Embedding Logical Queries on Knowledge Graphs

Ioannis Fotopoulos
Master's degree student in the Department of Computing Science & Engineering, University of Ioannina, Ioannina, Greece, ifotopoulos@cs.uoi.gr

Christodoulos Asiminidis
Master's degree student in the Department of Computing Science & Engineering, University of Ioannina, Ioannina, Greece, chasiminidis@cs.uoi.gr

ABSTRACT: Knowledge graphs provide information about the world and have already become necessary resources to support applications that are clearly related to artificial intelligence. Knowledge graphs consist of relations between the entities. In this work, we consider the issue of embedding these entities and relations in a low-dimensional embedding space. Till today, we managed to predict simple edges, simple logical queries. However, predicting missing or unobserved edges is still hard to solve. Our goal is to implement and test a model which is easy to train, test, validate and give answers on incomplete graphs. For instance, given an incomplete medical knowledge graph, we may want to answer the query – what drugs are most likely to target receptors involved with diseases X and Y?-Also, for an incomplete knowledge graph it is a good technique to construct negative examples to reduce false negative labels in training. We suggest a method that reduces the possibility of false negative observations. We conduct experiments on low-dimensional embeddings of knowledge graphs, feature extraction, machine learning technique on two real-world biomedical data sets, the one is a subset of the Grank AI dataset and the second one comes from Therapeutic Target Database.

CCS CONCEPTS

- Computing →Neural networks, machine learning, graph theory
- Information systems →Networks

KEYWORDS

Knowledge graphs, embedding graphs, neural networks, statistical learning

1. Introduction

In recent years, graphs have started becoming popular and being part of many applications such of the real world such as social networks, citation graphs in research fields, operating system graphs, knowledge graphs etc. Connected, disconnected, bipartite, weighted, directed and undirected graphs are some of the types of a graph. Large data sets in conjunction with the fact that graphs gather information on them has made embedding graphs very popular in this research fields which are computationally able to face big-scale issues. Graph embedding has different ways of solving the high computation and space cost due to large-scale up that graphs suffer from. Precisely, graph embedding takes a graph as an input and gives a low-dimensional vector which is a subset of the initial graph or the whole graph. Graph embedding output can be categorized into four types as follows: node embedding, edge embedding, hybrid embedding and whole-graph embedding. The one criterion on which output embedding will be beneficial depends on the needs of the application.

Analyzing graphs is a method for extracting information. One of the main concerns in this field is that graphs can scale up to very large scales. Hence, they are computationally expensive, require plenty of memory and cost a lot of space. That is the point where embedding graphs come in handy. They are able to transform a large graph into a low-dimensional space by which the information included into the graph is preserved. In that way, graph methods can be computed computationally effectively.

In this article, the focus is embedding graphs which correspond to a set of nodes and edges that are strongly correlated. For instance, given an incomplete medical knowledge graph that has known interactions amidst drugs, disease and/or hormone/hormone receptors, one possible query would be: "what drugs are possible to be related to hormone/hormone receptors that have both diseases X and Y". In this query, we consider hormone/hormone receptors as nodes, hormone/hormone receptors as edges and drugs as target variables.

2. Related Work

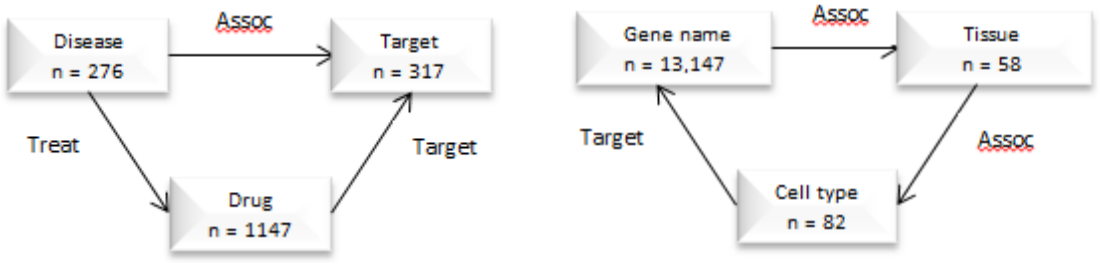
Knowledge graphs are the solution of various natural language processing applications. They have seen a very important growth for the past years. Many models have been designed to manipulate KG's in vector spaces and thus, infer unobserved edges. RESCAL [9] is unremarkably one of the earliest works which models the relation between the nodes. Another significant approach which models relationships in low-dimensional vector space has been proposed By Bordes et al. [1]. More advanced models have been created and followed this area of research, such as TransH [2], DistMult [2] and ConvE.[10].

Neural embeddings is part of our research as well. Neural-embedding models have demonstrated good scalability on large-scale network data. Embeddings Augmented by Random Permutations (EARP) [6], FormulaNet [7] and a Semantic Parser that scales up to Freebase have as main objective to improve natural language question-answer pairs [8].

Last, but not least, statistical relational learning is undoubtedly related to our article. Knowledge graphs have casually plenty of statistical patterns or laws which are predictively powerful. For instance, homophily is a characteristic statistical pattern in which it has got the ability to relate entities with other entities with similar characteristics. A very distinctive example is the actor dataset by which actors who are born within a country are most likely to play in the country's films. Block structure is another real example of statistical learning method. This is referring to that can be distinguished into blocks, in a way that all the entities of the block have similar relationships to the entities of another blocks [16]. For example, if there are two actors who play in the same movie category, they belong to the same block. In addition, if there two movies which are grouped into the same movie category, they belong to the same movie category. Thus, there actors block and movies block most likely to be highly connected. In our research, we seek to make predictions on unobserved logical relationships in a knowledge graphs given a set of know edge relationships where all missing edge relationships can be considered as true [11]. Let $G = (V, E)$ be a knowledge graph where V is the set of u nodes and e denotes the e directed edges.

3. Prerequisites

Lately, the term knowledge graphs has intensively been used even more often as the time glides by in research area and on the market. It is usually related to semantic web applications, large-scale data, and cloud computing and linked data. [11] Knowledge graphs consist of nodes and directed edges which are strongly connected.



a) b)

Figure 1: Schema diagrams for biomedical and normal tissue data.

In this work, we first use a typical biomedical knowledge graph (Figure 1.a) The Therapeutic Target Database (Bio-Data) [12] to evaluate and test our models on knowledge graph completion. TTD is a database designed to provide information about the already known therapeutic protein nucleic acid targets, the corresponding drugs/ligands and the disease name at each of these targets. It currently contains records for 3173 protein nodes covering 274 disease names and 1128 drugs/ligands. The subset of the data set totally contains more than 7 thousands relational edges.

The second dataset is the GrankAI tissue dataset (Figure 1.b) which is available in the Human Protein Atlas version 15 [13]. It is a data set that contains profiles for proteins in human tissues based on immunohistochemistry using tissue micro arrays. The subset of the data set that we use includes gene name, tissue name and cell type. We found out that it currently contains 13.147 gene types and subsequently the same number of gene names covering 58 tissues that can be found in 82 different cell types. In this subset of the normal tissue data set there are more than 200.000 multiple edges in total.

4. Data Preprocessing

4.1 Query Structure

On data preprocessing stage we were aware of extracting the most semantic features of the knowledge graph, but before define our feature space it would be more efficient to determine the query structure that we want to examine. Generally speaking in each dataset exist triplets that describe a relation scheme, each unit of those triplets is a node in our graph and is associated with neighbors under a specific relation, so a triplet can be formed with this type $t=(t_1,t_2,t_3)$. Our query structure must be formed so that we can answer to questions like what kind of node should be t_i (or what's the content) with the restriction that the other two nodes are associated with a fixed rule and one of those nodes is associate with the t_i. Consider the case of a media network where we want to answer to questions of the form 'what's the community C (community-node) where a certain post or set of posts (post-node) has been uploaded from a certain user or a set of users (user nodes). This can be expressed as query with the users-nodes defined as anchor nodes that are associated with a fixed rule, especially in this example upload or not, with the post-nodes where we can define them as variable nodes and the community node as our target node or the nodes that we want to predict in a new given query where probably the target node is missing. Obviously in a real world dataset these entities are not predefined, this task it's up to the researcher and the knowledge that we want to extract, so the definition of each nodes is based on the needs of our application.

Sometimes more technical reasons overcome our needs, consider the case where a set of nodes (e.g community nodes) are missing so our expectations are limited and we are enforced to define the community nodes as target nodes so we can fill our knowledge graph by learning the rest full filled structure. We aim to define nodes of type $\gamma \in \Gamma$, subject to $\Gamma \in \{\text{Anchor}, \text{Variable}, \text{Target}\}$ and the association between them $e_1=(A,V)$, $e_2=(V,T)$ e_1 , $e_2 \in \{$ set of associations $\}$, in this project we don't cope with the case where there are more than one sets of associations, the graphical content looks like a dag query with 2 length path. As we mention above our embedding method only utilizes one kind of edges for each e_1 , e_2 , each type of association forms a different path from the sink (Anchor node) to the target node, so we feed our method with 2-length dag paths, each path can be expressed as a triplet with given two types of edges.

4.2 Feature Extraction

Our first task is to express those conjuctive relations in a vector [16] based form, the key idea is to construct a feature space that captures the information that describes the structure from a global point of view. The embedding generator could be benefit by this technique and achieve low generalization errors to unseen relations that don't participate to the main process and are used for testing. Usually a knowledge graph is composed by nodes whose content is consists of words or a bag of words, this observation obtains two cases that we should be considered about. The first trivial case concerns the situation where each node of type γ is assigned by one word, the second case is more general but the feature extraction approaches are the same for both of them. The general idea is to construct a unique feature for each node of type γ, in both cases we construct a dictionary with all the words of the set of nodes of type γ so each word corresponds to a unique key and it's value (random integer between 0 and number of words) points the non-zero position to the hash list. This technique called one hot binary indicator in the trivial case and multiple binary hot indicator in the second case, we must mention that each key-value pair is unique, in collision cases we just probing linearly the pointer of the random generated hashed value across the list. After we construct the dictionary we map each node's corresponding words (or word) to this embedding $x \in \mathbb{Z}^m$ and we assigned it to the current node (m: length of hash-list) where x is vector of zeros except from the positions that are equal to values of word-keys according to the dictionary. This technique is applied to each set of node type except from the set of target nodes.

5. Model Definition

5.1 Projection Operations

In the field of the knowledge graph applications and moreover in the line of successful work on path relationships encoding, we use the projection operation [17] **P(q)** that takes as input an embedding and produce a new one according to the following type:

$$\mathbf{P}(\mathbf{z_u}, \mathbf{t}) = \mathbf{R_\tau z_u} \; \forall \; \tau \text{ edge type } \mathbf{z_u} : \text{input embedding } \mathbf{R_\tau} \in \mathbb{R}^{\mathbf{dxd}} \quad (1)$$

Where $\mathbf{R_\tau}$ are trainable parameters that we want to estimate, we remind that the projection is applied for each different edge type (association between nodes). The semantics of this operation can be expressed as the union of all neighbor embeddings, that corresponds to a certain query structure. In other words nodes that their intersection of their neighborhood list is empty don't belong to the same neighborhood,

because their corresponding embeddings are separated or their distance is very big. Another advantage is that projects the initial huge feature sparse space to a smallest dense one. One of the disadvantages of this operation is that it doesn't leverage the intersection of the incoming information that is concentrated to variable or target nodes (nodes with in-degree>2).

5.2 Intersection Operations

As we described above projection operations works as a discriminative functions with respect to node's neighborhood, but in some cases this is not enough. More complex scenarios, where the query path contains nodes with in-degree greater than one, make our task more hard and set the need of encoding the incoming information that is encapsulated. The main idea is to generate new embedding using as input a set of embeddings, in our case projections operations feed the intersections cause of the priority that the methods follows. First projections generate for each anchor node an embedding vector and afterwards the intersection operations is fed by the sets that corresponds to intersections, in other words we sample the anchor embeddings whose neighbor has in-degree greater than one, this sets are used as the input to the intersection operation [18,19], that has the following form:

$$\mathbf{I}(\{\mathbf{q_1 q_2}, \dots, \mathbf{q_n}\}) = \mathbf{W_\gamma \Psi}\left(\mathbf{NN_k}(\mathbf{q_i})\right) \forall \boldsymbol{i} = \mathbf{1} \dots \mathbf{n} \; \forall \; \gamma \text{ node, type }, \mathbf{W_\gamma} \in \mathbb{R}^{\mathbf{dx}\boldsymbol{d'}} \quad (2)$$

$\mathbf{\Psi} \in \mathbb{R}^{\mathbf{d'}}$ elementwise mean over the set, d′: output shape of the NN with k layers and relu activation.

So consider $\{q_1 q_2 \dots . q_k\}$ embeddings that correspond to one variable node can be expressed as one new embedding q' where the parameters $\mathbf{W_\gamma}$ and $\mathbf{NN_k}$ can be estimated.

This new operation allow us to encode with higher efficiency all kind of query structures and represent them to a lower dense feature space. As it's described below all those parameters can be estimated with respect a classification task called Max-Margin that is used more often in the NLP tasks. The core semantics of the whole structure will be assigned to the target nodes of our training graph then we will be able to find top K nearest embeddings from a new query after adapt it to the new space using the pre-trained operations.

6. Model Training

Now it's time to concatenate all the previous details and to formulate the core process, the training stage is splited into two core axes. The first one concerns the case of estimating the right parameters of the projection operation and the second one the intersection respectivaly. Following this pipeline there will be two training phases with respect one loss function called max margin loss.

$$\mathcal{L}(\mathbf{q}) = \mathbf{max}(\mathbf{0}, \mathbf{1} - \mathbf{score}(\mathbf{q}, \mathbf{z_u},) + \mathbf{score}(\mathbf{q}, \mathbf{z_{U_N}})$$

where $\mathbf{z_u} \epsilon [[\mathbf{q_{train}}]]$, $\mathbf{z_{U_N}} \notin [[\mathbf{q}]]_{\mathbf{train}}$ initial embeddings

We tried to minimize this loss for each training example, generally the objective task of this minimization problem is to separate negative and positives examples with respect that notation:

$$\mathbf{score}(\mathbf{q}, \mathbf{z_u}) = \frac{\mathbf{q} * \mathbf{z_u}}{\|\mathbf{q}\|\|\mathbf{z_u}\|} = \begin{cases} \mathbf{a} > \mathbf{0} \ \textbf{if u} \ \boldsymbol{positive} \\ \mathbf{a} < \mathbf{0}, \ \textbf{else u negative} \end{cases} (3)$$

In our experiments $\mathbf{z_u}$ is the initial embedding of each anchor node and it's neighbors, thus are concatenated vectors for each edge e.g if one anchor node has out degree greater than one then immediately we form two $\mathbf{z_u}$ initial embeddings with dimensionality equal to $\mathbf{m_\gamma}$ (summary of the lengths of two vocabularies from anchor and variable sets). We have to mention that one node-embedding belong to the set of positives examples if the path that passed this node is negative or positive. We sample query structures with different types of paths starting from the target node and backtracking to the anchors (described in Section 6). Thus in the first training scheme $\mathbf{q}$ takes the roll of the projection operation $\mathbf{q=P(z_u,t)}$ (1). In relational learning procedures the key objective is to minimize the energy between similar examples and to maximize the margin between different examples. So at each step we sample randomly a negative $\mathbf{z_{U_N}}$ and a positive $\mathbf{z_u}$ example and we update the gradients of our parameters according to the following procedure:

- If $\mathbf{score(z_u) > score(\ z_{U_N}) - 1}$ apply gradient step to minimize margin loss.
- Else continue to the next step.
- Normalize the embedding vector to norm 1.

As in the classical optimization methods we can adjust some parameters such as batch size $= 64$ and learning rate $=0.01$. The evaluation on the margin requires only two examples one for each set (negative or positive), to cope with case of overfitting we take advantage of the cross validation method by splitting the data set to 10% for validation and 10% for testing. The common question is how can we guarantee that this discriminative function works , we just evaluate the estimated parameters to the testing set and we check if :

$$\sum_{\mathbf{z_{U_N}}} \mathbf{score}(\ \mathbf{z_{U_N}}) < \mathbf{0} \ \text{and} \ \sum_{z_u} \mathbf{score}(\ \mathbf{z_u}) > \mathbf{0}\ .$$

This kind of process can help to rank each triplet among the others and to encode the structural behavior of our graph base.

The philosophy behind intersection training approach is reasonably the same as operation's training, thus the only difference between them is that only variable nodes with in-degree greater than one can participate in the training. The architecture of the structural neural network that we used is formed by one or at most two hidden layers with relu as activation function. The $\mathbf{\Psi}$ operator from (2) equation is a elementwise mean function over a set of examples evaluated by the $\mathbf{NN_k} \in \mathbb{R}^{d' x\, n}$, $\mathbf{\Psi} \in \mathbb{R}^{\boldsymbol{d'x1}}$, d' : dimension of the output layer and n:#number of examples and , $\mathbf{W_\gamma} \in \mathbb{R}^{\mathbf{dxd'}}$ is an additional projection operator. The sets of negative and positive examples are immutable that means that we don't sample so we avoid redundant overhead to the method. Nodes and their corresponding embedding vectors that has been assigned as negative or positive don't change. We used Adam's optimizer with training rate 0.01 and batch size 64 for both training procedures.

Inference using P and I operations

So far we have been consider of how to train our operations, but until now we haven't worried about how to inference on the structure and how we get our final results. On the first stage we compute the initial embedding vectors for each node of each type except from target nodes, afterwards for each relational triplet (structural path in our graph) we concatenate the computed initial vectors, so we can construct the input samples for the projection P. So for each edge between anchor nodes and variable nodes we have generated the projected embedding with dimension **d.** On the next step we sample variable nodes and their incoming edges assigned with the embedding if one variable node has in degree greater than one we put him in a list along with his edge vectors, then we evaluate the intersection operator and we have accomplish to encode the incoming information and produce a dense vector with dimensionality equal to **d**, then we apply again the projection operator to the variables so we can take advantage of the discriminative attribute of the operation. Until now each node has its own current vector embedding, iteratively we continue to evaluate projections and operations with the same manner (length of query path times) until we reach the target nodes and finally generate an embedding for each target node. Cause of the notation that our training structure has full filled queries we can construct a hash map list with key the embedding and value the target node. The same procedure we follow with the query that we want to answer with missing target node. Now we are capable to make a decision for the answer of the new query using an nearest neighbor approach called LSH and we pick the nearest vector of the trained knowledge graph as the answer to the query. It's important to mention that complexity of the inference method is sublinear to the **|V| V**: set of nodes in the graph.

7. Sampling Pipeline

In this section we declare the implementation scheme behind the sampling approach of positive and negative examples. We remind that each example is an edge assigned by the embedding vectors of its end; we define an edge as positive or negative if the end that corresponds to an anchor node has been assigned as positive or negative. Thus our task is to pick with a stochastic manner anchor nodes as positive or negative so we be able to feed our model. The structure of the positive query that we want to sample depends on the queries that we want to examine in each case, e.g. queries paths that contains variable nodes with in-degree greater than one are candidate for participating in positive edges. The key idea is to put weights on each node according to their in degree, so we sample over a distribution on nodes on the traversing graph. The pipeline follows the next procedure:

- Compute the probability of picking one target node by dividing each node in degree by the summary of all in degrees of the set of target nodes.
- Then we pick the target node over the created degree distribution.
- Backtracking on the traversal graph pick neighbor variable nodes of the target node after computing their degree distribution and reject nodes with in degree less than our constraint.
- Finally after picking the variable node we are capable to pick the set of anchor nodes as positive nodes.

Setting the constraint (query type 2 constraint=2) depends on what kind of query types we want to examine, so we need to pick variable nodes with in degree greater than our constraint. Query types have the following form:

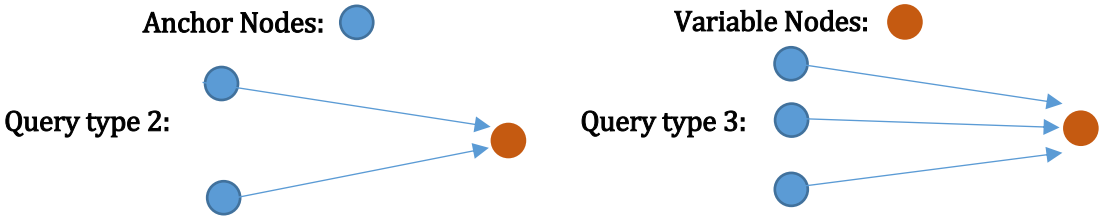


Figure 3

The whole process converges when we have filled a balanced percentage of examples from both of two classes, notice that the above procedure samples positive examples the rest of them are included to the negative class. The idea behind this kind of sampling is to collect with an efficient a fair manner positive and negative examples using stochasticity. The collected training set has been separated for testing and validation 0.1% for testing and 0.1% for validation from both negative and positive classes (0.05% from each one).

8. Results

In our experiments we worked on two real datasets GrankAI and Bio-data, both of them contain two length path conjunctive queries about bioinformatics concepts. As we mentioned earlier GrankAI knowledge graph consists of triplets of the form (gene name, tissue, cell type) and Bio-data (disease, proteins, drugs) respectively, and each of these triplets are associated with one kind of edge, we denote that this kind of association is unique in our graph. Due to the fulfilled path-queries in our graphs we were capable to experiment with situations where each node can participate with any role in the method (anchor, variable, target). So we separated our experiments in two cases, the first one sets as target nodes the elements with the maximum population and the second case the minimum respectively, for example in Bio-data disease element set is composed by 274 nodes and proteins with 3.173 nodes with 7.234 formed relational triplets and GrankAI's maximum population is the gene name element set with 13.147 units with 200.000 relations. Each case consists of two subcases where we examine the results of different query types (Figure 3). For each graph we split the relations (for each query type) into train and test, the testing queries was 20 for Bio-data and 50 for GrankAI set and don't take place in the training procedure. After the training stage we inference using LSH and NN (nearest neighbor), we apply projection and intersection operations on the new coming query and we generate the corresponding embedding vector. Our expectations is to track the true label on the closest 10 embedding vectors to our query target node. The metric that has been used measures the percentage of the queries that have been answered correctly according to their true label, we aim to find the top 10 closest embedding vectors to the query if their labels match to the true label this query is assigned as correct answered. In Table 1 we demonstrate the results for each case and query type that we focus on and we make some observations on them.

Table 1

Case 1	Bio-Data	GrankAI
Query Type 2:	40%	36%
Query Type 3:	30%	34%
Case 2	**Bio-Data**	**GrankAI**
Query Type 2:	60%	53%
Query Type 3:	50%	38%

These results drive us to obtain some interesting observations, the first one is that the method reasonable fails to find the closest true answer to the queries in case of the set of target nodes is the bigger set in our graph and the second observation is that can't handle more complex query structures query type 3 and beyond. The whole process for Bio-data after an intensive hyper parameter tuning finished after 3.000 number of epochs with batch size 64 and total average train loss 0.312 and validation loss 0.533 for Projection training the and 0.0 with 1.234 for intersection, for GrankAI set finished after 50.000 epochs and same batch size with almost same results as Bio-data training procedure.

9. CONCLUSIONS

In this work, the main purpose is to test and implement the frame worked that has been described and built by William et al. [17] thoroughly explaining the sampling, testing and training techniques. Our results have shown that the approach can accurately predict on real-world data and especially on biomedical ones including both, small and big amount of data. The research that we carried out showed that the approach works much more efficient in query type number 2 than in query type number 3. On the first bio data set evaluation has showed that it gives better results and thus correct answers compared to the evaluation on Grank AI data set and precisely it is 1.2 times better on query type 2 in the first case by which the population of the target nodes in the maximum one. However, exactly the opposite occurs for query type 3 in which Grank AI data set is more than 1.1 times better answered correctly than bio data set queries. In second case in which the population of the target nodes is the minimum, evaluation on bio data set is much more efficient and gives correct results in both query types. Accurately, bio data gives 1.1 times more than Grank AI data set correct answers on query type 2 and 1.3 times on query type 3.

10. REFERENCES

[1] Bordes An., Usunier N., Garcia-Duran Al. 2013 "Translating Embeddings for Modeling Multi-relational Data", *In proceedings of NIPS'13 International Conference on Neural Information processing*, vol. 2, pp. 2787-2795, Lake Tahoe, Nevada, USA

[2] Wang Z., Zhang J., Feng J., Ghen Z. 2014 "Knowledge Graph Embedding by Translating on Hyperplanes", *In proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI, Quebec City, Quebec, Canada

[3] Guu K., Miller J., Liang P. 2015 “Traversing Knowledge Graphs in Vector Space”, *In proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*, Lisbon, Portugal, DOI: 10.18653/v1/D15-1038

[4] Nickel M., Tresp V., Kriegel H., 2011 “A Three-Way Model for Collective Learning on Multi-Relational Data”, *In proceedings of the 28th International Conference on International Conference on Machine Learning*, pp. 809-816, Bellevue, Washington, USA

[5] Bouchard G., Naradowsky J., Riedel S., Rocktaschel T., Vlachos A., 2015 “Matrix and Tensor Factorization Methods for Natural Language Processing”, *In proceedings of Tutorials Conference*, DOI: 10.3115/v1/P15-5005

[6] Cohen T., Widdows D., 2018 “Bringing Order to Neural Word Embeddings with Embeddings Augmented by Random Permutations (EARP)”, *In Proceedings of the 22nd Conference on Computational Natural Language Learning*, pp. 465-475, ACL, Brussels, Belgium

[7] Wang M., Tang Y., Wang J., Deng J. 2017 “Premise Selection for Theorem Proving by Deep Graph Embedding”, *In proceedings of the Annual Conference on Neural Information Processing Systems*, pp. 2783-2793 Long Beach, CA, USA

[8] Berant J., Chou A., Frostig R., Liang P. 2013 “Semantic Parsing on Freebase from Question-Answer Pairs”, *In proceedings of the Conference on Empirical Methods on Natural Language Processing(EMNLP)*, pp. 1533-1544, ACL, Seattle, Washington, USA

[9] Yang B., Yih W., He X., Gao J., Deng L. 2016 “Embedding entities and relations for learning and inference in knowledge bases”, *In proceeding of the International Conference on Learning Representations (ICLR)*, San Diego, CA, USA

[10] Dettmers T., Minervini P., Stenetorp P., Riedel S. 2018 “Convolutional 2d knowledge graph embeddings”, *In proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)*, New Orleans, LA, USA

[11] Ehrlinger L., Wöß W. 2016 “Towards a Definition of Knowledge Graphs”, *In Joint proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems*, vol. 1695 Aachen, Germany

X. Chen and

[12] Chen X., Ji ZL, Chen YZ 2002 “TTD: Therapeutic Target Database”, PMCID: PMC99057, DOI: 10.1093/nar/30.1.412

[13] Knut & Alice Wallenberg, “The Human Protein Atlas project” https://v15.proteinatlas.org

[14] Rosset S., Zhu J., Hastie T. 2003 “Margin Maximizing Loss Functions”, *In proceedings of the NIPS ’03 16th International Conference on neural Information Processing Systems*, pp. 1237-1244, Whistler, British Columbia, Canada

[15] Nickel M., Murphy K., Tresp V., Gabrilovich E. 2015 “A Review of Relational Machine Learning for Knowledge Graphs”, *In proceedings of the IEEE*, DOI: 10.1109/JPROC.2015.2483592

[16] L. Wu, A Fisch S. Chopra K. Adams A. Bordes and J Weston Starspace Embed all the Things IAAAI 2017

[17] William L. Hamilton L. W., Bajaj P., Zitnik M., Jurafsky D.†, Leskovec J. 2018 "Embedding Logical Queries on Knowledge Graphs", *In proceedings of the NIPS '18 32nd International Conference on Neural Information Processing Systems*, pp. 2030-2041, Montreal, Canada.

[18] C. Qi, H, Su, K Mom and L. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In CVPR 2017

[19] T. Rocktaschel, and S. Siebel Low-dimensional embeddings of logicm. In ACL Semantic Parsing pages 45-49, 2014