Affinity Dependent Negative Sampling for Knowledge Graph Embeddings*

Mirza Mohtashim Alam[1,2], Hajira Jabeen[1,2], Mehdi Ali[1,2,3], Karishma Mohiuddin[2,3], and Jens Lehmann[1,2,3]

[1] Smart Data Analytics Lab, University of Bonn, Germany
[2] Department of Computer Science, University of Bonn, Germany
[3] Fraunofer IAIS

s6mialam@uni-bonn.de, jabeen@iai.uni-bonn.de, mehdi.ali@cs.uni-bonn.de, s6kamohi@uni-bonn.de, jens.lehmann@cs.uni-bonn.de

Abstract. Knowledge graph embedding system work in vector space where entities and relationships profoundly rely on positive and negative instances. Combination of a subject, a predicate and an object creates a triple, that make up a positive or negative assertion. Positive triples can be taken from the knowledge base, but additional steps are required to understand relations between entities and relations in order to create meaningful negative triples. In this paper we present an approach to create negative triples using affinities of triples. We create false triples meaningfully, rather than generating negative sample randomly. We rely on the closeness of the entities to create the false triples. We have compared our method with distributed negative sampling and random negative sampling and the results have been found comparable and efficient.

Keywords: Knowledge graph analysis · Negative sample creation · Embedding models · optimization techniques.

1 Introduction

Knowledge Graphs (KG) are a special kind of data where everything is presented in terms of subject (head), predicate (relations) and object (tail). Subjects/Objects (head or tail) can be considered as entities/nodes. Predicates (relations) can be considered as an edge/connection between two entities [1]. Most of the knowledge graphs encode the available information only, which is mostly the true cases. A lot of vector based embedding learning models require negative sampling to create data that contrasts with the already available data (discussed

* Supported by Smart Data Analytics Lab, University of Bonn
Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

in details in later sections). In this paper, we aim to develop an informative system of developing more prominent negative samples in order to aid the state of the art knowledge graph embedding models which are vastly used for link prediction tasks. This paper is divided into several sections. Sections 2 discusses about several closely related researches. In section 3, the two embedding models are discussed where we have tested our negative sampling technique. In section 4, we present the methodology and the algorithm. In section 5, we discuss about the implementation details (used datasets, hyperparameters etc.). In section 6, we discuss our experiment and results. Finally, section 6 provides an overview of the future directions.

2 Related Work

The transformation of KGs into the vector space using embedding models have become a renowned area of research. Lots of embedding learning approaches have been presented in the literature in the recent years. e.g, TransE [2], DistMult [3], Complex[4], TransH [5], RotatE [6] etc. Almost all of these models depend on the existing data encoded in the input KG. Following the closed world assumption the KG is considered as true, the knowledge not present in the KG cannot be considered false, but it can only be labelled as unknown. On the other hand, similar to most of the machine learning algorithms, The embedding models require negative examples to learn the embeddings efficiently. In order to cope with this, numerous negative sample generation methods have been proposed in the literature. In this section we will briefly cover a few related negative sampling methods. The widely used negative sampling method is randomly perturbs the head entity (subject) or tail entity (tail) in a triple with other available entities from the entity set. This approach was used in TransE [2]. This approach assumes that the data not present in the given RDF data can be a negative triple. Possible problems might be pulling out more relevant entities as replaceable entity for corruption [7]. Another approach includes corrupting positive triples for every relation [8], in this approach the graphs with less number of relations suffer from insufficient pool to corrupt positive instances [7]. Typed Sampling is the negative sampling based on entity and relation type by using their metadata [7]. Some other approaches discussed in [7] are Relational Sampling, Nearest Neighbor sampling, Near Miss sampling. Nearest Neighbor sampling and Near Miss sampling aim to draw out similar entities as a candidate for perturbation of triple subject or triple predicate. However, all of these approaches require pre-training of embedding. A negative sampling which pulls out the most prominent candidate as replaceable entity is Distributional Negative Sampling [9]. This approach does not require any pre-training in addition to corrupting the positive instances with more plausible entities. However, our implementation according to the algorithm given in distributional negative sampling paper [9], has a long training time that might have the potentiality for improvement of the duration of training, and computation. We therefore, tried to improve the training time while achieving a descent model performance.

3 Embedding Models Used

Model TransE [2] and DistMult [3] have been selected as the default models to demonstrate our negative sampling technique for their simplicity and less training effort.

3.1 Distmult

The bilinear scoring function of DistMult model is obtained by multiplying their entity vectors (head and tail) with their corresponding relation matrix which is diagonal [3]. The entities are considered as y_{e1}, y_{e2} and their corresponding diagonal relation matrix M_r, leads to the equation 1 [3].

$$g_b^r(y_{e1}, y_{e2}) = y_{y_{e1}}^T M_r y_{e2} \tag{1}$$

To minimize the margin based ranking loss, where, entities are denoted by $e1$, $e2$ and relation is denoted by r. For the corrupted triple and replaceable entities are $e1'$ and $e2'$. The positive formed triple is $(e1, r, e2) \in T$, where T is the total triple set. Thus corrupted triple is $T' = \{(e1', r, e2)|e1' \in E, (e1', r, e2)|e1' \notin T\} \cup \{(e1, r, e2')|e2' \in E, (e1, r, e2')|e2' \notin T\}$, where E is the total set of entities. Here, author also corrupted either head or tail of the given triple to obtain corrupted ones. Ranking loss was used to minimize the objective function. Equation 2 describes the loss function of [3].

$$L = \sum_{(e1,r,e2) \in T} \sum_{(e1',r,e2') \in T} max\{S_{(e1',r,e2')} - S_{(e1,r,e2)} + 1, 0\} \tag{2}$$

3.2 TransE

TransE uses similar randomly generated negative sampling techniques in their paper [2]. Their idea of scoring function is the edges (relations) corresponds to translation of embedding [2]. Means, in case of true triple, additive head and relation $h + l$ should be in the nearest vicinity of the tail t. They considered it as a distance function d, their scoring function is stated in equation 3 as stated in [2], where γ is the margin.

$$L = \sum_{(h,l,t) \in S} \sum_{(h',l,t') \in S_(h,l,t)} [\gamma + d(h + l, t) - d(h' + l, t')]_+ \tag{3}$$

4 Affinity Dependent Negative Sampling

As mentioned earlier, we improve upon an existing approach presented in [9] for the development of our negative sampling technique. In the training phase, for each of the positive triple in the KG, the aim is to generate C negative samples. In the training phase, Bernoulli sampling [10] is used to decide if the head (subject) or tail (object) is to be corrupted in a given triple. We use cosine similarity between the selected candidate (head or tail to be replaced by other entities) and all the other entities is calculated. This is stored as the affinity vector. Further, we made a affinity function based on the affinity vector M (the cosine similarity between the candidate entity and all the other entity). For each of the entities i in the entity set ξ, the affinity function is presented equation 4,

$$Fittness_i = \frac{M_i}{\sum_a M_j} \tag{4}$$

This affinity vector can act as a probability vector for each of the entities in the entity list. Using the affinity function as the base probability vector, we can randomly pull out the C replaceable entities based on the affinity. These C replaceable entities have most likely higher cosine similarity with candidate entity (which is to be replaced by other entities). Finally we form our negative triple by replacing the entity by the new chosen replaceable entities. We do this for all the triples in the batch and finally form the negative sample batch. For optimization, Adagrad (adaptive gradient descent) [11] optimizer has been used. We have used self adversarial loss function similar to [6] for both models. For negative sampling we have used the negative sample creation algorithm (1).

Algorithm 1: Affinity Dependent Negative Sampling

```
INPUT: Training set $S_{\{h,r,t\}}$ , Entity Set $\xi$, Relation set $R$, batch Size $\beta$,
Number of Negatives $C$
OUTPUT: For Given Batch $\beta_{\{h,r,t\}}$ return negative triples $N_{\{h',r,t'\}}$

Function Sample Negative($S_{\{h,r,t\}}$, $\xi$, $R$, $\beta$, $C$):
    for triple $t \in \beta_{\{h,r,t\}}$ do
        $candidate\_position = Bern(t, R)$
            ▷ bern negative sampling to decide whether head or tail corruption
        $candidate = t_{\text{candidate\_position}}$
        $\xi_c = \{\xi\}$ - candidate        ▷ subtract the candidate from total entity set
        $M_{\text{score}} = CosineSimilarity(candidate, \xi_c)$
        $M_{\text{score}} = max(0, M_{\text{score}})$
        for $i \in length(M_{score})$ do
            $probability\_fitness_i = M_{\text{score}_\text{i}} \div \sum_j M_{\text{score}_\text{j}}$
                                                ▷ generate the fitness vector
        $selected\_entities = random\_choice(\xi_c, C, probability\_fitness)$
        $NegativeTriple_{(\text{h'},\text{r},\text{t'})} =$
        $FormNegative(t, selected\_entities, candidate\_position)$
                                        ▷ form $C$ negative triples/positive
        $N_{\{\text{h'},\text{r},\text{t'}\}} = N_{\{\text{h'},\text{r},\text{t'}\}} \cup NegativeTriple_{(\text{h'},\text{r},\text{t'})}$
         ▷ append C negative triple per positive to the total batch negative set
    return $N_{\{\text{h'},\text{r},\text{t'}\}}$
```

In this algorithm, similar to [9] in first few lines the main aim is to, perform head or tail corruption based on the bern sampling. Later, we remove the candidate from the total entity set and perform cosine similarity between the candidate entity (which is to be replaced) and all the other entities from entity set ξ. We denote the cosine similarity vector as M_{score} from which the negative values from the cosine similarity vectors to 0 in the following line. Later, we generate probability fitness vector $probability\_fitness$ by equation 4. Further, using numpy's random choice, we achieve the C number $selected\_entities$ (With which we intend to corrupt the training instances). In the following lines, we corrupt the positive triple and form C number of corrupted triples finally forming a batch of corrupted triples against the positive ones. It is a very unusual case where all of the instances of the cosine similarity vector is negative. In this case we choose random replaceable entities out of entity set ξ.

5 Implementation details

5.1 Datasets and preprocessing

We have used medium to smaller datasets. Our datasets were in RDF (subject, predicate and object format). We have generated the dictionaries (entity to id, relation to id) for identification of specific entities and relations. We have used nations [12],kinship [13], and UMLS [14] dataset. These datasets are not very big and able to produce results in very short amount of time. These datasets were used in [15]. In our experiments the statistics of these datasets are provided in table 1

Table 1. Statistical information of the datasets

Dataset	# of entity	# of relation	# Training triple	# Test triple	# Validation triple
UMLS	135	46	5216	661	652
Kinship	104	25	8544	1074	1068
Nations	14	55	1592	201	199

Each of the entities and relations in the training and test set has the same identifier (the dictionary mentioned in the previous section).

5.2 Hardware and Tools

All of the results were achieved with a hardware of 16 GB ram, i7 4770 processor and Nvidia RTX 260 GPU. The experiments was built with Pytorch[1], numpy[2], scipy[3], pandas[4].

[1] https://pytorch.org/
[2] https://numpy.org/
[3] https://www.scipy.org/
[4] https://pypi.org/project/pandas/

5.3 Hyperparameters

For TransE the optimal hyper parameters were: learning rate = 0.1, gamma = 14 (for kinship, nations dataset), gamma = 10(for UMLS dataset), adversarial temperature = 0, negative sample number = 3. Normalization for embeddings and no regularization were done after each epoch for TransE.

For Distmult everything was same except the gamma which is 10 (for UMLS). In this case, normalization after each epoch has not been done. Regularization has been done for avoiding over-fitting.

For both cases self adversarial loss as in [6] and Adagrad optimizer [11] has been used.

5.4 Evaluation

Our evaluation focused on the model performances using three different negative samplings (i.e. random negative sampling, distributional negative sampling and affinity dependant negative sampling) on link prediction task.Filtered settings has been used as in [6]. Achievement of rankings of test triples against all the corrupted candidate triples (generation of the candidates are done by corrupting head or tail) has been done but since we are using the filtered settings candidate triples does not appear in training, validation or test set.

Standard evaluations for knowledge graph embedding model have been used in our researches which are mean rank (MR), mean reciprocal rank (MRR), Hit@1, Hit@3, Hit@5, and Hit@10.

6 Discussion and Result

Comparing with the Random Negative and Distributional Negative sampling [9], our method performed slightly better in certain cases. Authors in [9] showed that, the distributive negative sampling improves the performance of the model by choosing more salient negative samples [9]. The negative sampling method improves over time as with time the embedding vectors tends to get better. More appropriate negative sample is created as the steps go further. In case of distributional negative sampling it took a while for us for the training to be finished since at each iteration it calculates the cosine similarity. Later, it gets each entity (with which the correct ones to be replaced) by enumeration of the cosine similarity vector, which then acts acceptance and rejection probabilities for each of them. In this very case, full cosine similarity vector needs to be enumerated for getting C number of negative triples (or at least C times, but it may be possible that we might find more plausible negative sample if we traverse more than C). To mitigate this, we have calculated the fitness vector out of the cosine similarity vector and assigned it as a probability for choosing the plausible entities. By using the numpy standard library's choice function[5], it has been possible to avoid enumerating the whole cosine similarity vector and draw C number of

[5] https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.random.choice.html

replaceable entities at once (which takes fairly less amount of time). This function Generates C number of random sample from a given one dimensional array. If probability vector has been provided for the one dimensional array, then the probability vector becomes the basis of generation of C number of random samples. We have used the fitness vector as the probability argument for choosing C entities out of total entity set at once. The results for TransE and Distmult are stated in table 2 and 3 accordingly.

Table 2. Evaluation of Negative Sampling on TransE

Dataset	**Random Negative**	**DNS**	**ADNS**
Nations	MR: 5	MR: 5	MR: 4
	MRR: 26.96%	MRR: 33.79%	MRR: 42.42%
	Hit@1: 0%	Hit@1: 13.68%	Hit@1: 20.15%
	Hit@3: 40.05%	Hit@3: 37.31%	Hit@3: 53.98%
	Hit@5: 60.95%	Hit@5: 59.95%	Hit@5: 73.88%
	Hit@10: 93.53%	Hit@10: 94.53%	Hit@10: 98.01%
Kinship	MR: 9	MR: 13	MR: 8
	MRR: 25.66%	MRR: 24.89%	MRR: 28.47%
	Hit@1: 0.009%	Hit@1: 0.03%	Hit@1: 00.28%
	Hit@3: 39.80%	Hit@3: 38.13%	Hit@3: 47.77%
	Hit@5: 56.28%	Hit@5: 49.30%	Hit@5: 62.38%
	Hit@10: 76.44%	Hit@10: 64.20%	Hit@10: 78.91%
UMLS	MR: 3	MR: 3	MR: 2
	MRR: 64.47%	MRR: 72.90%	MRR: 80.21%
	Hit@1: 39.86%	Hit@1: 56.81%	Hit@1: 64.45%
	Hit@3: 88.12%	Hit@3: 86.76%	Hit@3: 95.54%
	Hit@5: 93.95%	Hit@5: 92.06%	Hit@5: 97.05%
	Hit@10: 97.13%	Hit@10: 96.44%	Hit@10: 98.03%

From the results, it can be seen that our model has slightly improved performance in terms of both models in most of the cases. The overall MR, MRR and Hit has a significant improvement for ADNS for Nations dataset in table 2. For Kinship and UMLS, ADNS has slight improvement in terms of MR, MRR and Hit@10. For Kinship and nations in table 2, some other Hits has significant improvement. In table 3, for Nations dataset, only Hit@10 has slight improvement. In this table no other significant improvement in model performance has been detected but the training time has been reduced significantly. The training time of our model is more than the random negative sampling but less than the distributional negative sampling in terms of our implementation with the original paper. Figure 1 is indicating the convergence of loss and taken time for UMLS dataset during 1000 steps which also indicates the training time with each of the negative sample discussed in legend.

Table 3. Evaluation of Negative Sampling on DistMult

Dataset	Random Negative	DNS	ADNS
Nations	MR: 3 MRR: 65.34% Hit@1: 49.50% Hit@3: 76.12% Hit@5: 88.31% Hit@10: 99.520%	MR: 2 MRR: 82.32% Hit@1: 73.63% Hit@3: 88.06% Hit@5: 94.53% Hit@10: 99.75%	MR: 2 MRR: 79.48% Hit@1: 68.91% Hit@3: 86.32% Hit@5: 93.78% Hit@10: 100.00%
Kinship	MR: 6 MRR: 48.45% Hit@1: 33.43% Hit@3: 54.24% Hit@5: 65.27% Hit@10: 85.38%	MR: 5 MRR: 57.19% Hit@1: 44.88% Hit@3: 61.78% Hit@5: 71.09% Hit@10: 83.80%	MR: 5 MRR: 58.74% Hit@1: 47.63% Hit@3: 61.87% Hit@5: 71.09% Hit@10: 85.94%
UMLS	MR: 8 MRR: 48.03% Hit@1: 34.72% Hit@3: 54.84% Hit@5: 64.07% Hit@10: 76.48%	MR: 5 MRR: 71.15% Hit@1: 63.69% Hit@3: 74.21% Hit@5: 78.97% Hit@10: 86.16%	MR: 6 MRR: 60.72% Hit@1: 49.24% Hit@3: 67.96% Hit@5: 74.96% Hit@10: 84.87%

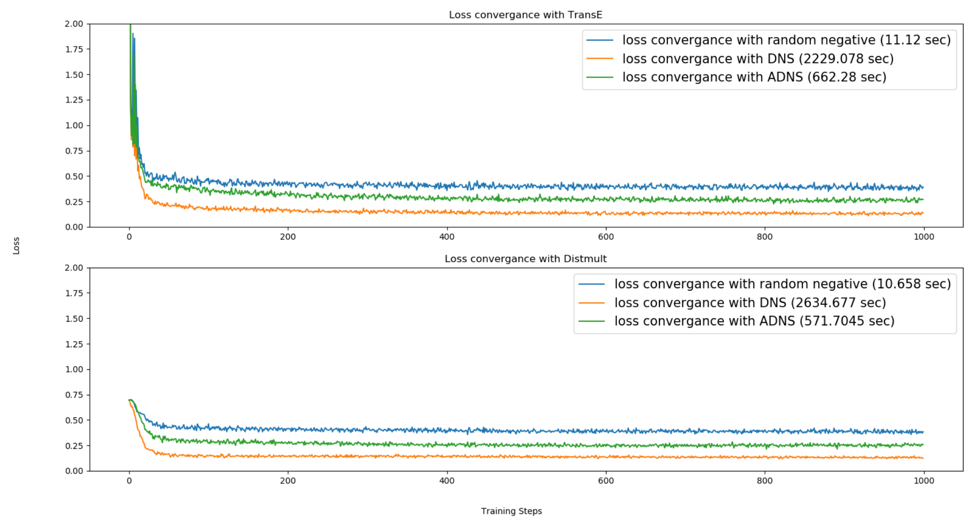

Fig. 1. Convergence of Loss with both models

7 Conclusion and Future work

Learning relational data plays a vital role in various aspects such as drug target interactions, social network analysis, recommender systems, decease analysis wherever, entities (i.e. movies, people, locations) and relations (i.e. friend of, located in, causes) are connected by edges. Learning these relational data needs negative samples to contrast with its instances. Apart from creating random negatives, other approaches can be time consuming since it would add an additional effort to fetch desired negative samples. In this research, we provided a glimpse how better negative sample can be generated without sacrificing too much time with a decent performance.In our experiment, one model i.e. TransE has good model performance in these data sets whereas DistMult did not provided significant improvement in terms of model performance but it reduced the training time. As it was discussed earlier in section 6., Though both model showed improved or similar or slightly less model performances. In each cases, both of the models had improved training time over DNS (in our implementation of both models). In future we aim to use other similarity measures apart from cosine similarity. We also want to expand our experiment with larger datasets like Freebase 15k and Wordnet 18 and also include more recent KG models like rotate, complex etc.

References

1. Dettmers, T., Minervini, P., Stenetorp, P., & Riedel, S. (2018, April). Convolutional 2d knowledge graph embeddings. In Thirty-Second AAAI Conference on Artificial Intelligence.
2. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In Advances in neural information processing systems (pp. 2787-2795).
3. Yang, B., Yih, W. T., He, X., Gao, J., & Deng, L. (2014). Learning multi-relational semantics using neural-embedding models. arXiv preprint arXiv:1411.4072.
4. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., & Bouchard, G. (2016). Complex embeddings for simple link prediction. International Conference on Machine Learning (ICML).
5. Wang, Z., Zhang, J., Feng, J., & Chen, Z. (2014, June). Knowledge graph embedding by translating on hyperplanes. In Twenty-Eighth AAAI conference on artificial intelligence.
6. Sun, Z., Deng, Z. H., Nie, J. Y., & Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. arXiv preprint arXiv:1902.10197.
7. Kotnis, B., & Nastase, V. (2017). Analysis of the impact of negative sampling on link prediction in knowledge graphs. arXiv preprint arXiv:1708.06816.
8. Socher, R., Chen, D., Manning, C. D., & Ng, A. (2013). Reasoning with neural tensor networks for knowledge base completion. In Advances in neural information processing systems (pp. 926-934).
9. Dash, S., & Gliozzo, A. (2019). Distributional Negative Sampling for Knowledge Base Completion. arXiv preprint arXiv:1908.06178.

10. Wang, Z., Zhang, J., Feng, J., & Chen, Z. (2014, June). Knowledge graph embedding by translating on hyperplanes. In Twenty-Eighth AAAI conference on artificial intelligence.
11. Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(Jul), 2121-2159.
12. Rummel, R. J. (1968). Dimensionality of Nations Project (No. 405846). HAWAII UNIV HONOLULU DEPT OF POLITICAL SCIENCE.
13. Denham, W. W. (1973). The detection of patterns in Alyawara nonverbal behavior (Doctoral dissertation, University of Washington, Seattle.).
14. McCray, A. T. (2003). An upper-level ontology for the biomedical domain. International Journal of Genomics, 4(1), 80-84.
15. Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T., & Ueda, N. (2006, July). Learning systems of concepts with an infinite relational model. In AAAI (Vol. 3, p. 5).