From Node Embeddings to Triple Embeddings

Valeria Fionda[1], Giuseppe Pirrò[2]

[1]DeMaCS, University of Calabria, via Pietro Bucci 30B, 87036, Rende, Italy
[2]DI, Sapienza Università di Roma, Piazzale Aldo Moro, 5, 00185 Roma, Italy

Abstract
An extended version of this paper has been published at the the 34th AAAI Conference on Artificial Intelligence (AAAI) with the title "Learning Triple Embeddings from Knowledge Graphs". Graph embedding techniques allow to learn high-quality low-dimensional graph representations useful in various tasks, from node classification to clustering. Knowledge graphs are particular types of graphs characterized by several distinct types of nodes and edges. Existing knowledge graph embedding approaches have only focused on learning embeddings of nodes and predicates. However, the basic piece of information stored in knowledge graphs are triples and thus, an interesting problem is that of learning embeddings of triples as a whole. In this paper we report on Triple2Vec, a new technique to directly compute triple embeddings in knowledge graphs. Triple2Vec leverages the idea of line graph and extends it to the context of knowledge graphs. Embeddings are then generated by adopting the SkipGram model, where sentences are replaced with walks on a wighted version of the line graph.

Keywords
Knowledge Graphs, Triple Embeddings, Line Graph

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 [1]. Several approaches focused on computing node embeddings for *homogeneous graphs* only including one type of edge (e.g., [2], [3]) or knowledge graphs (aka heterogeneous information networks) characterized by several distinct types of nodes and edges [4] (e.g., [5], [6], [7]). There have also been attempts considering *edge embeddings* in homogeneous graphs by applying some operator (e.g., average, Hadamard product) to the embeddings of the edge endpoint nodes [3]. In the knowledge graph landscape, node and predicate embeddings are learned separately and these pieces of information could be aggregate (e.g., [8], [9], [10], [11]).

In this paper we report about a technique to directly learn triple embeddings from knowledge graph, which builds upon the notion of *line graph* of a graph; in particular, we leverage the fact that here triples can be turned into nodes. 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 applying embedding techniques to the nodes of

AIxIA 2021 Discussion Papers
fionda@mat.unical.it (V. Fionda); pirro@di.uniroma1.it (G. Pirrò)
0000-0002-7018-1324 (V. Fionda); 0000-0002-7499-5798 (G. Pirrò)

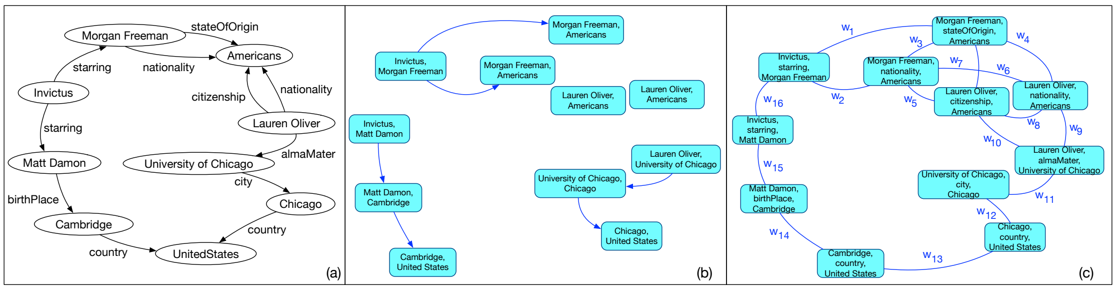

Morgan Freeman
stateOfOrigin
Americans
starring
nationality
Invictus
citizenship
nationality
Lauren Oliver
starring
almaMater
Matt Damon
University of Chicago
city
birthPlace
Chicago
Cambridge
country
country
UnitedStates
(a)
Morgan Freeman, Americans
Invictus, Morgan Freeman
Morgan Freeman, Americans
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.

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.

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. 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. Thus, we introduce a mechanism to weight the edges of $\mathcal{G}_L$ based on predicate relatedness [12]. 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.

Finally, to compute edge embeddings 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 [13], which will give the embeddings of the nodes of the weighted triple line graph that correspond to the embeddings of the original triples.

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. We conclude and sketch future work in Section 6.

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 iff, 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 counter-intuitive 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 states 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 graph* suitable for KGs.

Definition 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* $\mathcal{G}_L$.

We devised an algorithm (outlined in Algorithm 1) to compute the triple line graph $\mathcal{G}_L$ of a knowledge graph G that 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

```
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 $w(n,n') = \texttt{computeEdgeWeight}(n,n')$
12:         end for
13:     end for
14: end for
15: return $\mathcal{G}_L$
```

Algorithm 1: BuildTripleLineGraph (G)

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) based on predicate relatedness (Section 4).

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*; (iv) *computing walks* on the weighted triple line graph; and (v) *computing embeddings* via the Skip-gram model.

Triple Line Graph Edge Weighting. We have mentioned in Section 3 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 [14].

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 $\frac{|E|}{|\{p_i : C_{i,j} > 0\}|}$ [12]. 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.

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., [3]), none of them has tackled the problem of computing triple embeddings.

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 $f : V_L \to R^d$, 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. 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 [3], 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: $p((e_j, e_i) \notin \mathcal{W}) = \sigma(-e_{v_i}^T e_{v_j})$. 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:

$$\mathcal{O}(\theta) = \log \sigma(e_{v_i}^T e_{v_{i+1}}) + \sum_{j=1}^{k} \mathbb{E}_{v_j}[\log \sigma(-e_{v_i}^T e_{v_j})], \tag{2}$$

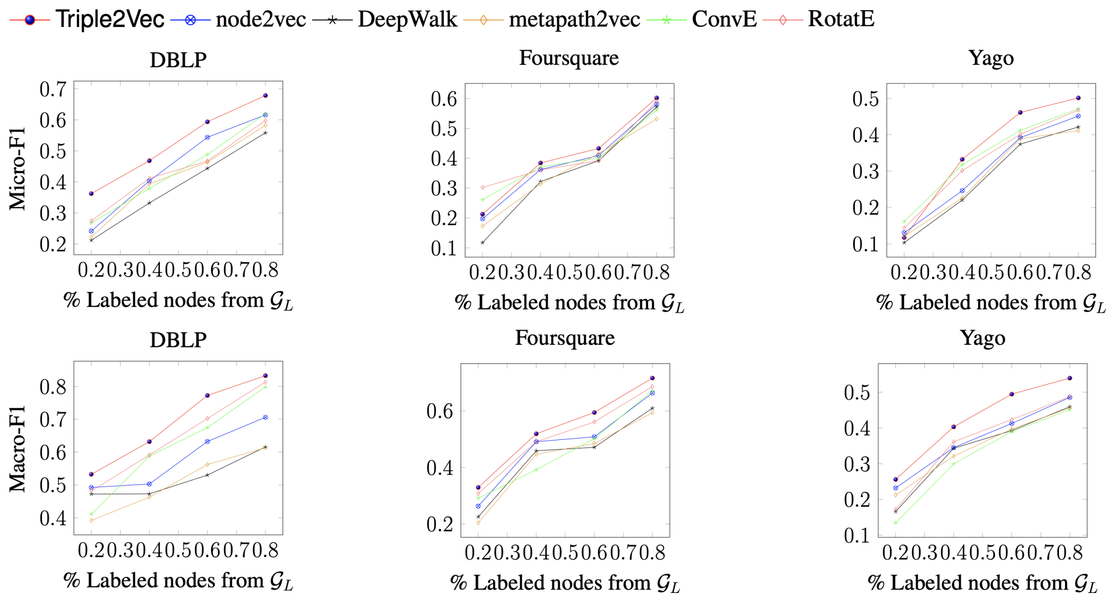


Figure 2: Triple classification results in terms of Micro and Macro F1.

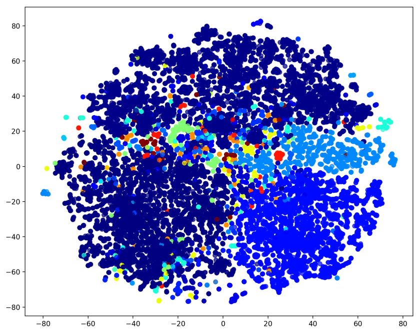
where θ denotes the set of all parameters and k is the number of negative samples. For the optimization of the objective function, we use the parallel asynchronous stochastic gradient descent algorithm [15].

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. In the experiments, we used three real-world data sets. DBLP [16] about authors, papers, venues, and topics, containing ∼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 [7] with ∼30K nodes and ∼83K edges including four different kinds of entities, that is, users, places, points of interests and timestamps where each point of interest has also associated one among 10 labels. Finally, we used a subset of Yago [16] in the domain of movies including ∼22K nodes, ∼89K edges, and 5 predicate types, where each movie is assigned one or more among 5 available labels.

Since there are no benchmarks available for the evaluation of triple embedding approaches, we constructed two benchmarks for *triple classification* and *triple clustering* by using the labels available in the datasets. For DBLP the 4 labels for author nodes have been propagated to paper nodes and from paper nodes to topic and venue nodes. For Yago, the labels for movies have been propagated to actors, musicians and directors. For FourSquare, the 10 labels for points of interest have been propagated to places, users and timestamps. With this reasoning, each node has been labeled with *a subset* of labels and nodes of $\mathcal{G}_L$ (i.e., the triples of G) have been labeled

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


(a) Triple2Vec

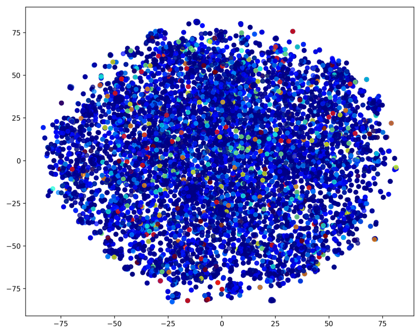

(b) metapath2vec

Figure 3: DBLP triple embedding visualization.

with the union of the labels associated with the triple endpoints. To each subset of labels we assigned a different meta-label.

We compared Triple2Vec to the following two groups of related approaches: *(i)* metapath2vec [6], node2vec [3] implemented in StellarGraph (https://www.stellargraph.io) and DeepWalk [2] configured with the best parameters reported in their respective papers; that compute embeddings for each node only (not for predicates), and a triple embedding was obtained by averaging the embeddings of the triple endpoints; *(ii)* ConvE [9] and RotatE [11] proposed for knowledge graph embedding configured with the best parameters reported in their respective papers and implemented in the pykg2vec (https://github.com/Sujit-O/pykg2vec), that compute embeddings for each node and each predicate, and triple embeddings were obtained by concatenating the embeddings of the triple endpoints and the predicate. For sake of space, 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 [7], 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. 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.

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 [17] 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.

6. Concluding Remarks and Future Work

We discussed a technique to directly learn triple embeddings in knowledge graphs. While for homogeneous graphs, there have been some sub-optimal proposals [3], for knowledge graphs, this problem was never explored. The presented solution builds upon the notion of 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.

References

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

[2] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online Learning of Social rRpresentations, in: Proc. of KDD, 2014, pp. 701–710.

[3] A. Grover, J. Leskovec, node2vec: Scalable Feature Learning for Networks, in: Proc. of Int. Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864.

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

[5] P. Ristoski, H. Paulheim, Rdf2vec: RDF Graph Embeddings for Data Mining, in: Proc. of International Semantic Web Conference, 2016, pp. 498–514.

[6] X. Dong, N. V. Chawla, A. Swami, metapath2vec: Scalable Representation Learning for Heterogeneous Networks, in: Proc. of Int. Conference on Information and Knowledge Management, 2017, pp. 135–144.

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

[8] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, G. Bouchard, Complex Embeddings for Simple Link Prediction, in: Proc. of Int. Conf. on Machine Learning, 2016, pp. 2071–2080.

[9] T. Dettmers, P. Minervini, P. Stenetorp, S. Riedel, Convolutional 2D Knowledge Graph Embeddings, in: Proc. of the AAAI Conference, 2018, pp. 1811–1818.

[10] H. Xiao, M. Huang, X. Zhu, TransG: A Fenerative Model for Knowledge Graph Embedding, in: Proc. of Association for Computational Linguistics, 2016, pp. 2316–2325.

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

[12] G. Pirrò, Building Relatedness Explanations from Knowledge Graphs, Semantic Web 10 (2019) 963–990.

[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed Representations of Words and Phrases and their Compositionality, in: Proc. of Proc. of International Conference on Neural Information Processing, 2013, pp. 3111–3119.

[14] V. Fionda, G. Pirrò, Fact checking via evidence patterns, in: Proc. of International Joint Conference on Artificial Intelligence, 2018, pp. 3755–3761.

[15] B. Recht, C. Re, S. Wright, F. Niu, Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent, in: Proc. of Proc. of International Conference on Neural Information Processing, 2011, pp. 693–701.

[16] Z. Huang, N. Mamoulis, Heterogeneous Information Network Embedding for Meta path based Proximity, arXiv preprint arXiv:1701.05291 (2017).

[17] L. v. d. Maaten, G. Hinton, Visualizing Data Using t-SNE, J. of Machine Learning Research 9 (2008) 2579–2605.