Neurosymbolic Reasoning: Building Neural Networks using Datalog$^{\pm}$

Mattia Scaccia[1], Ilaria Stocchi[1] and Luigi Bellomarini[2]

[1]Department of Engineering, Roma Tre University, Via Vito Volterra 62, 00146 Roma, Italy
[2]IT department, Banca d'Italia, Centro Donato Menichella, Largo Guido Carli 1, 00044 Frascati, Italy

Abstract

Neurosymbolic reasoning has become an active research area for both academic and industrial context to handle today's complex knowledge-based problems, such as reasoning on Knowledge Graphs (KGs). Reasoning on KGs can effectively leverage the symbolic reasoning techniques, such as Datalog$^{\pm}$-based techniques, but the discrete nature of these representations makes them insufficient to capture all the intrinsic relationships among data. On the other hand, neural networks have been widely used to enrich KGs, thanks to their subsymbolic capabilities, but the need for better explainability, interpretability and trust of machine learning systems demands the addition of a symbolic representation. We present the Chase Graph Neural Network (CGNN), a neural network that mirrors the symbolic reasoning process and is able to compute a vector representation of facts produced by the reasoning process. We show, in the context of a real-world economic domain, that the CGNN enriches KGs with new knowledge beyond what it is achievable using only symbolic reasoning.

Keywords

Neurosymbolic Reasoning, Datalog, Vadalog, Knowledge Graph, Neural Network, Machine Learning, Artificial Intelligence

1. Introduction

Reasoning on KGs is a suitable setting for the query answering problem [1]. In real-world applications, however, query answering is particularly challenging: the language to express the reasoning rules must be chosen in such a way that it is able to represent complex domains (high expressive power) and, at the same time, guarantees tractability and decidability of the query answering task, in order to support high performance and scalability with large volumes of data.

We adopt the Vadalog engine [2] as the basis of our work, a highly optimized Knowledge Graph Management System, which performs reasoning over KGs and is able to meet the above query answering requirements by using a fragment of Datalog$^{\pm}$ [3], a family of logic query languages. Our goal is to enrich the domain knowledge that can be discovered by Vadalog during the reasoning process. We intend to demonstrate that introducing the subsymbolic reasoning paradigm, typical of machine learning, as a support of the Vadalog logic reasoning process, we can derive additional information w.r.t. what Vadalog is able to obtain, given a specific query answering problem. In fact the knowledge obtained by Vadalog is solid and explainable but, at the same time, also lacking of what can be derived from observing the raw data itself. We aim to develop a *Symbolic-Driven Neural Reasoning* [4] model, in which the rules that state the reasoning task are mapped into a subsymbolic representation in order to benefit both from logic and from the ability of machine learning to derive information from patterns hidden in data. According to the classification provided by Vardi et al. [5] this neurosymbolic computation model would fall in the type 5, in which logic acts as a regularizer of the neural network's loss function. We present the *Chase Graph Neural Network* (CGNN), a neural network able to compute a vector representations (embeddings) of every fact produced by the reasoning process. The CGNN has an innovative structure based on the chase graph, the dependency graph representing the Vadalog reasoning process produced by a chase based procedure [6]. Each fact (symbol) represents a neuron and the logical dependencies between facts produced by specific rules represent the neural connections. The CGNN computes an embedding for each fact generated by Vadalog during the reasoning process and we exploit these embeddings to perform link prediction tasks.

Overview. In Section 2 we describe the background about Vadalog and its reasoning process, while we present the main contributions of this paper in the remaining sections as follows:

- The **structure of the CGNN**. We describe the internal structure of a single neuron and show how to derive the structure of the CGNN from Vadalog chase graph.
- The **handling of labelled nulls**. We describe a novel technique to compute embeddings of facts that contain *labelled nulls*, fresh symbols standing for existentially quantified objects. This technique is needed to compute the loss function during training whenever a fact

Published in the Workshop Proceedings of the EDBT/ICDT 2022 Joint Conference (March 29-April 1, 2022), Edinburgh, UK
mat.scaccia@stud.uniroma3.it (M. Scaccia);
ila.stocchi@stud.uniroma3.it (I. Stocchi);
luigi.bellomarini@bancaditalia.it (L. Bellomarini)

with labelled nulls is produced as output.

- The **use of isomorphism to handle unseen data**. We describe a novel use of the state-of-the-art VF2 algorithm [7] to address the problem of handling new unseen data with a fully instantiated trained neural network such as the CGNN. In addition we describe improvements of the algorithm, specific to our use case, concerning performance in the average case.
- **Experimental evaluation**. There is a variety of applications for reasoning on KGs in the economic domain [8, 9]. We provide a first evaluation of the CGNN on a link prediction task for KGs in a financial domain and we present it in the form of an ablation study.

2. Vadalog Reasoning

The Vadalog engine is built around Warded Datalog$^{\pm}$ language, a fragment of Datalog$^{\pm}$ family of languages. Datalog$^{\pm}$ languages consist of existential rules, or tuple-generating dependencies (TGDs). A Datalog$^{\pm}$ rule is a first-order sentence of the form $\forall x \forall y(\varphi(x,y) \rightarrow \exists z \psi(x,z))$, where φ (the body) and ψ (the head) are conjunctions of atoms with constants and variables [2]. For brevity $\wedge$ is replaced with comma, to denote conjunction of atoms, while $\rightarrow$ is replaced with **:-** in the form of $\psi(x,z)$ **:-** $\varphi(x,y)$. In addition universal and existential quantifiers are omitted since they can be deduced: the variables in the body are all universally quantified, while only the variables that appear in the head, but not in the body, are existentially quantified. By some abuse of notations, we often use the terms atom, tuple and fact interchangeably.

Pattern-isomorphic facts. In Vadalog there is the concept of pattern-isomorphic facts. Two facts are pattern-isomorphic if they have the same predicate name, there exists a bijection between the constant values and there exists a bijection between the labelled nulls [2]. In our representation of patterns we use progressive positive integers for different constants and progressive negative integers for different labelled nulls.

Chase procedure. Vadalog uses the chase procedure to perform logic reasoning tasks. In the context of database theory, the chase procedure is considered among the fundamental algorithmic tools enabling a variety of applications [10]. The chase procedure takes as input a database D and a set T of constraints which are TGDs and, if it terminates, its result is a finite instance D_T that is a universal model of D and T, i.e., a model that can be homomorphically embedded into every other model of D and T [10]. According to this definition the chase adds new tuples to the database D, that could potentially have labelled nulls, as dictated by the rules of T, and it keeps adding tuples until all the rules of T are satisfied.

Chase graph. The chase procedure produces as output

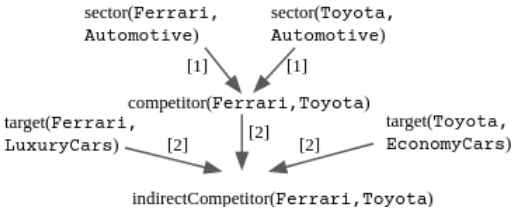

Figure 1: *Chase Graph related to the reasoning task shown in Example 1. The nodes of the chase graph represent facts while edges represent the application of rules.*

the chase graph. The chase graph for a database D and a set of rules Σ is the directed graph(D, Σ) having as nodes the facts obtained from chase(D, Σ) and having an edge from a node `a` to `b` if `b` is obtained from `a` and possibly from other facts by the application of one chase step, i.e., of one rule of Σ [2]. Therefore the chase graph is a structure that represents the entire reasoning process, as illustrated in Example 1 and Figure 1.

```
D = {sector(Ferrari, Automotive),
sector(Toyota,Automotive), target(Ferrari,
LuxuryCars), target(Toyota, EconomyCars)}
Σ = {
1. competitor(x,y):-sector(x,z),sector(y,z),
   x<>y
2. directCompetitor(x,y):-competitor(x,y),
   sourceIncome(x,z),sourceIncome(y,z)
3. indirectCompetitor(x,y):-competitor(x,y),
   target(x,z),target(y,w), z<>w}
```

Example 1: *This reasoning task aims at determining if two companies x and y are competitors (if they operate in the same sector z, rule 1) and if x and y are direct competitors (if they have the same source of income z, rule 2) or indirect competitors (if they have different market segments z and w, rule 3).*

3. Chase Graph Neural Network

In this section we describe the structure of our neurosymbolic computation model. We present the internal function of a single neuron and an overview of the overall structure of the CGNN.

3.1. Neuron Structure

Each neuron of the CGNN produces the embedding for a specific fact f generated during the chase procedure. Each neuron is made up of different computation layers as summarized in Fig. 2.

Input layer. Unlike in Feed-Forward Neural Networks [11], the CGNN does not have a specific input layer that transfers the input to the first hidden layer, instead it has at most as many input layers as the number of facts in the input database instance.

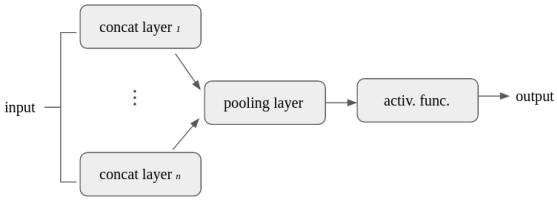

Figure 2: *Generic structure of a neuron of the CGNN.*

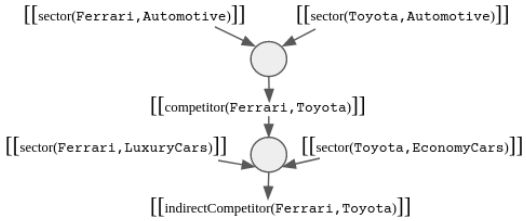

Figure 3: *The structure of the CGNN built upon chase graph in Figure 1. The ⟦fact⟧ notation is used to represent the embedding of a specific fact.*

Concat layer. The concat layer is the first layer of the neuron and it receives the input embeddings. The input consists in the embeddings of all facts that activate a specific rule r to generate a fact f. The input embeddings are concatenated columnwise, the result is a matrix that is, then, multiplied by $W_{r,f}$, the weight matrix related to rule r and fact f, as described as follows.

$$W_{\mathrm{r,f}}[[\![g_1]\!]; ...; [\![g_\mathrm{N}]\!]] \tag{1}$$

There are as many concat layers, inside a neuron, as (r_i, f) pairs, where r_i is the i-th rule that can generate f. Each r_i represents a different way to generate f. For this reason each concat layer carries a different weight matrix $W_{r_i,f}$ and therefore the CGNN can learn to distinguish these different contributions.

Pooling layer. The pooling layer receives the result matrix of all the concat layers of its neuron and it performs the following operation on each result:

$$\bigoplus_{g_1,\dots,g_\mathrm{N}}^{\beta_\mathrm{r}} W_{\mathrm{r,f}}[[\![g_1]\!]; ...; [\![g_\mathrm{n}]\!]] \tag{2}$$

where the pooling operator is defined as:

$$\bigoplus_{m}^{\beta} x_\mathrm{m} \doteq v^{-1}\left(\sum_m v(x_\mathrm{m})\right). \tag{3}$$

For each row of the input matrix the pooling sums up the result of the function v applied to each component x_m of each column. The function v and its inverse v^{-1} are defined as:

$$v(x) = sign(x)|x|^\beta \tag{4}$$

$$v^{-1}(x) = sign(x)|x|^{1/\beta} \tag{5}$$

Each element is raised to a β power, where β is a tunable parameter, and then it is performed the β-root of the result. In the end all the results produced by the pooling operator applied to all the concat layers of the neuron are summed together elementwise.

Activation Function. The resulting vector is passed to the Rectified Linear Unit (ReLU) activation function. We choose ReLU since it is easy to compute with respect to other non linear activation functions and we work with non-negative embeddings.

3.2. CGNN Structure

The CGNN is built upon the chase graph topology. For the purpose of this work, we only consider the subset of directed acyclic chase graphs that can be generated during the chase procedure. For this reason data flow forward from input to output, in the CGNN, without any feedback loop. The structure of the CGNN is made up of neurons, each one represents a specific fact generated by Vadalog. The procedure for creating each neuron n that represents a fact f follows these steps: (i) we create as many concat layers as the number of rules that contribute to generate f in the reasoning process; (ii) we add a pooling layer and establish the connections between this layer and each concat layer created with the previous step; (iii) we use the output embedding of f computed by neuron n as input for all the neurons that takes f to compute other embeddings.

In this way neurons are linked together in the same way nodes of the chase graph are, that is according to the logical dependencies between facts originated from the application of the rules. Figure 3 reports, as an example, the structure of the CGNN built upon the chase graph shown in Figure 1.

4. Embeddings for Facts with Labelled Nulls

In the context of this work we use data describing input facts in the form of text documents, since it is the most simple and adaptive form to a lot of domains. We create an embedding for each of these documents and use them as input for the CGNN. There are many state-of-the-art techniques for word or document embedding [12, 13, 14, 15], but a challenging and unanswered task, to our knowledge, is to compute an embedding for a fact with labelled nulls. In fact, whenever a fact f with labelled nulls is generated during the reasoning process, we need a technique to produce the expected embedding for f in order to compute the loss function during training.

Labelled nulls. A labelled null represents an identifier

for an unknown value and is produced as a result of existential quantification, as illustrated in Example 2.

```
D = {employee(Jack), contract(Jack),
employee(Al), contract(Al), employee(John)}
Σ = {
1. manager(y,x):-employee(x)
2. hired(y,x):-manager(y,x), contract(x)
3. contractSigned(x):-hired(y,x), manager(y,z)}
```

Example 2: *This set of rules states that every employee x has a manager y (rule 1). If a manager y sees that there is a pending contract for his employee x, then he hires x (rule 2). Once an employee x has been hired by a manager y, then the respective contract for x is signed, but if someone has been hired by an employee who is not a manager, then the contract will not be signed (rule 3).*

Using the cosine similarity between two embeddings, we claim that an embedding with one or more labelled nulls should be similar to all the embeddings which represent that same predicate name. Moreover this similarity should grow with the number of variables, at a given position, that are equal to each other and are not labelled nulls. For example the embedding for the fact hired(z1,Al) should be most similar to the embeddings of hired(Kevin,Al) or hired(Helen,Al) since they have the same predicate name and the same instantiated variables in the same position (in this case Al at position 2). It should also be less similar to the embedding of hired(Bill, John) since only the predicate name is the same. Lastly it should be completely different from the embeddings of employee(Al) or manager(Tim,Jack) since the predicate name is not the same and they represent entirely different entities.

Solution. Our novel technique to create embeddings for facts with labelled nulls is to take all the embeddings of the facts in the input database with the same predicate name of the one needed to embed and with the same instantiated variables at a given position except for the labelled nulls. All this embeddings delimit a portion in the embedding space and the centroid of this space represents the embedding of the fact with the labelled nulls. In Figure 4 the respective embedding spaces and embeddings for the two facts manager(z1,Jack) and manager(z2,Al) of Example 2 are presented in a two dimensional overall embedding space. With this technique we are able to compute embeddings for facts with labelled nulls without having a text document describing them and, at the same time, we keep these embeddings coherent with their semantic meaning. In order to include the relationship between nulls, in case of multiple labelled nulls, the pattern of the nulls is considered as well to create the embedding space. This means that if a fact has two or more labelled nulls that are the same, then only the facts that have a same constant value at the same position of the nulls will be considered from the input database. For example if the fact to embed is manager(z1,z1) with pattern manager(-1,-1), then only the facts that represent managers of themselves with pattern manager(1,1) will be taken into account to create the embedding space needed to find the centroid. In this case facts manager(Jack,Jack) or manager(John,John) are taken, while fact manager(Jack,John) is not.

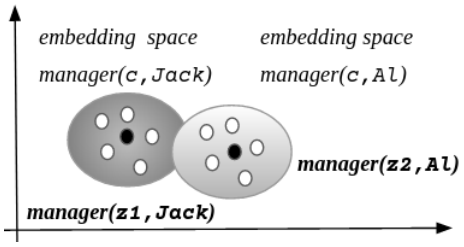

Figure 4: *On the left (dark grey) the locations of the facts manager, found in the input database, that have some constant c at position 0 and Jack at position 1. On the right (light grey) the locations with constant Al at position 1. It can be observed that, for predicate name manager, the two centroids in black are most similar to all the embeddings with same predicate name and constant at a certain position found in the input database.*

5. Isomorphisms and Unseen Facts

The CGNN is fully instantiated during the training phase. In fact it assumes a configuration that mirrors the chase graph G_{train} built during a chase procedure that has, as input, the database instance D_{train} and the set of rules Σ_{train}. As can be easily deduced, even though the domain is the same and $\Sigma_{train} = \Sigma_{test}$, D_{train} is always different from D_{test}, therefore we have to address the problem of which input layers of the CGNN to direct the new input represented by D_{test}.

Solution. To solve this problem we use subgraph isomorphism and the VF2 algorithm [7] on the two chase graphs G_{train} and G_{test} in order to find a mapping between the input layers of the trained network and the facts in D_{test} that is an isomorphism. The process of finding the mapping function can be suitably described by means of a State Space Representation (SSR). Each state s of the matching process can be associated to a partial mapping solution M(s), which contains only a subset of M. M(s) univocally identifies two subgraphs $G_{train}(s)$ and $G_{test}(s)$, obtained by selecting from G_{train} and G_{test} only the nodes included in M(s), and the edges between them. A transition from a generic state s to a successor s_0 represents the addition to the partial graphs associated to s in the SSR, of a pair (n, m) of matched nodes. In our context the algorithm always produces a mapping between G_{train} and G_{test}, representing the Vadalog program used during a test scenario. To avoid the possibility of not finding

an isomorphism, we work under the closed-world assumption [16] that let us considers the knowledge base as complete. This assumption guarantees to have a training Vadalog program that is always enough descriptive for every new test program provided. Since G_{test} is always smaller than G_{train}, by construction, then to find the mapping M is always a case of graph-subgraph isomorphism problem. The idea of using the isomorphism is based on the intuition that, given a fixed domain, facts from different database instances are similar if they are pattern-isomorphic, they use the same rules to produce pattern-isomorphic facts and the same properties are valid for the facts produced. These properties can be verified recursively when considering the chase graphs representing the two different reasoning process. To find this isomorphism, in addition to the syntactic feasibility rules presented by Cordella et al. [7], we add semantic feasibility rules to better represent the knowledge expressed in the chase graphs, since the performance of the CGNN is directly coupled to the quality of the isomorphism found, and to significantly reduce the number of possible states that are feasible in the algorithm.

5.1. Improving Isomorphism using Clustering

The subgraph isomorphism problem is NP complete [17]. In fact, the time complexity of this algorithm in the worst case is $\mathcal{O}(n!\,n)$ [7], where n is the number of nodes of G_{test}. To minimize the time needed in the average case we reduce the number of possible candidate pairs in P(s) at each step of the algorithm.

Computation Step of Candidate Pairs P(s). The main step of the VF2 algorithm is computing the set P(s) of all the possible candidate pairs to be added to the current state. Let us denote with $T_1^{out}(s)$ and $T_2^{out}(s)$ the sets of nodes, not yet in the partial mapping, that are the destination of edges starting from $G_1(s)$ and $G_2(s)$, respectively; similarly, with $T_1^{in}(s)$ and $T_2^{in}(s)$, we denote the sets of nodes, not yet in the partial mapping, that are the origin of edges ending into $G_1(s)$ and $G_2(s)$. The set P(s) is obtained by considering first the sets of the nodes directly connected to $G_1(s)$ and $G_2(s)$. In fact the set P(s) will be made of all the node pairs p(n,m), with $n \in T_1^{out}(s)$ and $m \in T_2^{out}(s)$. If one of these two sets is empty then the set P(s) is obtained by using all the node pairs p(n,m) with $n \in T_1^{in}(s)$ and $m \in T_2^{in}(s)$. All of the above sets may be empty in the presence of not connected graphs for some state s or if it is the first step of the algorithm. In this case, the set of candidate pairs of P(s) will be the set of all the pairs of nodes not contained neither in $G_1(s)$ nor in $G_2(s)$.

Clustering for Computation Step of Candidate Pairs P(s). We compute, using k-means clustering technique [18], a cluster for each input node $n \in G_{train}$, namely C_n, based on its embeddings. Then, whenever there is an input node $m \in G_{test}$ to be matched either at the start of the algorithm or when we are trying to match a new connected component, the set of candidate pairs making up P(s) will be the set of all the pairs of nodes p(n,m) where $m \in C_n$. In front of multiple choices, we choose the pair of input nodes in order of similarity starting from the pair of input nodes that are the most similar in their embedding, considering it the most promising pair. The number of clusters is initialized at a high number based on the number of input nodes in G_{train}. If an isomorphism is not found, then the number of clusters is halved, the VF2 algorithm is used again and this process is repeated until a matching is found. A matching is guaranteed to exist under the closed-world assumption when there is only one cluster. In Example 3 there is an high level description of the isomorphism matching algorithm combined with the use of clustering.

```
PROCEDURE Isomorphism Matching
  INPUT: two graphs G1 and G2.
  OUTPUT: the isomorphism matching between G1
          and G2.
  Initialize the number of clusters.
  Get a map with an embedding for every input
  node ∈ G1.
  WHILE (isomorphism is not found)
    Compute clusters using input node ∈ G1.
    Call VF2 matcher procedure with clustering
    to get an isomorphism.
    IF isomorphism is found THEN
        RETURN isomorphsim
    ELSE
        Halve the number of clusters.
    END IF
  END WHILE
END PROCEDURE Isomorphism Matching
```

Example 3: *The VF2 algorithm is called multiple times with a decreasing number of clusters for the input nodes until a match between G_{train} and G_{test} is found.*

6. Evaluation in an Economic Setting

In this section we describe the experimental evaluation we carried out to test the effectiveness of our model. We performed a link prediction task [19] in the domain of company ownership, using a dataset extracted from DBpedia [20]. The goal is to identify the unobserved true links in the KG between companies. The evaluation is presented in the form of an ablation study, in which we compare the result achieved using only Vadalog and the result of Vadalog supported by our CGNN.

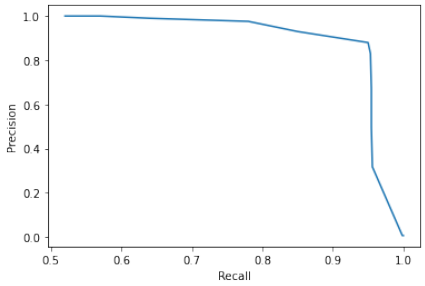

Figure 5: *Precision-Recall curve.*

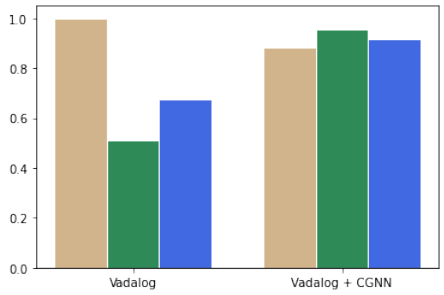

Figure 6: *Comparison between Vadalog and the CGNN. From left to right the measurements of precision (sand yellow), recall (green) and F-score (blue).*

6.1. Experimental Setting

We are interested in predicting related companies, based on the concepts of *ownership*, intended in terms of stock percentage holding, and *key person*, intended as executive people, core to the business operations of a certain company. In the former case we state that a company c_1 is related to another one c_2 if c_1 owns c_2 (the stock percentage holding is irrelevant and omitted) while in the latter we state that c_1 is related to c_2 if they have the same key person p. We used the DBpedia triples *(c_1, own, c_2)* and *(p, isKeyPersonOf, c)* to build the facts of D in the form of own(c_1,c_2) and keyperson(c,p). The golden set was obtained by running the program in Example 4 with Vadalog. The output of the program represents all true related companies. The chase graph obtained with this program is made up of a set of components that represent distinct sub-groups of related companies, causing rules 5 and 6 to contribute minimally to the total of related links produced. Moreover the dataset used is quite small with 215 *own* edges and 40 *keyperson* edges, for a total of 730 related edges. These numbers represent the 20% of the total dataset, the remaining 80% is used for the training set (65%) and the validation set (15%). The program used for validation and testing is composed of a subset of the rules used in Example 2, namely the rules from 1 to 4.

```
Σ = {
1. related(x,y):-keyperson(x,p),keyperson(y,p)
2. related(x,y):-keyperson(y,p),keyperson(x,p)
3. related(x,y):-own(x,y)
4. related(x,y):-own(y,x)
5. related(x,y):-own(x,z), own(z,y)
6. related(x,y):-own(y,z), own(z,x)
7. related(x,x):-related(x,y)
8. related(y,y):-related(x,y) }
```

Example 4: *Reasoning task to extract the golden set. The related companies are determined by the following constraints: (i) two companies are related if the have a key person in common (rule 1 and 2); (ii) two companies are related if the first one owns the second one or vice versa (rule 3 and 4); (iii) transitive property of ownership (rule 5 and 6); (iv) reflection property of related companies (rule 7 and 8).*

6.2. Ablation Study and Results

When we compare the result of Vadalog and the result of Vadalog plus the CGNN with the golden set, we intend to illustrate how much of the facts "related" only inferable from the rules not used at validation and test time are recovered with the support of the CGNN. To produce new facts related we use the embeddings produced by the CGNN as follows: for each pair (a,b) where a=related(X_1,Y_1) and b=related(X_2,Y_2), if the cosine similarity between the embeddings of a and b is greater than a threshold, then we produce the new facts related(X_1,Y_2) and related(X_2,Y_1). The threshold was tuned during validation time. Figure 5 illustrates the precision-recall curve.

Results. The results achieved in term of recall, precision and F-score, illustrated in Figure 6, highlight that our neurosymbolic approach was able to retrieve most of the false-negative related companies discovered by Vadalog, while maintaining a low number of false-positive, thus proving its effectiveness in recovering links inferred through logic rules. In fact, the recall goes from 51.10% (Vadalog) to 95.75% (Vadalog with CGNN) while maintaining 88.44% precision. Moreover we must consider that the 91 false positives of the CGNN could still be true predictions. In fact the CGNN has the potential to go over the expressiveness of the rules used to obtain the golden set and find more related connections. To prove this assertion we took 26 samples from the false positives and we manually checked if they were truly unrelated pairs or not. We found out that 20 out of the 26 samples were, instead, true predictions, where most of them were about a company related to banks or investment firms that were indeed its investors. Therefore, with this new insights, we can re-evaluate the results achieved by the CGNN and consider true positives the 77% (20/26) of the false positives. In this way our approach reaches 97.33% precision and 95.54 f-score.

Considerations. This data hint that our neurosymbolic approach has both the potential to go beyond the expressiveness of the logic rules and to recover information

from domain rules that have not been made explicit, but to have confirmation, the same experiments should be carried out in a much larger scale. In fact there were plenty of company and person entities in DBpedia that could contribute to generate the relationships own and keyperson, but we were forced to not consider a relevant number of them due to not being able to recover a meaningful text description for their embeddings to use as input for the CGNN.

7. Conclusion

We showed how to build the *Chase Graph Neural Network* (CGNN), a neurosymbolic model, simply by keeping track of the chase graph produced during the chase procedure. The CGNN is able to provide embeddings for each fact inferred during reasoning. We proposed a novel technique to handle facts with labelled nulls and a novel use of subgraph isomorphism to handle unseen data when using the CGNN. We demonstrated the effectiveness of our approach in a real-world financial domain by using the embeddings to discover new links in a KG. Despite the very promising results achieved, the solution presented is still a prototype. An obvious improvement would be to make use of the full potential of Datalog$^{\pm}$ rules by introducing recursion with the use of Long Short-Term Memory cells [21] as base structure for the neurons.

References

[1] L. Bellomarini, D. Fakhoury, G. Gottlob, E. Sallinger, Knowledge graphs and enterprise ai: The promise of an enabling technology, in: IEEE ICDE, 2019, pp. 26–37.

[2] L. Bellomarini, E. Sallinger, G. Gottlob, The vadalog system: Datalog-based reasoning for knowledge graphs, CoRR 11 (2018) 975–987.

[3] A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, A. Pieris, Datalog+/-: A family of logical knowledge representation and query languages for new applications, in: 2010 25th Annual IEEE Symposium on Logic in Computer Science, 2010, pp. 228–242.

[4] J. Zhang, B. Chen, L. Zhang, X. Ke, H. Ding, Neural, symbolic and neural-symbolic reasoning on knowledge graphs, AI Open (2021).

[5] L. C. Lamb, A. S. d'Avila Garcez, M. Gori, M. O. R. Prates, P. H. C. Avelar, M. Y. Vardi, Graph neural networks meet neural-symbolic computing: A survey and perspective, CoRR abs/2003.00330 (2020).

[6] D. Maier, A. O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, ACM Trans. Database Syst. 4 (1979) 455–469.

[7] L. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, IEEE PAMI 26 (2004) 1367–1372.

[8] L. Bellomarini, M. Benedetti, S. Ceri, A. Gentili, R. Laurendi, D. Magnanimi, M. Nissl, E. Sallinger, Reasoning on company takeovers during the covid-19 crisis with knowledge graphs, in: RuleML+ RR, 2020.

[9] L. Bellomarini, M. Benedetti, A. Gentili, R. Laurendi, D. Magnanimi, A. Muci, E. Sallinger, Covid-19 and company knowledge graphs: assessing golden powers and economic impact of selective lockdown via ai reasoning, arXiv (2020).

[10] T. Gogacz, J. Marcinkowski, A. Pieris, All-instances restricted chase termination: The guarded case, CoRR abs/1901.03897 (2019).

[11] G. Bebis, M. Georgiopoulos, Feed-forward neural networks, IEEE Potentials 13 (1994) 27–31.

[12] T. Kenter, A. Borisov, M. De Rijke, Siamese cbow: Optimizing word embeddings for sentence representations, arXiv (2016).

[13] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: ICLR, 2013.

[14] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: ICML, volume 32, 2014, pp. 1188–1196.

[15] J. Pennington, R. Socher, C. D. Manning, Glove: Global vectors for word representation, in: EMNLP, 2014, pp. 1532–1543.

[16] R. Reiter, On Closed World Data Bases, Springer US, Boston, MA, 1978, pp. 55–76.

[17] I. Wegener, Complexity theory: exploring the limits of efficient algorithms, Springer Science & Business Media, 2005.

[18] J. A. Hartigan, M. A. Wong, Algorithm as 136: A k-means clustering algorithm, J R Stat Soc Ser C Appl Stat 28 (1979) 100–108.

[19] A. Rossi, D. Barbosa, D. Firmani, A. Matinata, P. Merialdo, Knowledge graph embedding for link prediction, ACM 15 (2021) 1–49.

[20] S. e. a. Auer, Dbpedia: A nucleus for a web of open data, in: The Semantic Web, 2007, pp. 722–735.

[21] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (1997) 1735–1780.