embeddings*. To tackle this first challenge, we build upon the notion of *line graph* of a graph. As an example, the directed line graph obtained from the knowledge graph in Fig. 1 (a) is shown in Fig. 1 (b), where the two copies of the node Lauren Oliver, Americans correspond to the triples having nationality and citizenship as a predicate, respectively. It would be tempting to directly apply embedding techniques to the nodes of the directed line graph to obtain triple embeddings. However, we detect two main problems. The first is that it is impossible to discern between the two triples encoded by the nodes Lauren Oliver and Americans. The second is that the directed line graph is disconnected and as such, it becomes problematic to learn triple embeddings via random walks. Therefore, we introduce the notion of *triple line graph* $\mathcal{G}_L$ of a knowledge graph G. In $\mathcal{G}_L$, nodes are the triples of G and an edge is introduced whenever the triples of G share an endpoint.

*Corresponding author.

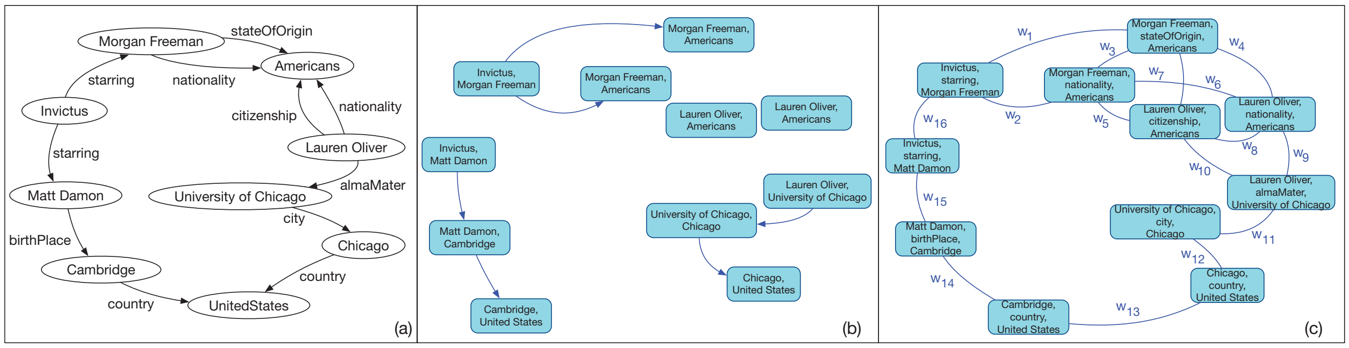

Morgan Freeman
stateOfOrigin
Americans
starring
nationality
Invictus
citizenship
nationality
Lauren Oliver
starring
Matt Damon
University of Chicago
almaMater
city
birthPlace
Chicago
Cambridge
country
country
UnitedStates
(a)
Morgan Freeman, Americans
Invictus, Morgan Freeman
Morgan Freeman, Americans
Lauren Oliver, Americans
Lauren Oliver, Americans
Invictus, Matt Damon
Lauren Oliver, University of Chicago
University of Chicago, Chicago
Matt Damon, Cambridge
Chicago, United States
Cambridge, United States
(b)
Morgan Freeman, stateOfOrigin, Americans
Invictus, starring, Morgan Freeman
Morgan Freeman, nationality, Americans
Lauren Oliver, nationality, Americans
Lauren Oliver, citizenship, Americans
Invictus, starring, Matt Damon
Lauren Oliver, almaMater, University of Chicago
University of Chicago, city, Chicago
Matt Damon, birthPlace, Cambridge
Chicago, country, United States
Cambridge, country, United States
w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9 w_{10} w_{11} w_{12} w_{13} w_{14} w_{15} w_{16}
(c)

Figure 1: A knowledge graph (a), its directed line graph and (c) its triple line graph.

This construction guarantees that $\mathcal{G}_L$ is connected if G is connected. The triple line graph for the graph in Fig. 1 (a) is shown in Fig. Fig. 1 (c). Unfortunately, directly working with $\mathcal{G}_L$ may lead to low-quality embeddings because edges in $\mathcal{G}_L$ are added based on adjacency. *This introduces the second challenge related to the fact that high degree nodes in G are over-represented, in terms of the number of edges, in $\mathcal{G}_L$.* This affects the dynamics of the graph thus having a direct impact on network embedding approaches like Deepwalk or node2vec that are based on the computation of random walks. To tackle this seconds challenge, we introduce a mechanism to weight the edges of $\mathcal{G}_L$ based on predicate relatedness (Pirrò 2019). The weight of an edge between nodes of $\mathcal{G}_L$ is equal to the semantic relatedness between the predicates in the triples of G represented by the two nodes. As an example, in Fig. 1 (c) the weight of the nodes of $\mathcal{G}_L$ (M. Damon, birthPlace, Cambridge) and (Cambridge, country, United States) will be equal to the relatedness between birthPlace and country.

The third challenge consists of how to compute the edge embeddings from the weighted $\mathcal{G}_L$. To this end, we generate truncated random walks, in the form of sequences of nodes, on the weighted triple line graph. Note that weights based on semantic relatedness will bias the random walker to obtain similar contexts for nodes in the weighted triple line graph linked by related predicates. The underlying idea is that similar contexts will lead to similar embeddings. Finally, the walks are fed into the Skip-gram model (Mikolov et al. 2013), which will give the embeddings of the nodes of the weighted triple line graph that correspond to the embeddings of the original triples. We believe that triple embeddings can open up a novel class of downstream applications that go beyond that based on node embeddings. In particular, we are going to discuss applications in the area of edge classification, clustering, and user-item recommendation.

The remainder of the paper is organized as follows. We introduce some preliminary definitions in Section 2. In Section 3, we introduce the notion of triple line graph of a knowledge graph along with an algorithm to compute it. Section 4 describes the Triple2Vec approach to learn triple embeddings from knowledge graphs. In Section 5, we discuss an experimental evaluation. Related work is dealt with in Section 6. We conclude and sketch future work in Section 7.

2 Preliminaries

A Knowledge Graph (G) is a kind of heterogeneous information network. It is a node and edge labeled directed multigraph G=(V_G, E_G, T_G) where V_G is a set of uniquely identified vertices representing entities (e.g., D. Lynch), E_G a set of predicates (e.g., director) and T_G a set of triples of the form (s, p, o) representing directed labeled edges, where $s, o \in V_G$ and $p \in E_G$. The line graph $\mathcal{G}_L$=(V_L, E_L) of an undirected graph G=(V_G, E_G) is such that: (i) each node of $\mathcal{G}_L$ represents an edge of G; (ii) two vertices of $\mathcal{G}_L$ are adjacent if, and only if, their corresponding edges in G have a node in common. Starting from G it is possible to compute the number of nodes and edges of $\mathcal{G}_L$ as follows: *(i)* $|V_L| = |E_G|$; *(ii)* $|E_L| \propto \frac{1}{2}\sum_{v \in V_G} d_v^2 - |E_G|$, where d_v denotes the degree of the node $v \in V_G$.

The concept of line graph has been extended to other types of graphs, including multigraphs and directed graphs. The extension to multigraphs adds a different node in the line graph for each edge of the original multigraph. If the graph G is directed, the corresponding line graph $\mathcal{G}_L$ will also be directed; its vertices are in one-to-one correspondence to the edges of G and its edges represent two-length directed paths in G.

3 Triple Line Graph of a Knowledge Graph

As we have discussed in the Introduction, apply the notion of line graph to knowledge graphs would lead to counterintuitive behaviors. Consider the graph G in Fig. 1 (a). Fig. 1 (b) shows the directed line graph $\mathcal{G}_L$, obtained by applying the standard definition, where nodes of $\mathcal{G}_L$ are in one-to-one correspondence to the edges of G. Moreover, from the same sub-figure it can be noticed that each directed edge of $\mathcal{G}_L$ corresponds to a path of length 2 in G.

At this point, three main issues arise. First, the standard definition associates to each node of the directed line graph the two endpoints of the corresponding edge; however, the edge labels in a knowledge graph carry a semantic meaning, which is completely lost if only the endpoints are considered. As an example, it is not possible to discern between the two copies of the nodes Morgan Freeman, Americans. Second, the edges of the line graph are determined by considering their direction. This disregards the fact that edges in G witness some semantic relation between their endpoints (i.e., entities) that can be interpreted bidirectionally. As an example, according to the definition of directed line graph, the two nodes (Lauren Oliver, Americans) in $\mathcal{G}_L$ remain isolated since the corresponding edges do not belong to any two-length path in G (see Fig. 1 (a)-(b)). However, consider the triple (Lauren Oliver, nationality, Americans). While the edge label from Lauren Oliver to Americans serves the purpose of stating the relation nationality, the same label in the opposite direction states the relation is nationality of.

Hence, *in the case of knowledge graphs, two nodes of the line graph must be connected by an edge if they form a two-length path in the original knowledge graph no matter the edge direction, as the semantics of edge labels can be interpreted bidirectionally*. Finally, triples encode some semantic meaning via predicates, and the desideratum is to preserve this semantics when connecting two nodes (i.e., triples of G) in the line graph. Because of these issues, we introduce the novel notion of *triple line graphs* suitable for KGs.

Definition 3.1 *(Triple Line Graph). Given a knowledge graph $G = (V_G, E_G, T_G)$, the associated triple line graph $\mathcal{G}_L = (V_L, E_L, w)$ is such that: (i) each node of $\mathcal{G}_L$ represents a triple of G; (ii) two vertices of $\mathcal{G}_L$, say s_1, p_1, o_1 and s_2, p_2, o_2, are adjacent if, and only if, $\{s_1, o_1\} \cap \{s_2, o_2\} \neq \emptyset$; (iii) the function w associates a weight in the range $[0, 1]$ to each edge of $\mathcal{G}_L$.*

3.1 Computing Triple Line Graphs

We now describe an algorithm (outlined in Algorithm 1) to compute the triple line graph $\mathcal{G}_L$ of a knowledge graph G. This is at the core of Triple2Vec for the computation of triple embeddings. After initializing the set of nodes and edges of $\mathcal{G}_L$ to the empty set (line 1), the algorithm iterates over the triples of the input G and add a node to $\mathcal{G}_L$ for each visited triple (lines 2-3), thus inducing a bijection from the set of triples of G to the set of nodes of $\mathcal{G}_L$. Besides, if two triples share a node in G then an edge will be added between the corresponding nodes of $\mathcal{G}_L$ (lines 5-14). In particular, the data structure $I(s)$ (line 6) keeps track, for each node s of G, of the triples in which s appears as subject or object (lines 7-8). Since such triples correspond to nodes of the triple line graph, by iterating over pairs of triples in $I(s)$ it is possible to add the desired edge between the corresponding nodes of $\mathcal{G}_L$ (lines 9-10). The algorithm also considers a generic edge weighting mechanism (line 11) that will be detailed in Section 4.1.

By inspecting Algorithm 1, we observe that $\mathcal{G}_L$ can be computed in time $\mathcal{O}(|T|^2 \times costWeight)$, where $costWeight$ is the cost of computing the weight between nodes in $\mathcal{G}_L$.

4 Triple2Vec: Learning Triple Embeddings

We now describe Triple2Vec that directly learns triple embeddings from knowledge graphs. Triple2Vec includes four main phases: (i) *building of the triple line graph* (Section 3); (ii) *weighting of the triple line graph edges* (Section 4.1); (iv) *computing walks* on the weighted triple line graph (Section 4.2), and (v) *computing embeddings* via the Skip-gram model (Section 4.3).

```
Input  : Knowledge Graph $G$
Output: $\mathcal{G}_L$: the Triple Line Graph associated to $G$
 1: $\mathcal{G}_L = \{\emptyset, \emptyset, \emptyset\}$
 2: for all $(s, p, o)$ in $G$ do
 3:     add the node $(s, p, o)$ to $\mathcal{G}_L$
 4: end for
 5: for all $s \in G$ do
 6:     $I(s) = \emptyset$
 7:     for all $(s, p, o)$ (resp., $(o, p, s)$) in $G$ do
 8:         add $s, p, o$ (resp., $o, p, s$) to $I(s)$
 9:         for all pair $n, n'$ in $I(s)$ do
10:             add the edge $(n, n')$ to $\mathcal{G}_L$
11:             set $w(n, n') =$ computeEdgeWeight $(n, n')$
12:         end for
13:     end for
14: end for
15: return $\mathcal{G}_L$
```

Algorithm 1: BuildTripleLineGraph (G)

4.1 Triple Line Graph Edge Weighting

We have mentioned in Section 3.1 that the number of edges in the (triple) line graph can be large. This structure is much denser than that of the original graph and may significantly affect the dynamics of the graph in terms of random walks. To remedy this drawback, we introduce edge weighting mechanism (line 11 Algorithm 1). The desideratum is to come up with a strategy to compute walks so that the neighborhood of a triple will include triples that are semantically related. To this end, we leverage a predicate relatedness measure (Fionda and Pirrò 2018).

This measure is based on the Triple Frequency defined as $TF(p_i, p_j)$=log$(1 + C_{i,j})$, where $C_{i,j}$ counts the number of times the predicates p_i and p_j link the same subjects and objects. Moreover, it uses the Inverse Triple Frequency defined as $ITF(p_j, E)$=log $\frac{|E|}{|\{p_i : C_{i,j} > 0\}|}$ (Pirrò 2019). Based on TF and ITF, for each pair of predicates p_i and p_j we can build a (symmetric) matrix C_M where each element is $C_M(i, j)$=$TF(p_i, p_j) \times ITF(p_j, E)$. The final predicate relatedness matrix M_R can be constructed such that $M_R(p_i, p_j)$=$Cosine(W_i, W_j)$, where W_i (resp., W_j) is the row of p_i (resp., p_j) in C_M. This approach guarantees that the more related predicates in the triples representing two nodes in the triple line graph are, the higher the weight of the edge between these nodes.

Driving the random walks according to a relatedness criterion can capture both the graph topology in terms triple-to-triple relations (i.e., edges in the triple line graph) and semantic proximity in terms of relatedness between predicates in triples.

4.2 Computing Walks

Triple2Vec leverages a language model approach to learn the final edge embeddings. As such, it requires a “corpus” of both nodes and sequences of nodes similarly to word embeddings techniques that require words and sequences of words (i.e., sentences). We leverage truncated graph walks.

The idea is to start from each node of $\mathcal{G}_L$ (representing a triple of the original graph) and provide a context for each of such node in terms of a sequence of other nodes. Although walks have been used by previous approaches (e.g., (Grover and Leskovec 2016)), none of them has tackled the problem of computing triple embeddings.

4.3 Computing Triple Embeddings

Once the "corpus" (in terms of the set of walks $\mathcal{W}$) is available, the last step of the Triple2Vec workflow is to compute the embeddings of the nodes of $\mathcal{G}_L$ that will correspond to the embeddings of the triples of the input graph G. The embedding we seek can be seen as a function $f : V_L \rightarrow R^d$, which projects nodes of the weighted triple line graph $\mathcal{G}_L$ into a low dimensional vector space, where $d \ll |V_L|$, so that neighboring nodes are in proximity in the vector space. For every node $u \in V_L$, $N(u) \subset V_L$ is the set of neighbors, which is determined by the walks computed (Section 4.2). The co-occurrence probability of two nodes v_i and v_{i+1} in a set of walks $\mathcal{W}$ is given by the Softmax function using their vector embeddings e_{v_i} and $e_{v_{i+1}}$:

$$p((e_{v_i}, e_{v_{i+1}}) \in \mathcal{W}) = \sigma(e_{v_i}^T e_{v_{i+1}}) \tag{1}$$

where σ is the Softmax function and $e_{v_i}^T e_{v_{i+1}}$ is the dot product of the vectors e_{v_i} and $e_{v_{i+1}}$. As the computation of (1) is computationally demanding (Grover and Leskovec 2016), we use negative sampling to training the Skip-gram model. Negative sampling randomly selects nodes that do not appear together in a walk as negative examples, instead of considering all nodes in a graph.

If a node v_i appears in a walk of another node v_{i+1}, then the vector embedding e_{v_i} is closer to $e_{v_{i+1}}$ as compared to any other randomly chosen node. The probability that a node v_i and a randomly chosen node v_j *do not* appear in a walk starting from v_i is given by:

$$p((e_j, e_i) \notin \mathcal{W}) = \sigma(-e_{v_i}^T e_{v_j}) \tag{2}$$

For any two nodes v_i and v_{i+1}, the negative sampling objective of the Skip-gram model to be maximized is given by the following objective function:

$$\mathcal{O}(\theta) = \log \sigma(e_{v_i}^T e_{v_{i+1}}) + \sum_{j=1}^{k} \mathbb{E}_{v_j}[\log \sigma(-e_{v_i}^T e_{v_j})], \tag{3}$$

where θ denotes the set of all parameters and k is the number of negative samples. For the optimization of the objective function, we use the parallel asynchronous stochastic gradient descent algorithm (Recht et al. 2011).

5 Experiments

We now report on the evaluation and comparison with related work. Triple2Vec has been implemented in Python and uses the Gensim[1] library to learn embeddings.

[1]https://radimrehurek.com/gensim

5.1 Experiments on Classification and Clustering

In the experiments, we used three real-world data sets. DBLP (Huang and Mamoulis 2017) about authors, papers, venues, and topics. This dataset has ∼16K nodes, ∼52K edges, and 4 predicate types. Authors are labeled with one among four labels (i.e., database, data mining, machine learning, and information retrieval). Foursquare (Hussein, Yang, and Cudré-Mauroux 2018) with ∼30K nodes and ∼83K edges including four different kinds of entities, that is, users, places, points of interests and timestamps. Each point of interest has also associated one among 10 labels. Finally, we used a subset of Yago (Huang and Mamoulis 2017) in the domain of movies. It includes ∼22K nodes, ∼89K edges, and 5 predicate types. Here, each movie is assigned one or more among 5 available labels.

Due to the novelty of the task we face, there are no benchmarks available for the evaluation of triple embedding approaches. However, we constructed a benchmark for two tasks, that is, *triple classification* and *triple clustering* by using the labels available in datasets above described. Specifically, for DBLP the 4 labels for author nodes have been propagated to paper nodes by following authorship links; then, from paper nodes to topic and venue nodes. For Yago, the labels for movies have been propagated to actors, musicians and directors by following actedIn, wroteMusicFor and directed edges, respectively. Finally, the 10 labels for points of interest in FourSquare have been propagated to places, users and timestamps by following locate, perform and happendAt edges, respectively. With this reasoning, each node of each G is now labeled with *a subset* of labels. Hence, nodes of $\mathcal{G}_L$ (i.e., the triples of G) have been labeled with the union of the set of labels associated with the triple endpoints. Finally, to each different subset of labels, we assigned a different meta-label.

Systems and Parameter Setting. As Triple2Vec tackles the novel problem of directly embedding triples, there is an intrinsic difficulty in finding competitors. Nevertheless, we considered the following two groups of related approaches:

1. this group includes metapath2vec (Dong, Chawla, and Swami 2017), node2vec (Grover and Leskovec 2016) implemented in StellarGraph[2] and DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) configured with the best parameters reported in their respective papers. As these approaches compute embeddings for each node only (not for predicates), a triple embedding was obtained by averaging the embeddings of the triple endpoints.
2. this group includes two state-of-the-art approaches for knowledge graph embedding, that is, ConvE (Dettmers et al. 2018) and RotatE (Sun et al. 2019) configured with the best parameters reported in their respective paper and implemented in the pykg2vec[3]. As these approaches compute embeddings for each node and each predicate, triple embeddings were obtained by concatenating the embeddings of the triple endpoints and the predicate.

[2]https://www.stellargraph.io
[3]https://github.com/Sujit-O/pykg2vec

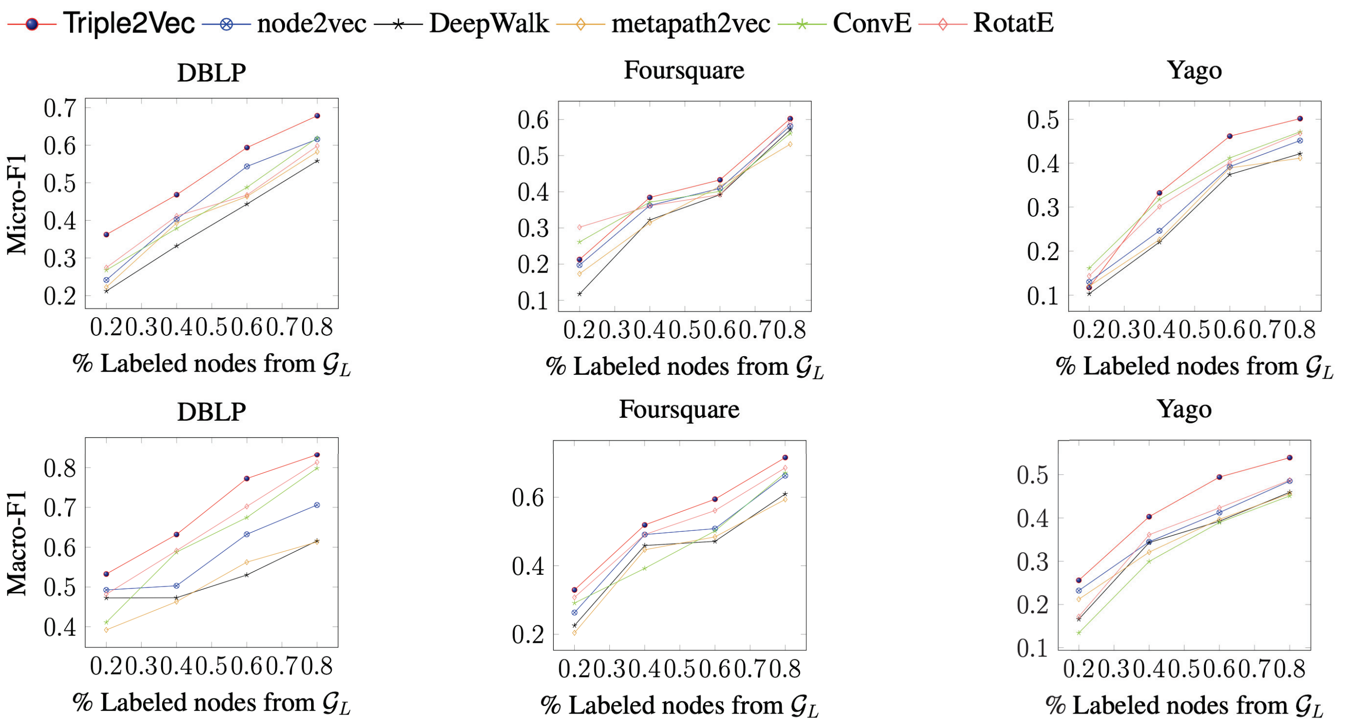
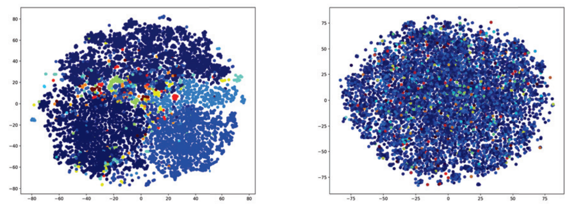


Figure 2: Triple classification results in terms of Micro and Macro F1.

For sake of space, in what follows, we only report the best results obtained by setting the parameters of Triple2Vec as follows: number of walks per node n=10, maximum walk length $L = 100$, window size (necessary for the context in the Skip-gram model) $w = 10$. Moreover, we used d=128 as a dimension of the embeddings. The number of negative samples Γ is set to 50. All results are the average of 10 runs.

Results on Triple Classification. To carry out this task, we trained a one-vs-rest Logistic regression model, giving as input the triple embeddings along with their labels (the labels of the node of $\mathcal{G}_L$). Then, we compute the Micro-F1 and Macro-F1 scores by varying the percentage of training data. The results of the evaluation are reported in Fig. 2. We observe that Triple2Vec consistently outperforms competitors. This is especially true in the DBLP and Yago datasets. We also note that metapath2vec performs worse than node2vec and DeepWalk, although the former has been proposed to work on knowledge graphs. This may be explained by the fact that the metapaths used in the experiments (Hussein, Yang, and Cudré-Mauroux 2018), while being able to capture node embeddings, fail short in capturing edge (triple) embeddings.

As for the other group of approaches, we can see that the performance are even worse than the first group in some cases although also predicate embeddings are considered. This may be since the goal of these approaches is to learn entity and predicate embeddings for link prediction. Therefore, the concatenation of entity and predicate embeddings to form triple embeddings does not correctly capture triple embeddings. Another reason is the fact that these approaches compute a single predicate and node embedding, which is the same for all triples in which it appears. On the other hand, Triple2Vec, which directly compute triple embeddings can better capture and discriminate the roles of the same predicate /node in different triples.



(a) Triple2Vec (b) metapath2vec

Figure 3: DBLP triple embedding visualization.

Results on Triple Clustering and Visualization. To have a better account of how triple embeddings are placed in the embedding space, we used t-SNE (Maaten and Hinton 2008) to obtain a 2-d representation of the triple embeddings (originally including d dimensions) obtained from Triple2Vec and metapath2vec. We only report results for DBLP (Fig. 3) as we observed similar trends in the other datasets. Moreover, metapath2vec was the system giving the "most graphically readable" results. We note that while Triple2Vec can clearly identify groups of triples (i.e., triples labeled with the same labels), metapath2vec offers a less clear perspective.

We can explain this behavior with the fact that Triple2Vec defines a specific strategy for triple embeddings based on the notion of semantic proximity, while triple embeddings for metapath2vec have been obtained from the embedding of endpoint nodes according to predefined metapaths.

5.2 Recommendations via Triple Embeddings

As a concrete application of triple embeddings, we considered the task of user-item recommendation using KGs as a source of background knowledge. The underlying idea of this approach is to leverage paths that exist between users and items in a KG to estimate the probability that a user will interact with a target item. Incorporating background knowledge is useful for two main reasons. First, it can help in *visually* explaining why a user may be interested in a particular item. As an example, a user may be interested in a song written by the same author of another song s/he has listened to. Second, a set of user-item paths, if appropriately encoded, can be feed into a learning model to *automatically* provide a verdict whether a given user will interact with an item.

We identify three main challenges for the setting up of this machinery. The first is how to extract user-item paths. The second concerns how to encode these paths for the learning model. The third is the definition of the learning model itself. As for the first challenge, we use an simple algorithm, which given a pair (*user*, *item*) finds all paths of at most length d. This approach has been used in previous work (e.g., (Wang et al. 2019)) and we found this strategy viable for moderately-large KGs. As for the second challenge, we proceed as follows: consider a path $\mathsf{u}\xrightarrow{p_1}\mathsf{e}_1\xrightarrow{p_2}\mathsf{e}_2\ldots\mathsf{e}_{d-1}\xrightarrow{p_{d-1}}\mathsf{i}$, u is a user, i is an item, p_i ($i \in [1, d]$) is a predicate, and e_i is an entity all available in the same KG. We replace each of the constituent triples of the path with its *triple embedding*.

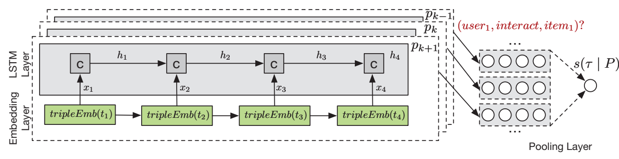


Figure 4: User-item recommendation learning architecture.

As the goal of this experiment is to show the usage of triple embeddings rather than specifically focusing on the problem of item recommendation in KGs, to tackle the third challenge, we use the KPRN (Wang et al. 2019) learning model. KPRN leverages an LSTM, capable of incorporating long-term dependencies, and a weighted pooling layer, which can discriminate the strength of the different paths (see Fig. 4). Finally, the task of user-item recommendation is treated as a binary classification problem, where an observed user-item interaction is assigned a target value 1, otherwise 0. Further details are available in (Wang et al. 2019).

Datasets and Experimental Setting. We used the KPRN available implementation[4] as well as the same datasets. These are MovieLens-1M (referred to as MI) enriched with data from IMDb (∼6K users, ∼4K items, ∼1M interactions, ∼1M triples in the final KG, ∼55M extracted user-item paths) and KKBox, which includes user-item interaction data along with information about songs, singers, songwriters and genre (∼34K users, ∼2.2M items, ∼3.7M interactions, ∼11M triples in the final KG, ∼38M extracted user-item paths). To test the usefulness of triple embeddings in the recommendation task, we considered two variants of KPRN by changing the function *tripleEmb*(·) in Fig. 4.

- KPRN, the original way, where *tripleEmb*(·) embeds each triple as (E_i, E_i^t, P_i), where E_i is the embedding of the entity e_i in the path, E_i^t is the embedding of the entity type of e_i, and P_i the embedding of the predicate in the triple. A user-item path is a sequence of such triplets;
- KPRN$^{\mathsf{Triple2Vec}}$, where *tripleEmb*(·) is the triple embedding implemented by Triple2Vec. A user-item path is a sequence of triple embeddings.

In both cases, the system has been configured with the best hyper-parameters reported in (Wang et al. 2019). We did not consider other competitor approaches in this experiments, since KPRN has been shown to outperform state-of-the-art approaches (Wang et al. 2019).

For the evaluation, we adopted the methodology in (Wang et al. 2019) based on the computation of:

- hit@k: it considers whether relevant items are retrieved within the top-k positions of the recommendation list
- ndcg@k: it measures the relative orders among positive and negative items within the top-k of the ranking list

The results reported in Table 1 are the average of all instances in the test set for k={5,10,15}.

Results. We want to mention that this experiment was meant to give an idea of the usefulness of triple embeddings in an important downstream application like user-item recommendation using KGs and was not meant to design a novel user-item recommendation model.

Nevertheless, we can observe in Table 1 that in both datasets, encoding the user-item paths using triple embeddings brings some benefit to the KPRN model. We believe that this is mainly because while KPRN encodes each triple in a path by considering the concatenation of the entity, entity type and predicate, our approach directly considers the whole triple.

To give a more precise account of why triple embeddings are important, consider the path $\mathsf{Mary}\xrightarrow{interact}\mathsf{Lost}\xrightarrow{sungBy}\mathsf{Rob}$ with u of type User, Lost of type Item and Rob of type Person. According to KPRN, this path will be encoded as the sequence {(*Emb*(Mary)⊕*Emb*(User)⊕*Emb*($interact$)),(*Emb*(Rob)⊕*Emb*(Person)⊕*Emb*(END))} where *END* denotes the end of the path and *Emb* is the knowledge graph embedding function that learns entity and predicate embeddings.

On the other hand, our approach encodes the same path as {*Triple2Vec*(Mary, *interacts*, Lost), *Triple2Vec*(Lost, *sungBy*, Rob)}. We believe that our direct triple encoding, where there the object of a triple at position i is shared with

[4]https://github.com/eBay/KPRN

Table 1: Performance comparison of KPRN and KPRN$^{\text{Triple2Vec}}$

	MI						KKBox					
	hit@5	hit@10	hit@15	ndcg@5	ndcg@10	ndcg@15	hit@5	hit@10	hit@15	ndcg@5	ndcg@10	ndcg@15
KPRN	0.676	0.773	0.832	0.584	0.616	0.632	0.717	0.823	0.881	0.613	0.637	0.652
KPRN$^{\text{Triple2Vec}}$	0.685	0.781	0.836	0.587	0.618	0.654	0.723	0.828	0.886	0.621	0.647	0.685

the subject of a triple at position $i + 1$, can better model the path as a sequence of triple embeddings. The preliminary user-item recommendation approach we presented leaves room for future work as, for instance, jointly encoding (and combining) triples and the types of entities within. As an example, we will have $\{$*Triple2Vec*(User, *interacts*, Item), *Triple2Vec*(Item, *sungBy*, Person)$\}$.

6 Related Work

There is a vast body of related research about graph embedding techniques (Cai, Zheng, and Chang 2018; Wang et al. 2017). We group related work in three main categories.

1. Node Embeddings in Homogeneous Graphs. Early work on node embeddings has focused on homogeneous graphs. These techniques sample a set of random walks in the original graph that is fed to a Skip-gram model (Mikolov et al. 2013) to generate the vector representation of nodes so that two nodes that frequently co-occur in a randomly sampled path will have similar embeddings. DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) generates short truncated random walks by uniformly sampling the starting node and the additional nodes from the neighbors of the last node visited. node2vec (Grover and Leskovec 2016) generates biased random walks by using two parameters to control how fast the walk explores and leaves the neighborhood of a node. LINE (Tang et al. 2015) guides the generation of random walks by using 1-hop and 2-hop neighborhoods of nodes as such it learns two different latent representations of nodes. We have compared Triple2Vec with DeepWalk and node2vec. More recent approaches (e.g., (Veličković et al. 2017)) focus on refined notions of neighborhoods.

2. Node Embeddings in Knowledge Graphs. Another strand of research has focused on heterogeneous graphs, where nodes and edges can have different types (Dong, Chawla, and Swami 2017; Fu, Lee, and Lei 2017; Hussein, Yang, and Cudré-Mauroux 2018; Ristoski and Paulheim 2016). Here, the random walk generation for the Skip-gram model has been adapted to consider nodes and edge types. RDF2Vec (Ristoski and Paulheim 2016) focuses on computing node embeddings by using the continuous bag of words or Skip-gram models. It computes two kinds of walks: sub-trees up to a fixed depth k and breadth-first search walks(by uniformly sampling the nodes on the walks among the neighbors). metapath2vec (Dong, Chawla, and Swami 2017) uses metapaths to guide the generation of walks, but it also proposes to use heterogeneous negative samples in the Skip-gram model for learning latent vectors of nodes. Hin2vec (Fu, Lee, and Lei 2017) is an evolution of metapath2vec, which considers multiple metapaths. JUST (Hussein, Yang, and Cudré-Mauroux 2018) provides a sampling strategy that balances the presence of homogeneous and heterogeneous edges along with the node distribution over different domains (i.e., node types) in the generated walks. We have compared Triple2Vec with metapath2vec.

3. Knowledge Graph Representational Learning. These approaches roughly use either the translational embedding TransE (Bordes et al. 2013) or bi-linear models (Nickel, Tresp, and Kriegel 2011) or their extensions. These models jointly learn the vector representations of entities (nodes) and relations (edge labels). Henceforth, the learned representations are used for link prediction (whether there is a relation/edge between two entities). Note that link prediction is the most important task in knowledge graph embedding whereas node classification and clustering are tasks associated with approaches in **1**. and **2**. above. This is reflected in the way these two approaches are trained; while the former are trained using random graph walks, the latter are trained using triples and triples corrupted by replacing the subject and object entities by randomly sampled entities. We have compared Triple2Vec with the recent approaches ConvE (Dettmers et al. 2018) and RotatE (Sun et al. 2019).

All the above approaches have a different departure point. As for **1.** and **2.**, they focus on learning node embeddings only. We mention that approaches like node2vec have proposed ways to learn embeddings for edges in unlabeled graphs as some combination of the embeddings of the edge endpoints (e.g., Hadamard product, average). Although this approach is inherently sub-optimal, when applied to knowledge graphs it will lead to counter-intuitive behaviors. Indeed, all the triples involving the same pair of nodes will be given the same vector representation. We have shown that this approach does not perform comparably to our direct way of computing triple embeddings. As for **3.**, these approaches learn both node and predicate embeddings. Even in this case, we found that when adapting these approaches to compute triple embeddings (e.g., by concatenating the embeddings of the nodes and predicate in a triple) the results are not satisfactory. The reason can be understood with the following example: consider two triples t_1=(s_1, p, o_1) and t_2=(s_1, p, o_2). While our approach is designed to directly provide embeddings for t_1 and t_2, these approaches would consider the embeddings of s_1, o_1, p and s_1, p and o_2, respectively as building blocks for triple embeddings. However, the embeddings of both s_1 and p will be the same in the two final triple embeddings. On the other hand, our approach (working at the level of triple) will be able to better discern the role of both s_1 and p in the two triples.

7 Concluding Remarks and Future Work

We introduced the novel task of learning triple embeddings. While for homogeneous graphs, there have been some sub-optimal proposals (Grover and Leskovec 2016), for knowledge graphs, this problem was never explored. We presented a solution, which builds upon the notion line graph coupled with semantic proximity. Although existing approaches can be adapted we have shown that: *(i)* simply aggregating the embedding of the triple endpoint nodes is problematic when the endpoint nodes are connected by more than one predicate: *(ii)* even concatenating node and predicate embeddings would not work since it is not possible to discriminate the role of the same entities and predicates in all triples they appear. We introduced new applications, that is, triple classification and clustering. We also showed how triple embeddings can be used in downstream applications and discussed the case of user-item recommendations where we found that a state-of-the-art approach can benefit from triple embeddings. Other areas where triple embeddings are useful include fact-checking and reasoning over KG-paths in general. Including novel ways of imposing triple proximity (e.g., via constraints (Minervini et al. 2017)) is in our research agenda.

Acknowledgment

This work was partially supported by the Ministry of Education, University and Research (MIUR) under the grant "Dipartimenti di Eccellenza 2018-2022" awarded to the Department of Computer Science, Sapienza University of Rome.

References

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating Embeddings for Modeling Multi-Relational Data. In *Proc. of International Conference on Neural Information Processing*, 2787–2795.

Cai, H.; Zheng, V. W.; and Chang, K. C.-C. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *Transactions on Knowledge and Data Engineering* 30(9):1616–1637.

Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proc. of the AAAI Conference*, 1811–1818.

Dong, X.; Chawla, N. V.; and Swami, A. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *Proc. of Int. Conference on Information and Knowledge Management*, 135–144.

Fionda, V., and Pirrò, G. 2018. Fact checking via evidence patterns. In *Proc. of International Joint Conference on Artificial Intelligence*, 3755–3761.

Fu, T.-Y.; Lee, W.-C.; and Lei, Z. 2017. Hin2vec: Explore meta-paths in Heterogeneous Information Networks for representation Learning. In *Proc. of Conference on Information and Knowledge Management*, 1797–1806.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks. In *Proc. of Int. Conference on Knowledge Discovery and Data Mining*, 855–864.

Huang, Z., and Mamoulis, N. 2017. Heterogeneous Information Network Embedding for Meta path based Proximity. *arXiv preprint arXiv:1701.05291*.

Hussein, R.; Yang, D.; and Cudré-Mauroux, P. 2018. Are Meta-Paths Necessary?: Revisiting Heterogeneous Graph Embeddings. In *Proc. of Proc. of Conference on Information and Knowledge Management*, 437–446.

Maaten, L. v. d., and Hinton, G. 2008. Visualizing Data Using t-SNE. *J. of Machine Learning Research* 9:2579–2605.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Proc. of Proc. of International Conference on Neural Information Processing*, 3111–3119.

Minervini, P.; Costabello, L.; Muñoz, E.; Nováček, V.; and Vandenbussche, P.-Y. 2017. Regularizing Knowledge Graph Embeddings via Equivalence and Inversion Axioms. In *Proc. of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 668–683.

Nickel, M.; Tresp, V.; and Kriegel, H.-P. 2011. A Three-Way Model for Collective Learning on Multi-Relational Data. In *Proc. of International Conference on Machine Learning*, volume 11, 809–816.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online Learning of Social rRpresentations. In *Proc. of KDD*, 701–710.

Pirrò, G. 2019. Building Relatedness Explanations from Knowledge Graphs. *Semantic Web* 10(6):963–990.

Recht, B.; Re, C.; Wright, S.; and Niu, F. 2011. Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Proc. of Proc. of International Conference on Neural Information Processing*, 693–701.

Ristoski, P., and Paulheim, H. 2016. Rdf2vec: RDF Graph Embeddings for Data Mining. In *Proc. of International Semantic Web Conference*, 498–514.

Sun, Z.; Deng, Z.-H.; Nie, J.-Y.; and Tang, J. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *Proc. of International Conference on Learning Representations*.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. LINE: Large-Scale Information Network Embedding. In *Proc. of Int. World Wide Web Conference*, 1067–1077.

Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; and Bouchard, G. 2016. Complex Embeddings for Simple Link Prediction. In *Proc. of Int. Conf. on Machine Learning*, 2071–2080.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph Attention Networks. In *Proc. of International Conference on Learning Represenation*.

Wang, Q.; Mao, Z.; Wang, B.; and Guo, L. 2017. Knowledge Graph embedding: A Eurvey of Approaches and Applications. *Trans. on Knowledge and Data Engineering* 29(12):2724–2743.

Wang, X.; Wang, D.; Xu, C.; He, X.; Cao, Y.; and Chua, T.-S. 2019. Explainable Reasoning over Knowledge Graphs for Recommendation. In *Proc. of AAAI*, 5329–5336.

Xiao, H.; Huang, M.; and Zhu, X. 2016. TransG: A Fenerative Model for Knowledge Graph Embedding. In *Proc. of Association for Computational Linguistics*, 2316–2325.