CEUR-WS.org/Vol-2722/semiform2020-paper-1.pdf

Fact-checking via Path Embedding and Aggregation

Giuseppe Pirrò

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

Abstract. Knowledge graphs (KGs) are a useful source of background knowledge to (dis)prove facts of the form (s, p, o). The goal of this paper is to present the **F**act checking via path **E**mbedding and **A**ggregation (FEA) system. FEA starts by carefully collecting the paths between s and o that are most semantically related to the domain of p. It learns vectorized path representations, aggregates them according to different strategies, and use them to finally (dis)prove a fact. Our experiments show that our hybrid solution brings benefits in terms of performance.

1 Introduction

[1] We live in a digital era, where both false and true rumors spread at an unprecedented speed. In this open context, having a way to assess the reliability of individual facts is of utmost importance. How could one quickly verify the reliability of statements like (Dune, directed, D. Lynch)?
Related Work. Existing approaches, can roughly been categorized in three main categories. *First*, *text-based approaches* based on a variety of learning models; these can use probability and logics (e.g., [2]), deep-learning (e.g., [12]), and also include multi-modal (e.g., text and video) information (e.g., [5]). While these approaches can rely on large amounts of text and/or mutimedia sources like audio and video, there are difficulties in automatically understanding such pieces of information to (dis)prove a fact. *This makes it difficult to give precise semantics to the fact being checked and contextualize it.* On one hand, giving semantics boils down to understanding the fact itself rather than relying on statistical indicators like the popularity of a tweet about the fact. For instance, to (dis)prove the fact (Dune, director, D. Lynch), it is crucial to understand that the predicate director relates a Film and a Director and that Director is a subclass of Person. On the other hand, contextualizing facts and gaining insights from (chains of) related facts can represent a valuable source of knowledge [20]. As an example, the fact (Jaguar, owner, Tata Motors) provides more insights when understanding that it is about the car brand instead of the animal; the additional fact (Tata Motors, type, Company) can help in shedding light on this aspect.

[1] Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Second, *approaches that leverage structured knowledge* (e.g., knowledge graphs) instead of unstructured text (e.g., [16, 7, 14, 18]). In this case, structured background knowledge allows for more precise forms of reasoning for fact-checking. For instance, it has been shown that the paths between the subject and object of a targeted fact, that include other entities and predicates, form a valuable body of semantic evidence (see e.g., [15, 7]). These approaches offer advantages in terms of semantic interpretation and contextualization of a statement. For instance, the statement (Dune, director, D. Lynch) can be given both a semantic characterization and put into context by retrieving information from KGs like DBpedia. For instance, we understand that the domain of the fact is that of movies and that there are frequently occurring semantic relations between D. Lynch and actors (e.g., K. Mclaughlin) that also acted in Dune. Moreover, the usage of paths or entire portions of a KG of interest for the target statement can provide (visual) evidence about why the fact is true or false. *Nevertheless, KG-based approaches lack mechanisms to automatically differentiate the importance of the collected paths. Third*, a more recent strand of research has considered the usage of *entity and predicate embeddings* for fact-checking (e.g., [17, 3]). The idea of these approaches is to treat fact-checking as a link prediction problem. *While these approaches have the advantage of working with vectorized representations of entities and predicates to automatically identify and extract salient features, they are sub-optimal as they do not directly tackle the problem of vectorizing entire facts, paths, and their aggregation.*
Contributions. The goal of this paper is to present the **F**act checking via path **E**mbedding and **A**ggregation (**FEA**) system. FEA carefully collects paths from a KG between the subject and object of a fact to be checked that are most semantically relevant to it. However, instead of directly working with this subset of all paths, it learns vectorized path representations, aggregates them according to different strategies, and use them to finally (dis)prove a fact. To the best of our knowledge, this is the first work combining triple and path embedding and aggregation for fact checking.

2 Preliminaries

A Knowledge Graph (KG) contains facts (aka statements) that can be divided into an ABox and a TBox. We see the ABox as a node and edge-labeled directed multi-graph $G{=}(V, E, T)$ where V is a set of uniquely identified vertices representing entities (e.g., D. Lynch), E a set of predicates or properties (e.g., director) and T a set of facts of the form (s, p, o), where s, o $\in V$ and p $\in E$. The TBox is another multi-graph defined as $T = (C_t, P_t, L_t, T_t)$, where C_t is the set of all class names, P_t is the set of all property names, L_t is a set of properties defined in some ontological language, and T_f is a set of triples of the form (u, p, v) where $u, v \in C_t \cup P_t$ and $p \in L_t$. In this paper, we consider L_t to be the subset of the RDFS ontological language defined as follows: L_t={rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, rdfs:range}. We use the notation $domain(p)$ (resp., $range(p)$) to indicate the domains (resp., ranges) of a property. After applying the RDFS inference on the TBox T_f, we construct the corresponding TBox graph defined as $G_S=(V_s, E_s, T_s)$, where each $v_i \in V_s$ is a class name belonging to C_t, $p_i \in P_t \cup \{\mathsf{rdfs{:}subClassOf}\}$, and $(v_s, p_i, v_t) \in T_s$ is a triple such that $domain(\mathsf{p}_i)=v_s$ and $range(\mathsf{p}_i)=v_t$.

3 The FEA Framework

The problem we solve can be formulated as follows: given a fact $(\mathsf{s}, \mathsf{p}, \mathsf{o})$, and a set of paths $\mathcal{P}(\mathsf{s}, \mathsf{p}, \mathsf{o}) = \{\pi_1, \pi_2, \dots, \pi_k\}$ connecting s and o and *related to the domain* expressed by p, the goal is to estimate the truthfulness of the fact by:

$$\Phi_{(\mathsf{s},\mathsf{p},\mathsf{o})} = m_{\Theta}((\mathsf{s}, \mathsf{p}, \mathsf{o}), \mathcal{P}((\mathsf{s}, \mathsf{p}, \mathsf{o}))) \quad (1)$$

where m is the model having parameters Θ and $\Phi \in [0, 1]$ is the truthfulness score. The FEA framework consists of four main modules: *path extractor*, *path embedder*, *path aggregator*, and *fact checker*; Fig. 1 provides an overview of the framework. The end-to-end learning objective of FEA is guided by the input fact to be checked (s, p, o) and a set of (domain-specific) paths extracted from a knowledge graph. The output of the model $\Phi_{(\mathsf{s},\mathsf{p},\mathsf{o})}$ represents the truthfulness of the fact. Note that the set of input paths can provide evidence useful to understand why a given fact is true or false, for instance by displaying paths.

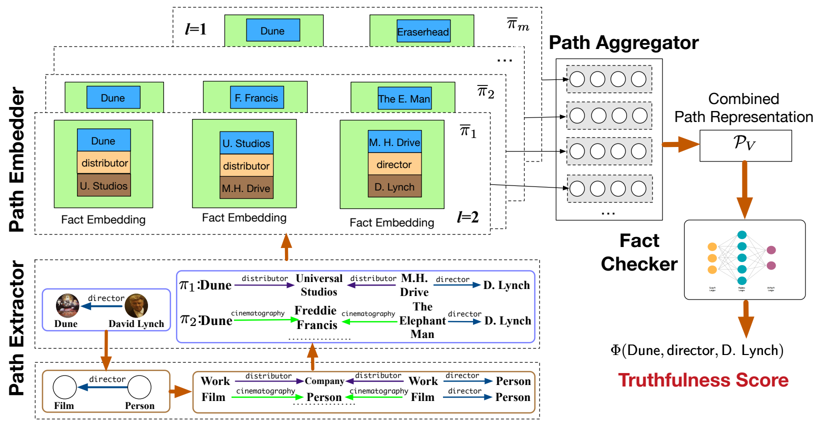
Path Embedder
Path Extractor
l=1
l=2
Dune
Eraserhead
π_m
F. Francis
The E. Man
π_2
U. Studios
M. H. Drive
distributor
director
D. Lynch
π_1
Fact Embedding
Path Aggregator
Combined Path Representation
$\mathcal{P}_V$
Fact Checker
π_1:Dune distributor Universal Studios distributor M.H. Drive director D. Lynch
π_2:Dune cinematography Freddie Francis cinematography The Elephant Man director D. Lynch
Dune director David Lynch
Film director Person
Work distributor Company distributor Work director Person
Film cinematography Person cinematography Film director Person
Φ(Dune, director, D. Lynch)
Truthfulness Score

Fig. 1. Overview of FEA. Paths are grouped and processed for each length l.

3.1 Path Extractor

This module is responsible for the exploration of the KG to gather information in the form of paths, which will be used by the other modules.

Schema-level patterns. The Path Extractor leverages the TBox graph to find schema-level patterns for an input predicate p. It assembles paths up to a length

l between the domain(s) and range(s) of p treating the input graph as undirected. To reduce the search space, this module only extracts the patterns most relevant to p, where relevance is defined in terms of the extent to which the path is semantically related to p. As an example, for the predicate director, paths including predicates like director, starring producer are intuitively more relevant than paths including birthDate or college. To quantify the relevance between a predicate and a schema-level pattern, the Path Extractor relies on a predicate relatedness measure. Given a pair of predicates (p_i, p_j) their relatedness is:

$$Rel(p_i, p_j) = Cosine(Emb(p_i), Emb(p_j)) \tag{2}$$

where $Emb(\cdot)$ is an embedding function (e.g., RotatE [19]) and $Cosine$ is the cosine operation between the vector embeddings of p_i and p_j. Finally, the relatedness between a path and a predicate p is computed as the average relatedness between p and all predicates in the path. The algorithm to extract patterns, not reported for sake of space, proceeds with a BFS traversal of the TBox graph conditioned on the top-k predicates.

Data-level paths. The Path Extractor has available a set of schema-level patterns $\mathcal{P}_{\mathsf{p}}$, for each predicate p, found in the previous step. Hence, given an input fact (s, p, o), the goal is to find data-level paths from the ABox for each schema-level pattern $\pi_i \in \mathcal{P}_{\mathsf{p}}$. We adopt an algorithm based on a variant of Depth-First-Search (DFS), which starts from s and at each traversal step of the graph ensures the compliance with π_i in terms of predicate traversed and entity types toward reaching the entity o. Consider the fact (Dune, director, D. Lynch), the schema-level-path $\pi = \mathsf{Work} \xrightarrow{\text{starring}} \mathsf{Actor} \xleftarrow{\text{starring}} \mathsf{Work} \xrightarrow{\text{director}} \mathsf{Person}$ and the DBpedia KG. The algorithm starts from the node Dune and traverses the edge starring (as per π) reaching the nodes J. Nance, K. Mclaughlin, and E. McGill. From each of these nodes, it traverses edges labeled as starring in reverse direction (again as per π) and reaches the nodes Eraserhead, Twin Peaks and Twin Peaks Fire Walks with Me. Finally, according to the last step of π, the algorithm traverses edges labeled as director thus closing the paths between the subject Dune and the object D. Lynch of the input fact. When considering the pattern $\pi = \mathsf{Film} \xrightarrow{\text{cinematography}} \mathsf{Person} \xleftarrow{\text{director}} \mathsf{Film} \xrightarrow{\text{editing}} \mathsf{Person}$, it is not possible to find any path between Dune and D. Lynch complying with π in the ABox. If no path can be found, FEA performs an unconstrained DFS.

3.2 Path Embedder

To be processed by the learning model at the core of FEA, paths found by the Path Extractor are given a numerical representation. This is done by vectorizing each fact (triple) in a path, which can be done in different ways. One way is to consider techniques like TransE [3] or DistMult [21] to first learn entity and predicate embeddings via a generic function Emb($\cdot$), which given an entity or a predicate, returns its corresponding vector embedding. Hence, to compute the embedding of a fact t=(s, p, o), one can perform some operation *op* (e.g., concatenation) on its constituents vectors, that is, Emb(t)=*op*(Emb(s), Emb(p), Emb(o)). Note that we do not consider one-hot encodings since these techniques do not take into account the structure of the KG. Another way to learn fact embeddings is to rely on approaches like triple2vec [8], which instead of learning embeddings for entities and predicates separately directly learns fact embeddings. For the time being, given a fact t=$(\mathsf{s}, \mathsf{p}, \mathsf{o})$, we define its embedding as t_E=$\mathsf{EmbF}(t)$. Building upon the embedding of facts, a path π=$\{t^1, t^2, \ldots t^l\}$ of length l including l facts is encoded as a sequence π_E=$[t_E^1, t_E^2, \ldots, t_E^l]$.

3.3 Path Aggregator and Fact Checker

Paths converted into their vector form by the Path Embedder are then passed to the Path Aggregator. The Aggregator implements a variety of aggregation strategies. We can see the aggregator as another learning module, which takes the paths from the Path Embedder and provides an overall vector representation for them. We considered the following aggregation strategies:

1. **Average Pool.** It combines the different representations of paths by concatenating the vector representations of the facts in a path. Then on the set of paths obtained, the aggregator performs a 1D average pooling operation. The final combined path representation is a single vector obtained by averaging the paths between s and o. This can be summarized as follows:

$$\mathcal{P}_V^l = AvgPool([\oplus(\pi_i^l), \forall \pi_i^l \in \mathcal{P}^l]) \tag{3}$$

   where $AvgPool$ is the one-dimensional average pooling operation, and $\oplus(\cdot)$ is the vector concatenation operation. This representation relies on the embeddings of the facts in each path.

2. **Max Pool.** What changes wrt the AvgPool is the final vector of the path; instead of being the average, it is now computed by using a dense neural network layer. The resulting activations are then passed through a max-pooling operation which helps to derive a single vector representation for the paths of length l. The whole operation can be summarized as follows:

$$\mathcal{P}_V^l = MaxPool([\sigma(W_l \cdot \oplus(\pi_i^l) + b_l), \forall \pi_i^l \in \mathcal{P}^l]) \tag{4}$$

   where $MaxPool$ is the one-dimension max pool operation (which selects bitwise the maximum value from multiple vectors to derive a single final vector.), W_l are the weights to be learned, b_l the bias, and σ the activation function.

3. **LSTM Max Pool.** The idea is to treat a (vectorized) path as a sequence an employ an LSTM network to cater for sequential dependencies between facts in a path. With this reasoning, each fact in a path represents a point of a sequence. At each step $l-1$, the LSTM layer outputs a hidden state vector h_{l-1}, consuming subsequence of embedded facts $[f_1, ..., f_{l-1}]$. In other words, x_{l-1}=f_{l-1}. The input x_{l-1} and the hidden state h_{l-1} are used to learn the hidden state of the next path step l. After processing all of them via the LSTM, the aggregator employs another LSTM followed by a max pool operation to produce the combined representation $\mathcal{P}_V$.

As the Path Extractor groups paths according to their different lengths, the Path Aggregator processes each length-specific set of paths separately. Finally, the path representations for each length are concatenated together to give the final length-specific path representation $\mathcal{P}_V$ (see Fig. 1). The last step of the FEA framework consists in providing the final truthfulness score about the input fact. This is done by the Fact Checker, which takes as input the output of the Path Aggregator (i.e., the vector representation $\mathcal{P}_V$) and feeds it into a classifier. We treat the fact-checking problem as a binary classification problem, where a true fact and a false fact are assigned 1 and 0 as target values, respectively. The final goal is to optimize the negative log-likelihood objective function, which defined as follows:

$$\mathcal{L} = -\sum_{f^+ \in \mathcal{F}^+} log\ \hat{y}_{f+} + \sum_{f^- \in \mathcal{F}^-} log(1 - \ \hat{y}_{f-}) \tag{5}$$

where $\mathcal{F}^+$=$\{f^+ \mid y_{f^+}$=$1\}$, $\mathcal{F}^-$=$\{f^+ \mid y_{f^-}$=$0\}$ are the true (resp., false) facts.

4 Evaluation

We tested the ability of our approach to check facts considering both existing and not existing facts in a given KG. We use the Area Under the Receiver Operating Characteristic curve (AUC) as the primary quality indicator because it is independent from thresholds and has been used previously (e.g., [18, 7]). All experiments have been carried out on a machine with a 4 core 2.7 GHz CPU and 16 GB RAM. We considered DBpedia as underlying KG.
Embedding and relatedness computation. To compute fact embeddings we used **triple2vec** [8], which directly computes fact embeddings by leveraging the notion of line graph of a KG. We tested other indirect approaches based on the embedding of the triple elements (see Section 3.2) but obtained less competitive results. Predicate embeddings were used to obtain a predicate relatedness matrix, where the relatedness of each pair of predicates is computed as per equation (2) for all datasets but DBpedia. In this case, we obtained the predicate relatedness matrix from KStream[2]. This was necessary since neither DistMult nor ComplEx could run on this dataset on our machine.

4.1 Comparison with related work

We considered the following competitors: *(i)* **CHEEP** [7], an approach, which leverages paths to come up with a truthfulness score for an input fact; *(ii)* **PredPath** [15], which exploits frequent anchored predicate paths between pair of entities in the KG; *(iii)* Path Ranking Algorithm (**PRA**) [9], which extracts (positive and negative) training set of triples via a two-sided unconstrained random walk starting from the fact endpoints to retrieve paths between them; *(iv)* **KStream** [18], which reduces the fact-checking problem to the problem of maximizing the flow between the subject and the object of the fact; *(v)* **Klinker**

[2] https://github.com/shiralkarprashant/knowledgestream

[4], which relies on a single short, specific path to differentiate between a true and a false fact; *(vi)* **LEAP** [1], which tackles the problem of link prediction on unlabeled graphs. We included LEAP as we took inspiration from it for the path aggregation strategies, although FEA tackles the more challenging problem of fact-checking. For LEAP, paths were generated ignoring the edge labels and we report the best results obtained with its aggregation strategies; *(vii)* we could not run experiments with DistMult and ComplEx because of memory issues. Nevertheless, we report the results for **TransE** [3] obtained by Shiralkar et al. [18]. We did not consider approaches based on logical rules learned from the KG (e.g., [13]) since it is not completely clear how to obtain high-quality rules.

Approach	**birthPlace** (273/1092)	**deathPlace** (126/504)	**almaMater** (1546/6184)	**nationality** (50/200)	**profession** (110/440)
FEA-LSTMAggr	.93	.91	.81	.89	.99
FEA-MaxAggr	.90	.87	.80	.86	.97
FEA-AvgAggr	.91	.86	.81	.87	.99
CHEEP	.91	.87	.77	.85	.98
KStream	.82	.84	.75	.93	.93
KLinker	.91	.87	.78	.86	.93
PredPath	.86	.76	.83	.95	.92
PRA	.74	.75	.63	.83	.50
LEAP	.81	.74	.80	.91	.88
TransE	.54	.56	.66	.77	.82

Approach	**author** (93/558)	**team** (41/164)	**director** (78/4680)	**keyPerson** (201/1208)	**spouse** (16/256)
FEA-LSTMAggr	.93	.92	.99	.84	.98
FEA-MaxAggr	.90	.91	.97	.79	.94
FEA-AvgAggr	.92	.93	.99	.81	.98
CHEEP	.91	.91	.99	.82	.96
KStream	.92	.99	.83	.80	.86
KLinker	.96	.92	.88	.83	.91
PredPath	.99	.92	.84	.88	.87
PRA	.96	.91	.99	.87	.88
LEAP	.99	.89	.81	.82	.85
TransE	.80	.56	.82	.83	.79

Fig. 2. Performance (average AUC) on both real-world (up) and synthetic (down) datasets (average of 4 runs); # true facts over all facts appears below the predicate.

Benchmarks. We compared the various systems on two benchmarks defined on DBpedia. The first defined in Shiralkar et al. [18] and available online[3]. It includes 5 real-world datasets derived from Google Relation Extraction Corpora and WSDM Cup Triple Scoring challenge and 5 synthetic datasets mix a-priori known true and false facts. The number of true/false facts for each benchmark is reported below the predicate name in Table 2. The second benchmark[4] re-

[3] https://github.com/shiralkarprashant/knowledgestream/
[4] https://github.com/huynhvp/BUCKLE-Fact_checking/

leased by Huynh and Papotti [11] takes into account popularity, transparency, homogeneity, and functionality properties of the facts to cover a broader variety of scenarios than previous benchmarks.

Evaluation results. We observe that on the first benchmark (Fig. 2), **FEA** performs quite well for all predicates considered. In particular, it brings some improvement wrt **CHEEP**, the second-best performing system, when considering the *LSTMMaxPool* aggregator. We note that approaches like **PredPath**, which only consider one path perform worse; perhaps a single path is not able to capture all needed semantic evidence. As expected, the worst-performing system is **LEAP**, which, however, has not been designed to work on labeled graphs as it aims to solve the link prediction problem in unlabeled graphs. We observe that **TransE**, which also tackles the link prediction problem performs better than **LEAP**; although worse than the other systems. This could be because it does not consider paths. Results are more interesting in the second benchmark (Table 1), which has carefully been designed to test the behavior of fact-checking systems on (non)popular (NP) and random entities (R). On this benchmark, we

§small

Approach	**Predicate**	**P**	**NP**	**R**
FEA-LSTMAggr	**nearestCity**	.87	.67	.78
	foundedBy	.82	.67	.86
	manufacturer	.91	.88	.94
	employer	.70	.52	.71
FEA-MaxAggr	**nearestCity**	.79	.61	.72
	foundedBy	.77	.63	.79
	manufacturer	.88	.79	.86
	employer	.66	.47	.62
FEA-AvgAggr	**nearestCity**	.85	.64	.75
	foundedBy	.79	.63	.80
	manufacturer	.89	.88	.91
	employer	.68	.50	.67
CHEEP	**nearestCity**	.86	.61	.72
	foundedBy	.81	.62	.79
	manufacturer	.78	.86	.81
	employer	.67	.43	.64
PredPath	**nearestCity**	.84	.58	.69
	foundedBy	.80	.63	.81
	manufacturer	.55	.51	.53
	employer	.58	.38	.50
KLinker	**nearestCity**	.87	.66	.76
	foundedBy	.82	.67	.80
	manufacturer	.90	.85	.92
	employer	.69	.43	.66
LEAP	**nearestCity**	.41	.40	.41
	foundedBy	.69	.58	.71
	manufacturer	.68	.57	.64
	employer	.58	.42	.43
TransE	**nearestCity**	.49	.40	.43
	foundedBy	.75	.60	.75
	manufacturer	.72	.47	.70
	employer	.62	.46	.48

Table 1. AUC on the benchmarks in [11] for popular (**P**), non popular (**NP**), and random (**R**) entity pairs. Train/test pairs have been provided by the authors of [11].

ran experiments for **FEA**, **LEAP**, and **CHEEP** while for **TransE**, **KLinker**, and **PredPath** we report results from [11]. Here we observe that **FEA** performs particularly well on non-popular entities with both *LSTMMaxPool* and *Avg* aggregators. This may be explained by the fact that even when the number of paths is smaller than between popular entities, the Path Aggregator can correctly capture the necessary evidence, which passed to the other modules of the **FEA** framework (after embedding) captures the truthfulness of facts eventually. **FEA** leverages deep-learning techniques for the embedding of paths providing a strategy that can capture dependencies between the facts in a path and aggregate them. Moreover, we remark the importance of considering the semantics of paths for fact-checking but, most importantly, the need to correctly relate the semantics of such paths with the fact to be checked in order to only consider the most relevant ones. Indeed, even if **LEAP** uses node embedding and path aggregation strategies, it is the worst performing system. We noted that the system fails to especially recognize false facts since even if the existence of a link is correctly predicted, this is not enough as for fact-checking it is necessary to establish the existence of a specific link.

5 Conclusions and Future Work

We describe a fact-checking approach that combines path-based approaches and embedding based approaches. Our experiments showed that first embedding whole facts in a path and then aggregating them is a viable solution. Investigating other aggregation strategies, the usage of adversarial learning techniques [10], and temporal information [6] is in our research agenda.

References

1. Agrawal, R., de Alfaro, L.: Learning edge properties in graphs from path aggregations. In: WWW. pp. 15–25. ACM (2019)
2. Ahmadi, N., Lee, J., Papotti, P., Saeed, M.: Explainable fact checking with probabilistic answer set programming. arXiv preprint arXiv:1906.09198 (2019)
3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NIPS. pp. 2787–2795 (2013)
4. Ciampaglia, G.L., Shiralkar, P., Rocha, L.M., Bollen, J., Menczer, F., Flammini, A.: Computational Fact Checking from Knowledge Networks. PloS one **10**(6) (2015)
5. ar Dhruv, K., Singh, G.J., Manish, G., Vasudeva, V.: Mvae: Multimodal variational autoencoder for fake news detection. In: WWW (2019)
6. Ercan, G., Elbassuoni, S., Hose, K.: Retrieving textual evidence for knowledge graph facts. In: ESWC. pp. 52–67. Springer (2019)
7. Fionda, V., Pirrò, G.: Fact checking via evidence patterns. In: IJCAI. pp. 3755–3761 (2018)
8. Fionda, V., Pirrò, G.: Learning triple embeddings from knowledge graphs. In: AAAI, pp. 3874–3881 (2020)
9. Gardner, M., Talukdar, P., Krishnamurthy, J., Mitchell, T.: Incorporating vector space similarity in random walk inference over knowledge bases. In: ENLMP. pp. 397–406 (2014)

10. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NIPS. pp. 2672–2680 (2014)
11. Huynh, V.P., Papotti, P.: A benchmark for fact checking algorithms built on knowledge bases. In: CIKM (2019)
12. Karadzhov, G., Nakov, P., Màrquez, L., Barrón-Cedeño, A., Koychev, I.: Fully automated fact checking using external sources. In: RANLP. pp. 344–353 (2017)
13. Leblay, J.: A declarative approach to data-driven fact checking. In: AAAI. pp. 147–153 (2017)
14. Pan, J.Z., Pavlova, S., Li, C., Li, N., Li, Y., Liu, J.: Content based fake news detection using knowledge graphs. In: ISWC. pp. 669–683. Springer (2018)
15. Shi, B., Weninger, T.: Discriminative Predicate Path Mining for Fact Checking in Knowledge Graphs. Knowledge-Based Systems **104**, 123–133 (2016)
16. Shi, B., Weninger, T.: Fact checking in heterogeneous information networks. In: WWW companion. pp. 101–102 (2016)
17. Shi, B., Weninger, T.: Proje: Embedding projection for knowledge graph completion. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
18. Shiralkar, P., Flammini, A., Menczer, F., Ciampaglia, G.L.: Finding Streams in Knowledge Graphs to Support Fact Checking. In: ICDM. pp. 859–864 (2017)
19. Sun, Z., Deng, Z.H., Nie, J.Y., Tang, J.: RotatE: Knowledge graph embedding by relational rotation in complex space. In: ICLR (2019)
20. Voskarides, N., Meij, E., Reinanda, R., Khaitan, A., Osborne, M., Stefanoni, G., Kambadur, P., de Rijke, M.: Weakly-supervised contextualization of knowledge graph facts. In: SIGR. pp. 765–774. ACM (2018)
21. Yang, B., Yih, W.t., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: ICLR (2015)