A Probabilistic Framework for Knowledge Graph Data Augmentation

Jatin Chauhan[*] [†]
IIT Hyderabad
chauhanjatin100@gmail.com

Priyanshu Gupta[*] [‡]
IIT Kanpur
priyanshu.42g@gmail.com

Pasquale Minervini
University College London
p.minervini@gmail.com

Abstract

We present NNMFAug, a probabilistic framework to perform data augmentation for the task of knowledge graph completion to counter the problem of data scarcity, which can enhance the learning process of neural link predictors. Our method can generate potentially diverse triples with the advantage of being efficient and scalable as well as agnostic to the choice of the link prediction model and dataset used. Experiments and analysis done on popular models and benchmarks show that NNMFAug can bring notable improvements over the baselines.

1 Introduction

The most widely used representation of *Knowledge Bases (KBs)* is in the form of *Knowledge Graphs (KGs)* where the nodes represent entities that are connected by relations in form of a directed graph. Extensive research in the past decade has shown that these KGs can be extremely useful for many core NLP tasks such as relation extraction [19, 25], summarization [12], question answering [3], dialog systems [16], recommender systems [29] and many more due to their simplistic structure and the ability to abstract out facts and knowledge.

Despite their success, a major drawback of KGs is their incompleteness [8]. Since the actual number of valid KG triples can be extremely large, ensuring that they are complete can be a daunting task, if done manually. This can in-turn stagnate the improvements on downstream tasks. The task of *Knowledge Graph Completion* [2] extensively focuses on tackling this issue by learning models, commonly known as *link predictors*, that can complete any triple with partial information. More recently, neural network based methods [24, 6, 23, 1], commonly referred to as *neural link predictors*, have become the state of the art for KG completion task. However, since these models are supervised learners, their ability is directly tied to the amount of training data available.

Recent threads of research present empirical and theoretical arguments to suggest that data augmentation can improve the performance of deep learning models [15, 4] by non-trivial margins while also leading to improved generalization. [26, 11]. Inspired from these works, we propose *NNMFAug*, a novel method to perform *data augmentation* over knowledge graphs to improve the performance of neural link predictors.

[*] equal contribution
[†] corresponding author is currently at Google AI
[‡] corresponding author is currently at Microsoft Research

Accepted at Data-Centric AI Workshop (DCAI 2021), (NeurIPS 2021).

We develop a *probabilistic* framework that is *agnostic* to the choice of link prediction model and dataset from which new and diverse triples can be sampled while ensuring scalability and efficiency. Further, we present a new training routine to gradually increase the number of these newly sampled triples that are augmented to the training set as a function of the training epochs completed. Experiments and analysis done on popular neural link predictors and benchmarks show that our technique can bring notable improvements over the baselines trained on the available training data only.

2 Method

We first provide the formulation of the probabilistic framework that fits a distribution over the set of all possible triples and then we present an efficient mechanism to sample from this distribution over the triples. Lastly, we describe a training routine that we used to effectively utilise these augmented triples in training link predictors.

2.1 Probabilistic Formulation

Borrowing notation from [7], we define *Knowledge Graph* $\mathcal{G} = \{(h, r, t)\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ as a set of triples of the form (h, r, t) such that $h, t \in \mathcal{E}$, $r \in \mathcal{R}$ and $h \neq t$, ie, no self loops in the graph. It can thus be viewed as a *directed graph* with *head entity*(h) and *tail entity*(t) as the nodes and the *relation type*(r) as corresponding edge label.

There can be many ways to factorize the distribution of the triples, one of which is as follows:

$$p(h, r, t) = p(h, t) * p(r|h, t) \tag{1}$$

where $p(h, t)$ is the distribution over the possible edges in the graph and ($p(r|h, t)$) is the distribution of the relations conditioned on an edge denoted by (h, t).
Since knowledge graphs are known to be inherently sparse (statistics of some benchmarks are provided in table 2) and the entities have a certain "type" that categorizes them semantically, we further propose to model the entities in the KG as a set of *clusters*, where all the generated clusters are disjoint. We thus arrive at the following factorization:

$$\begin{aligned} p(h, r, t) &= p(r|h, t) * \sum_{\forall cluster} p(cluster) * p(h, t|cluster) \\ &= p(cluster_i) * p(h, t|cluster_i) * p(r|h, t) \end{aligned} \tag{2}$$

where $cluster_i$ is the cluster containing a given entity tuple (h, t).

2.2 Generating Entity Clusters

We now define the matrices: $A \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{R}|}$ (call it *head-relation* matrix) and $B \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{R}|}$ (*tail-relation* matrix) as follows:

$$A = [a_{ij}]_{|\mathcal{E}|\times|\mathcal{R}|}, \;\; a_{ij} = |\{(i, j, k) \in \mathcal{G}\}| \tag{3}$$

$$B = [b_{ij}]_{|\mathcal{E}|\times|\mathcal{R}|}, \;\; b_{ij} = |\{(k, j, i) \in \mathcal{G}\}| \tag{4}$$

One way to generate disjoint entity clusters is by using an algorithm such as higher order spectral clustering [14], that can find cuts in the KG where the set of nodes in each cut form a weakly connected component. However note that it is non-trivial to achieve this clustering over the original KG directly, since there is no natural way to assign weights to the edges (where each edge is a relation type). We rather consider the digraph generated by taking the entities as nodes and the elements of the affinity matrix C (eq 5 below) as corresponding weighted edges. Intuitively speaking, matrix C can be viewed as a co-occurence matrix of entity pairs marginalized over all relation types. Formally, it can be represented as follows:

$$C = AB^T \tag{5}$$

Although spectral clustering over C can provide the desired entity clusters but despite its suitability to our problem, spectral clustering is computationally expensive with a high memory and time overhead which renders it impractical for the KGs with large number of entities. Alternately, seeking inspiration from the GloVe [21] algorithm, we first generate lower dimensional representations of

the entities by performing **Non-Negative Matrix Factorization (NNMF)** [5] (a brief introduction is provided in section A.3) of the affinity matrix C. In order to utilize all available information, the two matrices that are obtained from NNMF, represented as W_1 and W_2 here, are concatenated along the column dimension, denoted as $W^{'}$, and passed through a standard clustering method that operates on euclidean space to generate a partition denoted by $\mathcal{K} = \{K_i\}_{i=1}^{N}$, such that $\bigcup_{i=1}^{N} K_i = \mathcal{E}$ and $\forall i, j; i \neq j \rightarrow K_i \cap K_j = \emptyset$. The size of $\mathcal{K}$, ie, the number of clusters is a used defined hyperparameter. We use *Agglomerative clustering* [28] over $W^{'}$ as it provides a reasonable tradeoff between speed and quality.

2.3 Sampling

To generate and subsequently augment new triples to the training set, we utilize the factorisation of $p(h, r, t)$ proposed in equation (2) by first estimating the distributions p($cluster_i$), $p(h, t|Cluster_i)$ and $p(r|h, t)$ from the statistics of the training data. To simplify the computation, we assign a uniform distribution to $p(cluster)$, thus the probability of selecting all clusters is equal. Similarly, we also assign a uniform distribution to all pairs of entities (h, t) in a given cluster i.

Lastly, to estimate $p(r|h, t)$ we use matrices A and B. We perform element wise multiplication of the row of A corresponding to head entity h with the of row of B corresponding to tail entity t. This provides us a vector $\vec{d} \in \mathbb{R}^{|\mathcal{R}|}$ which is further normalized by dividing the entry in each dimension of $\vec{d}$ by the sum of all entries in $\vec{d}$ such that it becomes a probability simplex and is then used to sample the relation type r, weighted by its corresponding probability value in normalised $\vec{d}$, finally giving us a new triple (h, r, t).

The above procedure is repeated until we obtained a desired number of triples to augment. We denote the set of newly generated triples by $\mathbf{S}$. The complete pipeline is provided in algorithm 1.

Algorithm 1: Proposed NNMFAug Method

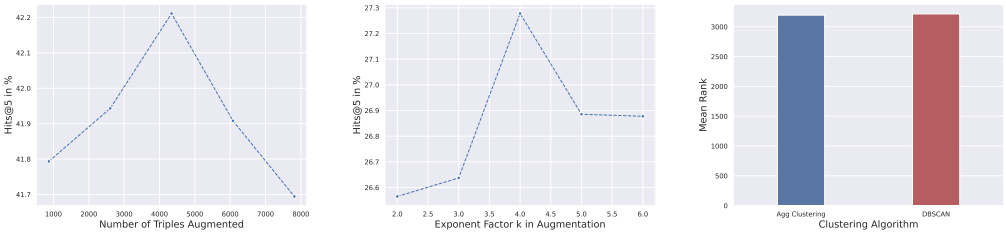
Input : Knowledge Graph $\mathcal{G}$, number of clusters N, number of triples to generate L
Initialize: $\mathbf{S} = \{\}$; matrices A, B, C ▷ via Eq 3, 4 and 5 respectively
$W_1, W_2 \leftarrow$ NNMF(C) ▷ Eq 7
$W' \leftarrow [W_1, W_2]$ ▷ Column-wise Concatenation
$\mathcal{K} \leftarrow$ AggClustering(W', N) ▷ Agglomerative Clustering of W'
while |**S**| < L **do**
  Generate New Triple (h', r', t'), given partitioning $\mathcal{K}$ ▷ Section 2.3
  $\mathbf{S} \leftarrow \mathbf{S} \bigcup \{(h', r', t')\}$ ▷ Set Union
end
Output : **S**

2.4 Routine to Monotonically Increase Augmented Data Size

Rather than augmenting the training data with the entire set $\mathbf{S}$, we follow a routine that monotonically increases the number of new triples r, added per epoch, as the training progresses. r is calculated as $(\frac{e}{E})^k \times |\mathbf{S}|$, where e is the current training epoch, E is the total number of training epochs, $k \in \mathbb{Z}^+$ is a hyperparameter and $|\mathbf{S}|$ is the size of set $\mathbf{S}$. We empirically observed that this routine helps as the augmented triples can be sometimes noisy and gradually introducing them to the model can help the model generalize better. Further analysis for the hyperparameter k has been done in section 4.

3 Experiments

We evaluate the efficacy of the proposed data augmentation method on two widely used neural link prediction models: *TransE* [2] and *RotatE* [23] over two widely used datasets: *Wordnet18RR* (WN18RR) [6] and *DeepLearning50a* (DL50a) [22]. A brief introduction to neural link predictors is provided in appendix A.1. The statistics for the datasets are provided in table 2 and the evaluation protocol is briefly described in section A.2. All the experiments have been performed using PyKg2Vec library on a single GPU with 8 GB cuda memory.



(a) TransE on Wordnet18RR dataset.

(b) TransE on DL50a dataset.

(c) RotatE on Wordnet18RR dataset.

Figure 1: Analysis for the variation of *Number of Triples Augmented* (left subfigure), *Factor k in Augmentation Routine* (middle subfigure) and the *Clustering Algorithm Used* (right subfigure).

Table 1 report the results (averaged over 5 runs) for each of the KG model-dataset combination. The suffix "Baseline" represents the original model performance whereas "NNMFAug" shows the model performance with our augmentation strategy.

Table 1: Results for various evaluation metrics (section A.2). Best values are highlighted in bold.

Dataset	Model	H@1↑	H@3↑	H@5↑	H@10↑	MRR↑	MR↓
DL50a	TransE-Baseline	8.13	19.27	25.03	33.21	0.1597	506
	TransE-NNMFAug	**9.97**	**21.76**	**27.25**	**34.25**	**0.1797**	**490**
	RotatE-Baseline	35.05	45.62	49.56	54.82	0.4221	156
	RotatE-NNMFAug	**35.62**	**46.08**	**50.10**	**55.34**	**0.4276**	**152**
WN18RR	TransE-Baseline	1.24	**35.41**	41.85	47.29	**0.1974**	3920
	TransE-NNMFAug	**1.31**	35.28	**41.99**	**47.48**	0.1971	**3766**
	RotatE-Baseline	39.59	47.89	51.10	55.39	0.4527	**3115**
	RotatE-NNMFAug	39.59	**48.09**	**51.14**	**55.67**	**0.4537**	3195

4 Analysis

In this section, we quantitavely analyze the model performance against some of the important hyperparameters of the augmentation method.

1) Size of the Augmented data: The number of triples augmented to the training set has a direct affect on the downstream model performance, as shown in figure 1a. It is interesting to note the *Inverted V-shaped* curve for the metrics. Improved downstream performance of neural networks via more augmented data is a well-known phenomenon [27, 10], however, in our case it is evident that augmentation beyond a certain point can affect the models negatively. We hypothesize that this is due to presence of some *false positive triples*, which are generated as a consequence of sampling (a probabilistic procedure), that can hinder the learning process because of excess noise. We also point that the location of the peak of the curve can vary depending upon the dataset.

2) Exponent factor in Augmentation Routine: The exponent k (section 2.4) that governs the number of generated triples augmented to the training data per epoch, also has a direct impact on the downstream performance, as shown in figure 1b. Here as well, we observe that gradually increasing the factor first improves the metrics to a peak value, post which the performance decreases, showing a similar *Inverted V-shaped* curve trend. It follows a similar reasoning as previous subsection that augmenting the training data with larger number of triples in the early phases of training can hinder the learning due to presence of some false positive triples. Thus, its important to learn from original training data in the early phases and follow a monotonic increment routine in augmentation.

3) Clustering Algorithm: We evaluate the performance of the models against the clustering algorithm used to group entities into multiple clusters. We compare the performance of two widely known clustering algorithms: *Agglomerative Clustering* [28] (used in this work) and *DBSCAN* [9]. From the comparison shown in figure 1c, its evident that our strategy is *less succeptible* to the clustering algorithm used and thus has a wide applicability.

5 Conclusion

In this work, we presented a novel data augmentation strategy for the task of link prediction in Knowledge Graphs named NNMFAug, which is agnostic to a specfic method and dataset as well as capable of performing data augmentation efficiently in an offline manner while utilizing multiprocessing. We show that NNMFAug provides consistent gains over multiple dataset and model combinations as well as anticipate that this work will draw attention and also pave way for more probabilistic as well as deterministic methodologies in this understudied space.

References

[1] Ivana Balazevic, Carl Allen, and Timothy Hospedales. TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5185–5194, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1522. URL https://aclanthology.org/D19-1522.

[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

[3] Antoine Bordes, Jason Weston, and Nicolas Usunier. Open question answering with weakly supervised embedding models. *CoRR*, abs/1404.4326, 2014. URL http://arxiv.org/abs/1404.4326.

[4] Shuxiao Chen, Edgar Dobriban, and Jane H. Lee. A group-theoretic framework for data augmentation. In *NeurIPS*, 2020.

[5] Andrzej Cichocki and Anh-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721, 2009.

[6] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. *CoRR*, abs/1707.01476, 2017. URL http://arxiv.org/abs/1707.01476.

[7] Agnieszka Dobrowolska, Antonio Vergari, and Pasquale Minervini. Neural concept formation in knowledge graphs. In *3rd Conference on Automated Knowledge Base Construction*, 2021. URL https://openreview.net/forum?id=V61-62OS4mZ.

[8] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 601–610, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329569. doi: 10.1145/2623330.2623623. URL https://doi.org/10.1145/2623330.2623623.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[10] Alex Hernández-García and Peter König. Further advantages of data augmentation on convolutional neural networks. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 95–103, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01418-6.

[11] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[12] Luyang Huang, Lingfei Wu, and Lu Wang. Knowledge graph-augmented abstractive summarization with semantic-driven cloze reward. *CoRR*, abs/2005.01159, 2020. URL https://arxiv.org/abs/2005.01159.

[13] Timothee Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2863–2872. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/lacroix18a.html.

[14] Steinar Laenen and He Sun. Higher-order spectral clustering of directed graphs. *CoRR*, abs/2011.05080, 2020. URL https://arxiv.org/abs/2011.05080.

[15] Cheng Lei, Benlin Hu, Dong Wang, Shu Zhang, and Zhenyu Chen. A preliminary study on data augmentation of deep learning for image classification. In *Proceedings of the 11th Asia-Pacific Symposium on Internetware*, Internetware '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450377010. doi: 10.1145/3361242.3361259. URL https://doi.org/10.1145/3361242.3361259.

[16] Yi Ma, Paul A. Crook, Ruhi Sarikaya, and Eric Fosler-Lussier. Knowledge graph inference for spoken dialog systems. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5346–5350, 2015. doi: 10.1109/ICASSP.2015.7178992.

[17] Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013. URL http://arxiv.org/abs/1310.4546.

[18] Pasquale Minervini, Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Adversarial sets for regularising neural link predictors. *CoRR*, abs/1707.07596, 2017. URL http://arxiv.org/abs/1707.07596.

[19] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL https://aclanthology.org/P09-1113.

[20] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, Jan 2016. ISSN 1558-2256. doi: 10.1109/jproc.2015.2483592. URL http://dx.doi.org/10.1109/JPROC.2015.2483592.

[21] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[22] Ahmet Salih. Link prediction with deep learning models. *UC Irvine Electronic Theses and Dissertations*.

[23] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ.

[24] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2071–2080. JMLR.org, 2016.

[25] Shikhar Vashishth, Rishabh Joshi, Sai Suman Prayaga, Chiranjib Bhattacharyya, and Partha Talukdar. RESIDE: Improving distantly-supervised neural relation extraction using side information. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1257–1266, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1157. URL https://aclanthology.org/D18-1157.

[26] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. *arXiv preprint arXiv:1805.12018*, 2018.

[27] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks, 2019.

[28] Marie Lisandra Zepeda-Mendoza and Osbaldo Resendis-Antonio. *Hierarchical Agglomerative Clustering*, pages 886–887. Springer New York, New York, NY, 2013. ISBN 978-1-4419-9863-7. doi: 10.1007/978-1-4419-9863-7_1371. URL https://doi.org/10.1007/978-1-4419-9863-7_1371.

[29] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 353–362, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939673. URL https://doi.org/10.1145/2939672.2939673.

A Appendix

Table 2: Dataset statistics

Dataset	# Entities	# Relations	# Triples			
			Training	Validation	Test	Total
WN18RR	40,943	11	86,835	3034	3134	93,003
DL50a	2705	20	6000	770	1249	8019

A.1 Neural Link Predictors

Link Predictors (in the KG setting) are models trained to maximize the likelihood of the triples in training data and further used to assign likelihood of new triples being correct during inference time (more details in section A.2). Neural Link predictors can thus be seen as deep learning based link predictors which essentially learn low dimensional representations for the entities (represented by $\mathbf{E}^{|\mathcal{E}|\times d}$) and relations (represented by $\mathbf{R}^{|\mathcal{R}|\times d}$) in the KG, along with some other trainable parameters (represented by θ), through back-propagation[20]. With these trainable parameters, a neural link predictor defines a **scoring function** f (mostly heuristic) over the embedding vectors $\mathbf{h}$, $\mathbf{r}$, $\mathbf{t}$ of an input triple (h, r, t) that are indexed from $\mathbf{E}$ and $\mathbf{R}$ respectively such that $f(\mathbf{h}, \mathbf{r}, \mathbf{t}; \theta) \rightarrow \mathbb{R}$; assigns a likelihood to the triple being correct. While there is a rich literature surrounding the designs of these link predictors, in this work we have focused on two popular models: *TransE* and *RotatE*. Their scoring functions are provided in table 3. Note that the d dimensional embedding space can be *real valued* (see TransE in table 3) or *complex valued* (see RotatE in table 3).

It is also noteworthy that along with the positive triples provided in the training data, these models are fed a large number of *negative* or *corrupt* triples generated by the same mechanism as described in section A.2 so that the models can learn to distinguish correct triples from the incorrect ones. This mechanism is usually referred as *negative sampling* and is intuitively same as the *negative sampling* procedure of *Word2Vec* algorithm [17].

Table 3: The scoring functions $f(h, r, t)$ and embedding constraints of *TransE* and *RotatE* models. $\mathbb{C}$ represents the complex space and d denotes the embedding dimensions.

Model	Score function	Embedding Constraints
TransE	$-\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|$	$\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$
RotateE	$-\|\mathbf{h} \circ \mathbf{r} - \mathbf{t}\|^2$	$\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^d, \|\mathbf{r}_i\| = 1$

A.2 Evaluation Protocol

For a given triple (h, r, t) in the test set, either the head entity h or the tail entity t is assumed to be missing and the aim is to predict the missing entity given the relation and the other entity. Without of loss of generality, lets assume that h is missing. First, a set of $\mathcal{E} - 1$ corrupt triples is generated by appending each entity $e \in \mathcal{E} \backslash h$ to (r, t), generating a total of $\mathcal{E}$ triples, including the original correct test triple. These triples are then passed through the neural link predictor and subsequently sorted in descending order of the scores. We then obtain the rank of correct triple (h, r, t). The same procedure is repeated for both the head and tail entities across the entire test set and the results are averaged to finally report the Mean Reciprocal Rank (MRR), Mean Rank (MR) and the percentage of correct triples in the top R ranks (Hits@R) for $R = 1, 3, 5$ and 10, after being sorted. For Mean Reciprocal Rank as well as the Hits@R metrics higher values are better whereas for Mean Rank lower is better.

A.3 Non Negative Matrix Factorization

Non-Negative Matrix Factorization is a decomposition of a matrix $M^{m\times n}$ into 2 component matrices $W_1^{m\times p}$ and $W_2^{p\times n}$ such that

$$M = W_1 W_2 \tag{6}$$

with the constraint that all the three Matrices have **non-negative elements**. While the factorization is not necessarily unique, polynomial closed form solutions can be calculated by enforcing additional

constraints on W_1 and W_2 matrices. However, in practice approximate methods prove to be a Time and Memory efficient alternative. In this work, we have used the NNMF implementation provided by scikit-learn library [4] that uses alternating minimization of W_1 and W_2 to minimize the objective function ($\mathcal{L}(W_1, W_2, M)$) below:

$$\mathcal{L}(W_1, W_2, M) = 0.5 * \|M - W_1 W_2\|_{fro}^2 + \alpha * \Omega(W_1, W_2) \tag{7}$$

where, fro represents the *frobenius* norm of the matrix, α is a hyper-parameter and $\Omega(W_1, W_2)$ is the regularization term such that:

$$\Omega(W_1, W_2) = c * (\|vec(W_1)\|_1 + \|vec(W_2)\|_1) + 0.5 * (1 - c) * (\|W_1\|_{fro}^2 + \|W_2\|_{fro}^2) \tag{8}$$

where c is another hyper-parameter and $\|\|_1$ is the L1 norm.

Time complexity of this algorithm is $\mathcal{O}(mpn \times q)$, where q is the number of iterations performed during alternate minimization.

A.4 Related Work

There have been a few prior works performing *data augmentation* for link prediction in knowledge graphs, however it still remains a fairly new and explored research area. [13] introduced the concept of augmenting the training data by adding new triples consisting of *inverse relations* which improves the performance of neural link predictors over multiple benchmarks. In another work, [18] proposed a method to generate sets of adversarial examples that maximizes an inconsistency loss which encodes specific background knowledge. In a more recent work, [7] revisit the notion of learning novel concepts in Knowledge graphs in a more principled way. More succinctly, they propose a method to cluster the entities where each cluster represents a concept. There are fundamental differences between our work and theirs in that: (i) they introduce a new "relation type" per cluster to generate new triples whereas we utilize the existing set of entities and relation types to generate new triples in a probabilistic manner; (ii) our method follows along the lines of Glove [21] algorithm since we seek latent entity embedding vectors via NNMF where the loss is minimized based on co-occurence of entity-relation pairs while accounting for the edge direction.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html