LoGNet: Local and Global Triple Embedding Network

Giuseppe Pirrò(✉)

Department of Computer Science, Sapienza University of Rome, Rome, Italy
pirro@di.uniroma1.it

Abstract. This paper introduces an end-to-end learning framework called LoGNet (Local and Global Triple Embedding Network) for triple-centric tasks in knowledge graphs (KGs). LoGNet is based on graph neural networks (GNNs) and combines local and global triple embedding information. Local triple embeddings are learned by treating triples as sequences. Global triple embeddings are learned by operating on the feature triple line graph $\mathcal{G}_L$ of a knowledge graph $\mathcal{G}$. The nodes of $\mathcal{G}_L$ are the triples of $\mathcal{G}$, edges are inserted according to subjects/objects shared by triples, and node and edge features are derived from the triples of $\mathcal{G}$. LoGNet brings a refreshing triple-centric perspective in learning from KGs and is flexible enough to adapt to various downstream tasks. We discuss concrete use-cases in triple classification and anomalous predicate detection. An experimental evaluation shows that LoGNet brings better performance than the state-of-the-art.

Keywords: Knowledge graphs · Triple embeddings

1 Introduction

Knowledge Graphs (KGs) are organized as a set of facts (or triples) of the form (s, p, o) where the predicate p represents a semantic relation holding between the subject entity s and the object entity o. As an example, the triple (M. Freeman, starring, Invictus) represents the fact that the actor *M. Freeman* was *starring* in the movie *Invictus*. Several approaches have focused on learning representations (aka embeddings), for both entities and predicates, in the form of low-dimensional vectors [5] to support knowledge discovery tasks. The problem of directly computing embeddings of entire triples has received little attention.

Related Work. We identify three main strands of related research. The first concerns node embeddings (e.g., RDF2Vec [27], metapath2vec [12], JUST [17], NESP [7]) and is based on first computing walks in KGs according to different strategies and feed them into language model techniques (e.g., Word2Vec [22]). Node embeddings can then be used in various downstream applications, including node classification and clustering (e.g., [7,17]). The second strand has focused on finding *both entity (node) and predicate (edge) embeddings* with the main

U. Sattler et al. (Eds.): ISWC 2022, LNCS 13489, pp. 336–353, 2022.
https://doi.org/10.1007/978-3-031-19433-7_20

goal to perform link prediction or knowledge graph completion (e.g., TransE [4], ComplexE [31], ConvE [11], RotatE [30]). The third strand leverages Graph neural networks (GNNs) [28] as a model that directly adapts to graphs in a variety of classification and prediction tasks (e.g., node-level, graph-level) and contexts, from drug discovery to neural translation. GNNs can be used to compute node, edge, and graph embeddings. We observe that *the problem of directly computing embeddings of entire triples has received little attention.* One *indirect way* to solve this problem would be to perform some operation (e.g., Hadamard product, concatenation, average) on the embeddings of the subject, object, and predicate in the triple. However, this approach is sub-optimal and fails to capture the essence of triple embeddings for two main reasons. First, it treats subject, predicate, and object (embeddings) as separate elements, disregarding that triples are inherently sequential. The second reason is that approaches based on entity/predicate embeddings aggregation fail to capture correlations among entire triples. Triple2Vec [13] is the only approach we are aware of that directly computes triple embeddings; it leverages the *triple line graph* $\mathcal{G}_L$ of a knowledge graph $\mathcal{G}$ where the nodes of $\mathcal{G}_L$ are the triples of $\mathcal{G}$ with edges between nodes inserted whenever the triples of $\mathcal{G}$ share an endpoint. Then, it uses walks computed on the triple line graph fed to word2vec to compute the embeddings of the nodes of $\mathcal{G}_L$ that correspond to the triples of $\mathcal{G}$.

Limitations of the State-of-the-Art. We identify some potential drawbacks for Triple2Vec: (i) Triple2Vec neither considers node nor edge features that may be derived in a KG, for instance, by looking at the semantics of predicates or node types. Besides, turning triples to nodes via the line graph transformation only considers topological information disregarding semantic relationships between triples; (ii) Triple2Vec is not trained in an end-to-end fashion; embeddings learned by Triple2Vec need to be fed to other learners (e.g., one-vs-rest logistic regressors) for downstream tasks. This approach requires to train additional modules on an objective unrelated to the initial task. *The goal of this paper is to present an end-to-end learning framework to compute triple embeddings.*

Challenges and Contributions. To accomplish the goal mentioned above, we address three main challenges. *The first concerns how to capture triple embedding information.* To solve this challenge, we note that triples in the input knowledge graph $\mathcal{G}$ have an inherently sequential nature and can be seen as three-word sentences; each triple is a sequence of subject→predicate→object (in the forward order) and object→predicate→subject (in the reverse order). We propose to learn *local* triple embeddings by using bidirectional recurrent neural networks [9], a class of neural networks specific for sequence learning. *The second challenge concerns node and edge features.* We have discussed that the state-of-the-art Triple2Vec neither considers node nor edge features. To initialize features one can consider, for instance, node degrees [15], random values [1] or position-based techniques [37]. Despite various approaches, it is unclear which kind of artificial feature initialization works best. To solve this challenge, we leverage semantic information carried by the triples of $\mathcal{G}$ to initialize both node and edge features of the $\mathcal{G}_L$. Specifically, the node features of $\mathcal{G}_L$ will be the local triple

embeddings obtained from $\mathcal{G}$. The edge features are obtained by looking at the relatedness between the triples corresponding to adjacent nodes of $\mathcal{G}_L$. As an example, suppose that in the edge (n_i, n_j) of $\mathcal{G}_L$ node n_i and n_j correspond to the triples $\mathsf{s}_k\mathsf{p}_i\mathsf{o}_j$ and $\mathsf{s}_k\mathsf{p}_k\mathsf{o}_w$ in $\mathcal{G}$, respectively. We can consider a 3-dimensional feature vector where the first dimension represents the relatedness between the predicates [29] p_i and p_k while the second and third dimensions are the relatedness of the subject and object entity node types, respectively. *The third challenge concerns how to combine local, global, and neighbor triple embedding information.* To solve this challenge, we introduce the Local and Global Triple Embedding Network (LoGNet), a novel learning framework based on GNNs. LoGNet's underlying idea is to intertwine information from $\mathcal{G}$ (local triple embeddings) and $\mathcal{G}_L$ (global triple embeddings). In LoGNet message passing and aggregation relies on both node and edge features of the $\mathcal{G}_L$; LoGNet adopts a multi-channel convolution operator that weights the contributions of neighbor nodes to the representation of a target node. LoGNet is flexible enough for a variety of triple-centric downstream applications by providing an appropriate loss function and output layer.

Impact and Applications. Investigating triple-centric applications brings a refreshing perspective to a landscape dominated by node/edge-centric applications. Triple embeddings are good support for any path-based downstream application; here, the intuition is to embed paths as sequences of triple embeddings and then aggregate them. Examples are fact-checking [26] or user-item recommendation [13]. Triple embeddings are useful in sensitive data release scenarios where the same predicate may not be sensitive depending on the subject and object. Consider the triples (Joe, marriedTo, Val) and (Frank, marriedTo, Mary) extracted from a government document. It may be the case that the same predicate marriedTo may be considered sensitive for Val and not for Mary. Therefore, using the same predicate embedding in a data analysis scenario may be insufficient. In this paper, we will focus on *the predicate anomaly problem where both the subject and the object of a triple are legitimate entities, and the potential anomaly resides in the predicate linking them* [18]. Predicate anomaly is a fundamental problem as KGs are traditionally incomplete [10] and can have a considerable amount of incorrect triples [20]. We will show how LoGNet can be customized to tackle this task by adopting a margin-based loss function and an output layer, which returns a plausibility score for each triple.

Outline. We provide some preliminary definitions in Sect. 2. We outline in Sect. 3 LoGNet, which is a generic learning framework for triple-centric tasks in KGs. Section 4 shows how LoGNet can be adapted to solve the problem of anomalous predicate detection from the novel perspective of triple embeddings. In Sect. 5, we discuss an experimental evaluation. We conclude in Sect. 6.

2 Definitions and Background

A Knowledge Graph ($\mathcal{G}$) is a labeled directed multigraph $(\mathcal{V}_G, \mathcal{E}_G, \mathcal{T}_G)$ where $\mathcal{V}_G$ is a set of uniquely identified nodes representing entities (e.g., D. Lynch),

$\mathcal{E}_G$ a set of predicates (e.g., director) and $\mathcal{T}_G$ a set of triples of the form (s, p, o) representing directed labeled edges, where $\mathsf{s}, \mathsf{o} \in \mathcal{V}_G$ and $\mathsf{p} \in \mathcal{E}_G$. Often, information in a $\mathcal{G}$ is organized according to an underlying schema defining, for instance, the types of the entity nodes (e.g., Person) and domain and range definitions for predicates stating what type of entity one should expect ad subject and object of a triple. We denote by $\mathsf{type}(\mathsf{e})$ the set of types of an entity e.

Entity and Predicate Embeddings. KG embedding approaches focus on learning vector representations $\mathbf{e} \in \mathbb{R}^{d_e}$ for each entity $\mathsf{e} \in \mathcal{V}_\mathcal{G}$ and possibly predicate embeddings $\mathbf{p} \in \mathbb{R}^{d_p}$ for each predicate $\mathsf{p} \in \mathcal{E}_\mathcal{G}$. Typically, KG embedding systems include an *embedding component* and a *scoring component*. The former maps each entity to its corresponding embedding while the latter learns a scoring function $\mathbf{f}$: $\mathcal{V}_\mathcal{G} \times \mathcal{E}_\mathcal{G} \times \mathcal{V}_\mathcal{G} \rightarrow \mathbb{R}$ where $\mathbf{f}(\mathsf{s}, \mathsf{p}, \mathsf{o})$defines the score of the triple (s, p, o). Embeddings are obtained by defining a loss function (e.g., Logistic Loss) and solving an optimization problem where the score of a positive triple (s, p, o) is to be higher than that of a (corrupted) negative triple. As an example, in the popular TransE model [4], where predicates are modeled as vector translations, the scoring function is $s_\mathsf{p} = d(\mathsf{s} + \mathsf{p}, \mathsf{o})$ where d is the euclidean distance. Other approaches capture more refined relations (see [5] for a survey).

Graph Neural Networks. We now introduce graph neural networks in a general form. To keep the presentation concise, we focus on undirected and unlabeled graph. Let $(\mathcal{V}_G, \mathcal{E}_G)$ be a graph with $N{=}|\mathcal{V}_G|$ nodes $v_i \in \mathcal{V}_G$, and edges $(e_i, e_j) \in \mathcal{E}_G$. Given a node u the set of neighbours is denoted by $\mathcal{N}(u)$. We denote by A the adjacency matrix where $A \in \mathbb{R}^{N \times N}$. Moreover, the matrix $H^{(0)} \in \mathbb{R}^{N \times D^0}$ is the initial node feature matrix and $h_i^{(0)}$ denotes the $D^{(0)}$-dimensional feature vector of node v_i. A GNN can be represented as:

$$h_i^{(l+1)} = \sigma\big(\sum_{j=1}^{N} \alpha(v_i, v_j) \cdot h_j^{(l)} \cdot W^{(l)}\big), \quad l = 0, \ldots, L-1 \tag{1}$$

where $h_i^{(l)}$ is the embedding of node v_i at layer l. Moreover, $\alpha{=}(v_i, v_j) \in \mathbb{R}^{N \times N}$ is a weight matrix, $W^{(l)} \in \mathbb{R}^{D^{(l)}} \times D^{(l+1)}$ is the transformation matrix at layer l, and σ is the activation function. The weight $\alpha(v_i, v_j)$ (abbreviated as $\alpha_{i,j}$) is non-zero if the node v_j is a direct neighbor of node v_i, that is, $v_i \in \mathcal{N}(i)$. Different ways have been proposed when it comes to the weight matrix α. As an example, Kipf and Welling [19] define *fixed* weights as $\alpha = \tilde{D}^{-1}$ or $\alpha = \tilde{D}^{-1}\tilde{A}D^{-1}$, respectively, where $\tilde{A} = A + I$, and $\tilde{D}$ is the diagonal degree matrix of A. More sophisticated approaches, instead of assigning fixed weight, try to learn them via attention coefficients [32]. GATs [32] learn weights via attentive functions of the form $\alpha_{i,j} = \frac{\tilde{\alpha}_{i,j}(\theta)}{\sum_{v_k \in \mathcal{N}(i)} \tilde{\alpha}_{i,j}(\theta)}$; here, unnormalized attentions $\tilde{\alpha}_{i,j}(\theta)(exp(ReLU(\alpha^T[Wh_i||Wh_j]))$, are parametrized by $\theta = \{\alpha\}$ with $||$ denoting concatenation. For attention networks, the weights that are learned $\alpha_{i,j} \approx \tilde{\alpha_{i,j}}$ can be evaluated only given the unnormalized neighborhood weights.

3 The LoGNet Framework

We now describe the design of a learning framework called LoGNet to directly compute triple embeddings from a knowledge graph combining local and global triple embedding information. The intuition is that local triple embeddings can be complemented with global embeddings derived by scrutinizing the feature triple line graph structure. This structural information is independent of node features and can be derived solely based on the links between nodes of the feature triple line graph. Finally, integrating local embeddings with connectivity information is crucial to fully capture the essence of each node (triple) and thus of its embedding [3]. LoGNet computes local embeddings obtained from the triples of an input $\mathcal{G}$. However, local embeddings do not consider triples from a global perspective; notably, they do not consider dependencies among entire triples. We introduce an alternative view of triple information and discuss a GNN-based triple embedding module to cope with this issue.

3.1 Local Triple Embeddings

Triple Embedding via Aggregation. One simple way to compute triple embeddings is by aggregating the embeddings of the entities and predicate within. Given a triple $\mathsf{t} = (\mathsf{s}, \mathsf{p}, \mathsf{o}) \in \mathcal{G}$, the idea is to define a mapping function Emb_L: $\mathsf{Agg}(\mathsf{Emb}(\mathsf{s}), \mathsf{Emb}(\mathsf{p}), \mathsf{Emb}(\mathsf{o})) \mapsto \mathbb{R}^d$, where $\mathsf{Emb}(\cdot)$ is an embedding function that maps entities or predicates into a D-dimensional vector. $\mathsf{Emb}(\cdot)$ can be instantiated with a variety of techniques such as TransE [4], RotatE [30], and DistMul [36] that return embeddings for s, p, and o separately. As an example, TransE to learn entity and predicate embeddings utilizes the triple implausibility score $\mathbf{s} + \mathbf{p} \approx \mathbf{o}$ while DistMul considers $\langle \mathbf{p}, \mathbf{s}, \mathbf{o} \rangle$ where $\langle \cdot \rangle$ denotes the generalized dot product with $\mathbf{s}$, $\mathbf{p}$, $\mathbf{o} \in R^k$. Hence, triple embeddings can be obtained via an aggregation function $\mathsf{Agg}(\cdot)$. Examples of aggregation include the average, maxpool, and Hadamard product.

Triple Embedding as Sequence Encoding. The approach based on aggregation suffers from some drawbacks. First, the usage of aggregation functions (e.g., mean, max) does not properly discern the contribution of each component of a triple to the final triple embedding. Second, as the approaches implementing the $\mathsf{Emb}(\cdot)$ return an embedding for each entity and predicate in $\mathcal{G}$, triples sharing the same entity or predicate will share part of the final embedding, which does not allow to obtain a fine-grained triple embedding representation. Third, the approach based on aggregation mostly ignores the sequential nature of a triple, which has a relevant role in directed graphs as KGs. Therefore, we consider a second approach to obtain local triple representations, which treats a triple as a directed and ordered sequence $\mathbf{s} \rightarrow \mathbf{p} \rightarrow \mathbf{o}$ composed of three steps (i.e., s, p, o). To deal with this sequence of elements, we employ Recurrent Neural Networks and, in particular, Bidirectional Gated Recurrent Unit (BiGRU) [8]. This architecture offers performance comparable to the LSTM model [16] with the advantage of being more computationally efficient [9].

To fulfill our ultimate goal of learning triple embeddings, we need to consider the context of a triple in terms of neighbor triples. Missing contextual information will fail to capture potential correlation and dependencies among entire triples. Therefore, we introduce a global triple embedding computation mechanism in the next section.

3.2 Global Triple Embeddings

Correlation and dependencies among entire triples can be captured by looking at triples from a global perspective, where triples become first-class citizens.

Feature Line Graph. The notion of the line graph of a graph is well-known in graph theory [34]. Given an undirected graph $\mathcal{G} = (\mathcal{V}_G, \mathcal{E}_G)$, the corresponding line graph $\mathcal{G}_L$ is such that: (i) each node of $\mathcal{G}_L$ represents an edge of $\mathcal{G}$; (ii) two vertices of $\mathcal{G}_L$ are adjacent if, and only if, their corresponding edges in $\mathcal{G}$ have a node in common. This notion has been extended to multigraphs and directed graphs. The multigraph extension adds a different node in the line graph for each edge of the original multigraph. If the graph $\mathcal{G}$ 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 $\mathcal{G}$ and its edges represent two-length directed paths in $\mathcal{G}$. Triple2Vec [13] used the triple line graph of a knowledge graph. However, we note that such a triple line graph is unlabeled, and neither nodes nor edges are endowed with features, thus making difficult the usage of deep learning techniques at their full potential. For example, in the absence of node features, GNNs fail to differentiate between similar graph sub-structures within graphs [35]. A workaround would be to consider one-hot encoding instead of features. However, this will hinder using the model on new nodes. Other approaches could include the usage of random values [1] or positional features [37]. However, there is no transparent approach that works best in all scenarios. We overcome the lack of features in $\mathcal{G}_L$ by leveraging information from the triples of the original graph $\mathcal{G}$:

1. As nodes of $\mathcal{G}_L$ correspond to triple of $\mathcal{G}$, we can consider local triple embeddings (see Sect. 3.1) as the *initial* features of the nodes of $\mathcal{G}_L$.
2. An edge (n_i, n_j) of $\mathcal{G}_L$ links the two corresponding triples t_i and t_j in $\mathcal{G}$. However, each of such triple implicitly includes semantic information deriving, for instance, from the subject and object entity types or the type of predicate linking subject and object. As an example, the triples (M. Freeman, starring, Invictus) and (M. Damon, starring, Invictus) taken from the DBpedia knowledge graph tell us that type(M. Freeman) = {Person}, type(Invictus) = {Film} and that domain(starring) = Actor and range(starring) = Work. Our proposal is to introduce, for each edge $v_{i,j} \in \mathcal{E}_L$ a P-dimensional feature vector $\mathbf{v}_{i,j} \in \mathbb{R}^P$ where each dimension captures a different relatedness perspective between the elements of the triples t_i (corresponding to node $n_i \in \mathcal{V}_L$) and t_j (corresponding to the node $n_j \in \mathcal{V}_L$).

Therefore, we introduce a triple line graph with node and edge features that we refer to as *feature triple line graph*.

Definition 1. *(Feature Triple Line Graph). Given* $\mathcal{G} = (\mathcal{V}_G, \mathcal{E}_G, \mathcal{T}_G)$ *with* $N = |\mathcal{T}_G|$ *triples, the associated features triple line graph* $\mathcal{G}_L$ *is a graph* $(\mathcal{V}_L, \mathcal{E}_L, \mathcal{X}_V, \mathcal{X}_E)$*, where* $t_i \in \mathcal{T}_G \mapsto n_i \in \mathcal{V}_L$ *and* $|\mathcal{V}_L| = N$*. There exists an edge* $(i,j) \in \mathcal{E}_L$ *between* $n_i \leftrightarrow t_i = (s_1, p_1, o_1) \in \mathcal{T}_G$ *and* $n_j \leftrightarrow t_j = (s_2, p_2, o_2) \in \mathcal{T}_G$ *if* $\{s_1, o_1\} \cap \{s_2, o_2\} \neq \emptyset$*. Node features are represented by a matrix* $\mathcal{X}_V = N \times F$*, where* $\mathcal{X}_V(i,j)$ *gives the* j*-th entry of the* F*-dimensional feature vector of the* i*-th node in the feature triple line graph. Edge features are represented via a* $\mathcal{X}_E = N \times N \times P$ *tensor;* $\mathcal{X}_E(i,j,p)$ *is the* p*-th channel of the* P*-dimensional vector of the edge* (i,j)*.*

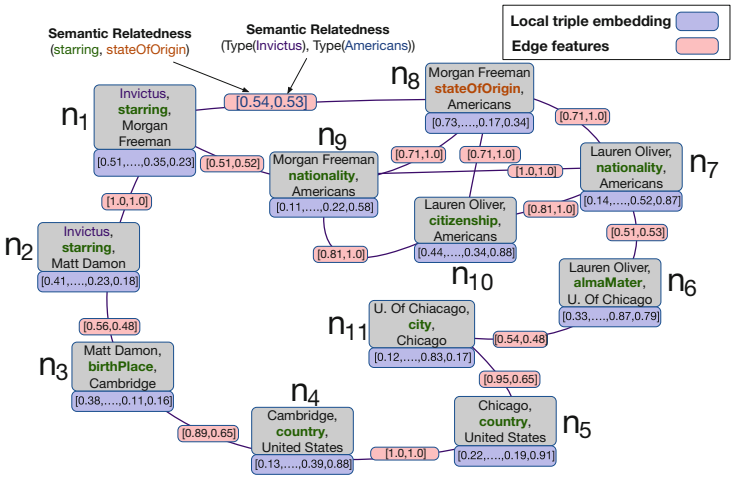

Fig. 1. Feature triple line graph.

Figure 1 shows an excerpt of feature triple line graph. We observe that nodes are endowed with features in the form of local triple embeddings, that is, embeddings computed by only looking at the triple elements (see Sect. 3.1). As for the edges, the figure shows a two-dimensional feature vector for each edge. The first dimension maintains the semantic relatedness between the predicates in the two neighbor nodes computed considering predicate co-occurrences in knowledge graph triples [25]. As an example, the relatedness between starring and stateOfOrigin is 0.54 while that between stateOfOrigin and nationality 0.71. The second dimension considers the relatedness between the type (e.g., Actor, Movie, Place) of entities not shared by neighbor nodes. As an example, for the node n_1 (Invictus, starring, M. Freeman) and n_2 (M. Freeman, stateOfOrigin, Americans) the not shared entities are Invictus[1] and Americans[2] whose type in the DBpedia KG are Film and Country, respectively. The semantic relatedness between these

[1] https://dbpedia.org/page/Invictus_(film).
[2] https://dbpedia.org/page/Americans.

types is 0.53 [29]. Note that the edge feature vector can be further extended to include additional dimensions considering, for instance, the cosine similarity between local triple embeddings or topological information such as the difference between node degrees. Going from the knowledge graph $\mathcal{G}$ to a feature-rich triple line graph $\mathcal{G}_L$, where triples are first-class-citizens, paves the way toward designing learning architectures that can leverage both node features, that in our context represent local triple embeddings obtained from $\mathcal{G}$, and edge features that capture different relatedness perspective between adjacent triples.

Triple Embeddings via Node and Edge Features. We introduced in the previous section a novel representation of the original knowledge graph $\mathcal{G}$ called feature triple line graph $\mathcal{G}_L$. We are now ready to show how LoGNet learns triple embeddings by intertwining information from $\mathcal{G}$ and $\mathcal{G}_L$. LoGNet is based on GNNs that learn node representations by recursively aggregating and transforming features of neighbor nodes [15]. In particular, as edge feature vectors in $\mathcal{G}_L$ include D-dimensions, LoGNet performs a separate weighted convolution operation for each dimension; the i-th feature value of dimension d is used to weight the contributions of node neighbors on that dimension. Concerning the general form of GNN outline in Sect. 2, we shall now make explicit the multi-dimensional edge feature vectors present in the feature triple line graph. For each dimension d of edge features, we consider the following formulation:

$$\hat{H}^{(l,d)} = \sigma\big(\tilde{E}_{i,j,d} \cdot H^{(l)} \cdot W^{(l,d)}\big), \tag{2}$$

where $\hat{H}^{(l,d)}$ denotes the matrix of embeddings at level l on dimension d, $\tilde{E}_{i,j,d}$ is the convolution coefficient matrix, $H^{(l)}$ the hidden state at layer l and $W^{(l,d)}$ a weight matrix. The D-dimensional information is is aggregated as follows:

$$H_i^{(l+1)} = \sigma\Big[\bigoplus_{d=1}^{D} (\hat{H}^{(l,d)}) W_{\oplus}^{(d)}\Big] \tag{3}$$

where $\bigoplus$ is an aggregation function (e.g., average) and $W_{\oplus}^{(d)}$ learnable weights.

Edge Feature Aggregation. Edge features represent an important element of innovation for LoGNet and can be seen as driving a sort of edge-centered attention mechanism. Each of the D edge features can be seen as a different attention coefficient. Multiple ways (instantiations of the function $\oplus$) can be used to aggregate edge features along the D dimensions. When considering aggregation based on sum, that is, $\oplus = \sum$ we obtain:

$$H_i^{(l+1)} = \sigma\Big[\sum_{d=1}^{D} (\hat{H}^{(l,d)}) \cdot W_{sum}^{(d)}\Big] = \sigma\Big[\sum_{d=1}^{D} \Big(\sigma\Big(\sum_{j \in \mathcal{N}(i)} e_{i,j,d} \cdot h_j^{(l,d)} \cdot W^{(l,d)}\Big)\Big) \cdot W_{sum}^{(d)}\Big]$$

We underline that the architecture of LoGNet differs from the classical GNN models in one central respect. The GNN adjacency matrix A is either a binary matrix denoting node adjacency (as in GAT) or a positive matrix with only *one dimension* capturing edge features (as in GCN). Contrarily, the LoGNet model builds upon a D dimensional edge feature matrix obtained by investigating different types of relatedness between triples in the feature triple line graph.

4 Anomalous Predicate Detection

We apply the LoGNet framework to the task of anomalous predicate detection [18]. Given a triple (s, p, o), the goal is to check whether the predicate p correctly models the relation between the subject s and the object o. As an example, (I' m a looser, recordedIn, Abbey Road) would seem correct. However, when contextualizing the triple, by looking at the neighbor triples, it is immediate to see that the fact refers to the wrong entity Abbey Road; it refers to the street and not to the recording studio Abbey Road Studio [21]. This example shows the importance of considering both a local perspective (i.e., among the triple elements) and a global (i.e., wrt neighbor triples) to have a more refined assessment of the plausibility of the predicate.

Problem 1 *(Anomalous Predicate Detection). Given* $G = (V_G, E_G, T_G)$ *and the set of triple embeddings* $\boldsymbol{T_G}$ *associated with the triples* T_G*, our goal is to devise a scoring function* $\boldsymbol{f}_A : (\mathsf{s}, \mathsf{p}, \mathsf{o}) \mapsto \mathbb{R}$*, which give an a triple* $t \in T_G$ *assigns a plausibility score to the predicate* p*.*

This section shows how LoGNet can be adapted to tackle this problem. *The main challenge that arises concerns how to combine local and global triple representations to find the plausibility of a predicate.* We introduce a local and global predicate plausibility score and optimize them to solve this challenge.

Local Plausibility Measure. To assess the plausibility of a predicate from a local perspective, we can readily use any of the existing scoring functions available from knowledge graph embedding techniques [5]. By considering TransE, the local plausibility of a triple can be measured as:

$$c_l(\mathbf{s}, \mathbf{p}, \mathbf{o}) \;=\; \|\mathbf{h}_s + \mathbf{h}_p - \mathbf{h}_o\|_2 \tag{4}$$

where $\mathbf{h}_s$, $\mathbf{h}_p$, and $\mathbf{h}_o$ are either the embeddings of the triple elements found by the $\mathsf{Emb}(\cdot)$ function or the hidden representations of the triple elements obtained from the BiGRU. The learning model can spot anomalous predicates from a local perspective by minimizing the above equation. Nevertheless, this approach does not consider the context of a triple in terms of neighbor predicates, which can fail to understand correlation and influence among triples. To this end, we also introduce a global plausibility measure.

Global Plausibility Measure. We compare information resulting from local triple embeddings, computed from $\mathcal{G}$, and global triple embeddings, computed from the feature triple line graph $\mathcal{G}_L$, to improve the overall plausibility check. In particular, minimizing the difference between local and global plausibility can point out anomalous predicates. More formally:

$$c_g(\mathbf{s}, \mathbf{p}, \mathbf{o}) \;=\; \|\mathbf{z} - \mathbf{h}\|_2 \tag{5}$$

where $\mathbf{z}$ is the global triple embedding and $\mathbf{h}$ the local one.

Joint Optimization. We define the plausibility as a linear combination of the local and global plausibility:

$$c(\mathsf{s},\mathsf{p},\mathsf{o}) \;=\; \alpha c_l + \beta c_g \tag{6}$$

where the hyper-parameter α and β with β =(1-α) weight the importance of the two scores normalized. To optimize the model, we leverage a margin based loss function to distinguish between positive triples and negative ones:

$$\mathcal{L} \;=\; \sum_{(\mathsf{s},\mathsf{p},\mathsf{o})\in T_G^+} \sum_{(\mathsf{s}',\mathsf{p},\mathsf{o}')\in T_G^-} \Big[\theta + c(\mathsf{s},\mathsf{p},\mathsf{o}) - c(\mathsf{s}',\mathsf{p},\mathsf{o}')\Big] \tag{7}$$

where $\theta > 0$ is the margin hyper-parameter, T_G^+ is the set of positive triples, and T_G^- is the set of negative triples. At this point, with the model trained, we can assess the anomaly of a predicate in a triple (s, p, o) as follows:

$$P_s(\mathsf{s},\mathsf{p},\mathsf{o}) \;=\; c(\mathsf{s},\mathsf{p},\mathsf{o}) \tag{8}$$

5 Experiments

We now report on the evaluation and comparison with related work in two tasks: anomalous predicate detection and triple classification. LoGNet[3] has been implemented using the DGL[4] library. We used Adam as an optimizer with a fixed batch size of 512 and initialized all model parameters via the Xavier initializer. We conducted a grid search to set the hyper-parameters learning rate, the weight of the plausibility scores, and the margin to their best values. All experiments have been conducted on an RTX6000 GPU and are the average of 5 runs (95% c.i). The goal of the evaluation is to answer the following research questions: **Q1**: How does LoGNet compare to the state-of-the-art in anomalous predicate detection? **Q2**: How does LoGNet compare with the state-of-the-art Triple2Vec? **Q3**: What is the impact of the plausibility on the quality of triple embeddings?

Datasets. For anomalous predicate detection, we used the following datasets: **NELL** [6]: it includes ∼75K entities, 200 predicates and ∼308K triples; **DBPEDIA** (DBP) [29]: it is a KG extracted from Wikipedia. This is a subset of DBpedia with neither typing information nor literals. It includes ∼2M entities, 661 predicates and ∼1.2M triples; **DBPEDIA1M** (DBP1M): a subset of the dataset in [29], which includes 1M entities. For triple classification, we considered **DBLP** [33]: this is a subset of the DBLP database containing information about authors, papers, venues, and topics. Labels are provided for authors that are assigned one among four labels (i.e., database, data mining, machine learning, and information retrieval); **Foursquare** [17]: this dataset includes information about users, places, points of interests, and timestamps. Labels are available for points of interest that are given one among ten labels; **Yago** [33]: this dataset is a subset of the Yago KG[5] focused on the domain of movies. Here, labels are available for movies assigned one or more among five available labels.

[3] https://github.com/giuseppepirro/lognet.
[4] https://www.dgl.ai.
[5] http://yago-knowledge.org/.

5.1 Q1: Anomalous Predicate Detection

We evaluate the performance of LoGNet as compared to the state-of-the-art. We note that this paper does not aim to tackle anomalous predicate detection specifically; we aim to show novel applications of triple embeddings. We considered the following approaches: RotatE [30] and ConvE [11] node/predicate embeddings; KGist [2]: this approaches leverages rules to rule out incomplete and erroneous information in KGs; KGTtm [18]: this approach introduces a triple trustworthiness measure based on semantic information derived from triples along with global information; KBAT [23]: this approach uses an attention mechanism to capture features of the neighborhood of entities; Triple2Vec: we fed the triple embeddings obtained by Triple2Vec along with the labels to a one-vs rest logistic regressor. For LoGNet, we consider a three-layer model in two variants: (i) LoGNet_E where local triple embeddings were obtained by concatenating the embeddings of the subject, predicate, and object obtained via ConvE[6]; (ii) LoGNet_G where local triple embeddings were obtained via BiGRU. Moreover, we considered a 3-dimensional edge feature vector including predicate relatedness [29], subject and object type relatedness. We used $d = 128$ as embedding dimension.

Experimental Setting and Metrics. We used the NELL and DBpedia KGs. As there is no explicit information about anomalous predicates, we assumed that all triples available were correct and assigned them the label 1. To generate negative examples that were given a label 0, we adopted the same method as the state-of-the-art [18]. Given the true triple (Newton, nationality, England), a potential negative triple is (Newton, nationality, American) rather than (Newton, nationality, Google), which have been obtained by randomly replacing the object of the original triple. We considered different corruption percentages, that is, {0.05%, 1%, 2%, 3%, 5%} of the available true triples. The goal of the experiments is to identify triples that include anomalous predicates. To evaluate the performance of the systems, as done in the similar task of detecting incorrect facts (e.g., [29]), we considered the AUC score, which is useful to express the probability that a triple including a correct predicate receives a higher score than a triple including an anomalous one.

Results. From the results in Table 1 we observe: (1) Triple2Vec and LoGNet consistently outperform the competitors in the larger datasets (DBpedia, NELL). As ConvE, RotatE, KBAT, we observe the worst-performing results. The possible reason for such a behavior is that these systems were originally designed to tackle the KG completion task, different from anomalous predicate detection. Indeed, while KG completion aims to understand which part of a triple is missing, anomalous predicate detection is concerned with understanding whether the predicate in a triple makes sense. This underlines two aspects: (i) the need for specific techniques to face this task; (ii) the need to define triple embedding mechanisms alternative to those based on the aggregation of triple elements; (2) Approaches like KGTm and KGist that were designed to detect triple anomaly

[6] With RotatE and TransE we obtained inferior results.

Table 1. Anomalous predicate detection. The best AUC values are reported in bold.

KG	% Corr.	ConvE	RotatE	KGTtm	KBAT	KGist	T2Vec	LNet$_E$	LNet$_G$
DBP	0.05%	0.532	0.542	0.674	0.553	0.532	0.678	0.679	**0.684**
	1%	0.526	0.534	0.671	0.550	0.534	0.675	0.680	**0.691**
	2%	0.524	0.525	0.670	0.546	0.542	0.672	0.672	**0.687**
	3%	0.513	0.521	0.667	0.538	0.560	0.668	0.668	**0.671**
	5%	0.501	0.513	0.662	0.535	0.561	0.670	0.675	**0.680**
DBP1M	0.05%	0.543	0.567	0.687	0.561	0.547	0.6912	0.696	**0.702**
	1%	0.536	0.560	0.671	0.560	0.541	0.687	0.690	**0.696**
	2%	0.531	0.556	0.676	0.559	0.550	**0.678**	0.669	0.671
	3%	0.530	0.542	0.678	0.551	0.546	0.6751	0.677	0.679
	5%	0.526	0.534	0.674	0.541	0.576	0.671	0.673	0.681
NELL	0.05%	0.531	0.534	0.614	0.529	0.54	0.6214	0.621	**0.631**
	1%	0.5301	0.532	0.623	0.546	0.543	0.6245	0.625	**0.632**
	2%	0.523	0.521	0.638	0.543	0.542	0.641	0.631	**0.647**
	3%	0.513	0.516	0.622	0.526	0.552	0.623	0.624	**0.631**
	5%	0.502	0.512	0.637	0.518	0.553	0.6401	0.641	**0.655**

perform better than ConvE, RotatE, and KBAT. However, approaches based on triple embeddings performed consistently better. The reason may be the usage of the line graph construction. This novel structure plays a crucial role in better capturing the contextual structure of a triple wrt neighbor triples compared to learned rules or paths used by KGTtm and KGist. Moreover, the interplay between the local triple representation, learned by treating triples as bidirectional sequences, and the global triple representation provides a fine-grained mechanism to spot predicate anomalies. We observe that in the smallest dataset DBpedia1M, in one case Triple2Vec performs negligibly better than LoGNet; (3) Comparing LoGNet$_E$ and LoGNet$_G$, we observe that the latter performs consistently better. The reason for this behavior is that in the first case, local triple embeddings, representing the initial features of the nodes, are learned by aggregating the embeddings of the element of a triple. Consequently, triples sharing entities and predicates will also share portions of the local triple embeddings. On the other hand, in LoGNet$_G$ local triple embeddings are learned not only by considering the sequential nature of triples but also by the fact that a triple can be read in both forward and backward directions; (4) Comparing Triple2Vec and LoGNet both based on triple embeddings, we observe that the former performs worse than LoGNet. Although Triple2Vec deals with triples as a whole, it cannot spot anomalous predicate at a finer-grained level as LoGNet. This may be because LoGNet adopts a completely different approach based on a joint learning model leveraging local and global triple representations using node and edge features. We observe that triple embeddings offer good support for detecting anomalous predicates. The local triple representation obtained by modeling a triple as a sequence and the global representation constructed on the feature triple line graph where both nodes and edges have features and the context of a triple is obtained from neighbor triples is generally a valid alternative compared to approaches using rules or paths.

5.2 Q2: LoGNet vs Triple2Vec

We now shed more light on the differences between Triple2Vec and LoGNet. We want to answer the following questions: (i) how does LoGNet perform when using triple embeddings learned via Triple2Vec instead of local triple embeddings? (ii) how does the semantics relatedness-based weighting mechanism of the edges of the triple line graph used by Triple2Vec compare with LoGNet's approach?

Experimental Setting. We considered two variants of LoGNet. The first, denoted as LoGNet_V, leverages embeddings learned by Triple2Vec instead of local triple embeddings learned via BiGRU and still uses a 3-dimensional edge feature vector. In the second variant, denoted as LoGNet_W, instead of using the 3-dimensional feature vector, we only consider the relatedness between the predicates of neighbor triples (hence a 1-d feature vector).

Results. We report in Table 2 results for the anomalous predicate detection task and refer to LoGNet_G as LoGNet. We make the following observations: (1) The usage of triple embeddings learned by Triple2Vec instead of local triple embeddings in LoGNet does not bring any tangible benefit. LoGNet_V performs slightly worse than LoGNet in all experiments. The downside of using this approach is that it requires paying the training time cost for both Triple2Vec and LoGNet. Moreover, learning triple embeddings by looking at triples from their sequential perspective has an important role in the overall quality of triple embeddings; (2) Using a 1-d edge feature vector downgrades the performance of LoGNet. We also observe that LoGNet_W performs worse than Triple2Vec (Table 1). The reason for this behavior may be found in the fact that Triple2Vec needs weights to find high-quality walks on the triple line graph that can correctly model node neighborhoods. The GNNs setting, where neighbor information is obtained via a message-passing scheme, brings some improvement.

Table 2. Variants of LoGNet.

KG	% Corr. triples	LoGNet	LoGNet_V	LoGNet_W
DBP	0.05%	0.6842	0.6832	0.6124
	1%	0.6912	0.6879	0.6613
	2%	0.6873	0.6823	0.6731
	3%	0.6712	0.6689	0.6612
	5%	0.6803	0.6734	0.6352
DBP1M	0.05%	0.7022	0.6987	0.6825
	1%	0.6967	0.6823	0.6742
	2%	0.6712	0.6711	0.6531
	3%	0.6790	0.6732	0.6643
	5%	0.6816	0.6789	0.6703
NELL	0.05%	0.6312	0.6235	0.6124
	1%	0.6321	0.6256	0.6013
	2%	0.6476	0.6342	0.6235
	3%	0.6311	0.6211	0.6134
	5%	0.6553	0.6478	0.6391

Results on Triple Classification. We also compared LoGNet and Triple2Vec on the triple classification task in terms of Micro-F1 and Macro-F1 scores following the methodology described in Triple2Vec [13]. We considered the following competitors: metapath2vec [12], node2vec [14], and DeepWalk [24] 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 using the *Hadamard operator over the embeddings of the triple endpoints* as it was the best performing; ConvE [11] and RotatE [30] configured with the best parameters reported in their respective paper and implemented. Triple embeddings were obtained by concatenating the embeddings of the triple endpoints and the predicate embedding. Figure 2 reports the results. We observe that the approaches based on triple embeddings consistently outperform 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 and previously used by Hussein et al. [17] while being able to capture node embeddings, fail short in capturing triple embeddings.

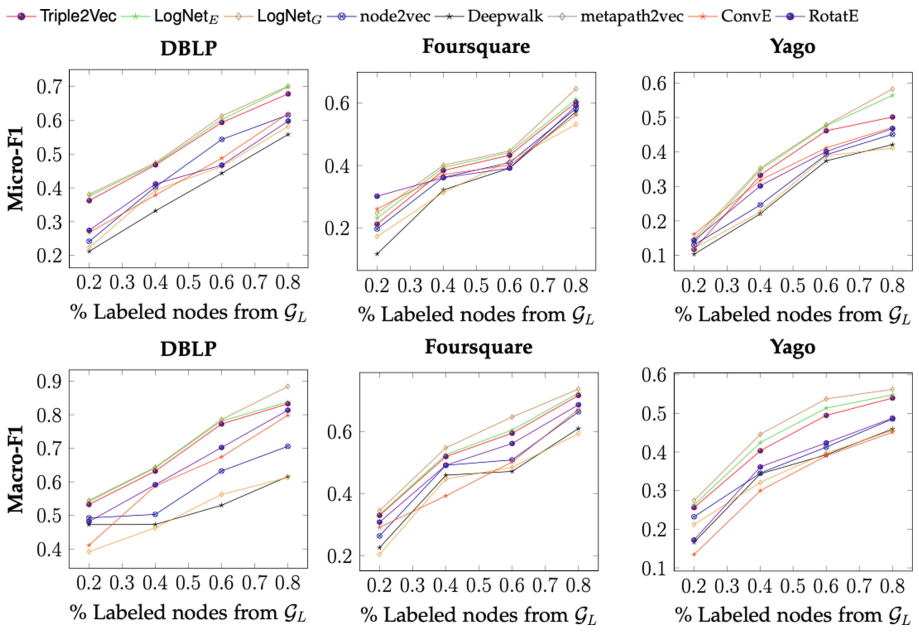

Fig. 2. Results on triple classification.

Moreover, for triple embeddings obtained via aggregation, we can see that the performance is even worse than those obtained by metapath2vec, node2vec, and Deepwalk, which did not consider the embeddings of predicates to compute triple embeddings. This may be due to two main reasons. First, the goal of these approaches is to learn entity and predicate embeddings for the link prediction task. Hence, the concatenation of entity and predicate embeddings does not correctly capture triple embeddings. Second, as these approaches compute a single predicate and node embedding, these embeddings will be shared by all

triples in which they appear. As an example, the embeddings of the two triples (s, p, o) and (s, p, q) will only differ in the concatenation of the embedding of their object. This underlines the fact that a direct way to compute triple embeddings can capture better and discriminate the roles of the same predicate/entity in different triples; (3) LoGNet performs better than Triple2Vec, especially in the variant that computes local triple embeddings using the BiGRU (i.e., LoGNet$_G$). This may be due to three main reasons. First, the Triple2Vec's triple embeddings need to be fed into a one-vs-rest logistic regressor for triple classification; this additional optimization step is not related to the original task. On the other hand, LoGNet is directly trained to perform triple classification. Second, the BiGRU approach may provide better local triple embeddings as it considers the sequential nature of triples. Third, the LoGNet mechanism may be better able to weight the importance of neighbor triples (nodes of $\mathcal{G}_L$) than the weighting mechanism used by Triple2Vec based on predicate relatedness.

5.3 Q3: Ablation Study

We conducted an ablation study introducing two more variants of LoGNet: (i) LoGNet$_L$ with only local triple embeddings and the local plausibility measure in Eq. (6); (ii) LoGNet$_G$ with only global triple representation and the global plausibility in Eq. (6). Table 3 reports the results.

Table 3. Ablation study.

KG	% Corr. triples	LoGNet	LoGNet$_L$	LoGNet$_G$
DBP	0.05%	0.6842	0.6611	0.6132
	1%	0.6912	0.6712	0.6567
	2%	0.6873	0.6684	0.6648
	3%	0.6712	0.6701	0.6467
	5%	0.6803	0.6734	0.6212
DBP1M	0.05%	0.7022	0.6911	0.6674
	1%	0.6967	0.6856	0.6689
	2%	0.6712	0.6687	0.6511
	3%	0.6790	0.6701	0.6621
	5%	0.6816	0.6745	0.6687
NELL	0.05%	0.6312	0.6256	0.6073
	1%	0.6321	0.6301	0.5998
	2%	0.6476	0.6412	0.6278
	3%	0.6311	0.6287	0.6192
	5%	0.6553	0.6493	0.6398

Results. We observe that the local plausibility score (LoGNet$_L$) seems to better capture anomalous predicates in triples. This comes as no surprise since local triple embeddings look at each triple from a finer-grained perspective, effectively analyzing the sequential nature of the triple and the dependency between subject, predicate, and object (reading the triple both forward and backward). On the other hand, LoGNet$_G$ can only rely on triple neighbor information aggregation, which cannot look into the triple. The fully-fledged LoGNet brings a clear improvement to the variants only considering either the local or global plausibility. This underlines that it is not convenient to separate local and global information.

6 Conclusions and Future Work

This paper showed how to compute triple embeddings by leveraging node and edge features derived from a KG. Our triple-centric embedding approach brings a refreshing perspective to a landscape dominated by node/edge-centric applications. It can support a variety of applications like path-based downstream applications where paths can be embedded as sequences of triple embeddings or data release scenarios where the same predicate may or not be considered sensitive depending on the subject and object like in the triples (Pat, marriedTo, Claire) and (Frank, marriedTo, Mary). Using the same predicate embedding to compute the embeddings of two triples may be counter-intuitive.

References

1. Abboud, R., Ceylan, İ.İ., Grohe, M., Lukasiewicz, T.: The surprising power of graph neural networks with random node initialization. In: Zhou, Z. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event/Montreal, Canada, 19–27 August 2021, pp. 2112–2118 (2021)
2. Belth, C., Zheng, X., Vreeken, J., Koutra, D.: What is normal, what is strange, and what is missing in a knowledge graph: unified characterization via inductive summarization. In: Proceedings of the Web Conference 2020, pp. 1115–1126 (2020)
3. Bianconi, G., Pin, P., Marsili, M.: Assessing the relevance of node features for network structure. Proc. Natl. Acad. Sci. **106**(28), 11433–11438 (2009)
4. Bordes, A., Usunier, N., García-Durán, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Burges, C.J.C., Bottou, L., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held 5–8 December 2013, Lake Tahoe, Nevada, United States, pp. 2787–2795 (2013)
5. Cai, H., Zheng, V.W., Chang, K.C.C.: A comprehensive survey of graph embedding: problems, techniques, and applications. Trans. Knowl. Data Eng. **30**(9), 1616–1637 (2018)
6. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr., E.R., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 5 (2010)
7. Chekol, M.W., Pirrò, G.: Refining node embeddings via semantic proximity. In: Pan, J.Z., et al. (eds.) ISWC 2020. LNCS, vol. 12506, pp. 74–91. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62419-4_5
8. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Empirical Methods in Natural Language Processing (EMNLP) (2014)
9. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS 2014 Deep Learning and Representation Learning Workshop (2014)
10. Darari, F., Nutt, W., Pirrò, G., Razniewski, S.: Completeness management for RDF data sources. ACM Trans. Web (TWEB) **12**(3), 1–53 (2018)
11. Dettmers, T., Minervini, P., Stenetorp, P., Riedel, S.: Convolutional 2D knowledge graph embeddings. In: Proceedings of the AAAI Conference, pp. 1811–1818 (2018)

12. Dong, X., Chawla, N.V., Swami, A.: metapath2vec: scalable representation learning for heterogeneous networks. In: Proceedings of International Conference on Information and Knowledge Management, pp. 135–144 (2017)
13. Fionda, V., Pirrò, G.: Learning triple embeddings from knowledge graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 3874–3881 (2020)
14. Grover, A., Leskovec, J.: node2vec: scalable feature learning for networks. In: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016, pp. 855–864. ACM (2016)
15. Hamilton, W.L., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA, pp. 1024–1034 (2017)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
17. Hussein, R., Yang, D., Cudré-Mauroux, P.: Are meta-paths necessary?: revisiting heterogeneous graph embeddings. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, 22–26 October 2018, pp. 437–446. ACM (2018)
18. Jia, S., Xiang, Y., Chen, X., Wang, K.: Triple trustworthiness measurement for knowledge graph. In: Liu, L., et al. (eds.) The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, 13–17 May 2019, pp. 2865–2871. ACM (2019)
19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, 24–26 April 2017, Conference Track Proceedings. OpenReview.net (2017)
20. Liang, J., Xiao, Y., Zhang, Y., Hwang, S.W., Wang, H.: Graph-based wrong ISA relation detection in a large-scarsescale lexical taxonomy. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, pp. 1178–1184 (2017)
21. Melo, A., Paulheim, H.: Detection of relation assertion errors in knowledge graphs. In: Proceedings of the Knowledge Capture Conference, pp. 1–8 (2017)
22. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of International Conference on Neural Information Processing, pp. 3111–3119 (2013)
23. Nathani, D., Chauhan, J., Sharma, C., Kaul, M.: Learning attention-based embeddings for relation prediction in knowledge graphs. In: 57th Annual Meeting of the Association for Computational Linguistics, pp. 4710–4723 (2019)
24. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2014, New York, NY, USA, 24–27 August 2014, pp. 701–710 (2014)
25. Pirrò, G.: Building relatedness explanations from knowledge graphs. Semant. Web **10**(6), 963–990 (2019)
26. Pirrò, G.: Fact-checking via path embedding and aggregation. In: Joint Proceedings of Workshops AI4LEGAL2020, NLIWOD, PROFILES 2020, QuWeDa 2020 and SEMIFORM2020 Colocated with the 19th International Semantic Web Conference (ISWC 2020), Virtual Conference, November 2020. CEUR Workshop Proceedings, vol. 2722, pp. 149–158. CEUR-WS.org (2020)

27. Ristoski, P., Paulheim, H.: RDF2Vec: RDF graph embeddings for data mining. In: Proceedings of International Semantic Web Conference, pp. 498–514 (2016)
28. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Networks **20**(1), 61–80 (2009)
29. Shiralkar, P., Flammini, A., Menczer, F., Ciampaglia, G.L.: Finding streams in knowledge graphs to support fact checking. In: 2017 IEEE International Conference on Data Mining (ICDM), pp. 859–864. IEEE (2017)
30. Sun, Z., Deng, Z.H., Nie, J.Y., Tang, J.: RotatE: knowledge graph embedding by relational rotation in complex space. In: Proceedings of International Conference on Learning Representations (2019)
31. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: Proceedings of International Conference on Machine Learning, pp. 2071–2080 (2016)
32. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: Proceedings of International Conference on Learning Representations (2017)
33. Wang, X., Lu, Y., Shi, C., Wang, R., Cui, P., Mou, S.: Dynamic heterogeneous information network embedding with meta-path based proximity. IEEE Trans. Knowl. Data Eng. **34**(3), 1117–1132 (2022)
34. West, D.B., et al.: Introduction to Graph Theory, vol. 2. Prentice Hall, Hoboken (1996)
35. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019. OpenReview.net (2019)
36. Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015, Conference Track Proceedings (2015)
37. You, J., Ying, R., Leskovec, J.: Position-aware graph neural networks. In: International Conference on Machine Learning, pp. 7134–7143. PMLR (2019)