Application of Graph Neural Networks to Semantic Search

Ahmet Korkmaz[1], *Amos* Dinh[1], *Henrik* Rathai[1], and *Matthias* Fast[1]

[1] *Cooperative State University Baden-Wuerttemberg Mannheim*

Abstract.
We present a novel approach for semantic search, utilizing Graph Neural Networks (GNNs). The proposed framework first constructs a graph linking papers, words, and authors, where edges are weighted by TF-IDF and word co-occurrence statistics. The Heterogeneous Graph Transformer is employed in combination with the Knowledge Graph Embedding (KGE) method TransE to find informative representations for entities in the graph. Applied in unision with a vector database, the approach enables content search beyond simple keywords. The representations are learned in a joint metric space, enabling users to find related papers, explore topics through keywords, or discover authors with similar research focus, enhancing the quality of academic literature exploration. The feasiblity of the training and application of the proposed method on large corpora is demonstrated on the basis of the arXiv dataset.

1 Introduction

Most commonly, semantic search on text corpora is enabled on the basis of word- or sentence-embeddings, learned with methods such as Word2Vec [1], GloVe [2] or Sentence-BERT [3]. The present paper builds on the intuition introduced by these methods that co-occurence presents a notion of similarity. The proposed method combines co-occurence statistics between words, as well as occurence statistics of words in documents, term frequency-inverse document frequency (TF-IDF). Further, this information is gathered and represented together with entity attributes through a graph structure.

In detail, the graph is constructed using the arXiv paper dataset [4], whereby different entities, such as authors, papers, journals and words are linked through different relationships, such as "paper-written_by-author". The Heterogeneous Graph Transformer can then combine information of entity relationships, the occurrence- and co-occurrence statistics to find representations for entities in the context of the arXiv dataset. The method is closely related to [5], which employ the combination of text-based statistics and graph structure to text-classification.

2 Related Work

[5] construct a graph containing documents and the words which occur in the documents as nodes. Documents are linked to words by the respective TF-IDF weights, and words share an edge weighted by their point-wise mutual information statistic (PMI). For the task of document classification, the authors demonstrate that a Graph Convolutional Network [6] trained on this graph outperforms a logistic regression model trained on TF-IDF-weighted bags-of-words.

[7] introduce the Heterogeneous Graph Transformer (HGT). Similar to [8] the HGT learns different weight matrices W per entity type for the neighborhood aggregation step. The intuition is that different entity types, such as authors and papers posses different feature distributions and thereby, distinct weight matrices W allow a richer information extraction. The HGT further employes the attention-mechanism [9] for the source-relationship-target triple, representing an edge between nodes. For computing a target nodes' representation, the HGT learns the attention matrices K, W and Q for source node, relationship and target node. The matrices are learned for each relationship type and node type independently. Attention allows the model to determine the neighbors' importance relative to the target node. To handle web-scale data, the authors introduce a novel sampling method. A subgraph around a minibatch of target nodes is sampled. The sampling of neighbors is probabilistic. It depends on the connection count of neighbors towards the target nodes, normalized by the neighbors' absolute degree count. This effectively allows to sample more dense subgraphs, such that each target node in the minibatch is represented by more neighbors than if neighbors were sampled uniformly at random.

[10] introduce the KGE method TransE. On a knowledge graph, embeddings of nodes are learned. Specifically, the distance $d(h_s + h_r, h_t)$ over all realizations (s, r, t) is minimized using a lookup-table-based approach, where h_s is the source node's embedding, h_r a relationship embedding with type $\phi(r)$, h_t the target node's embedding and d is the Minkowski distance. In contrast to methods such as [11], TransE embeds all entity types in the same metric space, thereby allowing to compare all embeddings with each other. Suppose two relationship types in a graph, $\phi(e_1) = R_1$ and $\phi(e_2) = R_2$ exist, where e_1 connects node

s to t, $e_1 = (s,t)$ and e_2 connects node t to u, $e_2 = (t,u)$. Then the embedding of node s can be compared with the embedding of node u by the distance $d(h_s + h_{R_1} + h_{R_2}, h_u)$. Although no explicit relationship type R_3 between node types $\psi(s)$ and $\psi(u)$ exist, TransE still enables their comparison.

3 Methodology

This section describes the proposed methodology, distributing information obtained from text-based metrics across links to other entities related to the main body of documents.

3.1 Graph Modeling

The basis of the method is represented by modeling documents and corresponding relational data in a graph structure. Taking the arXiv dataset [4] as an example, most information is contained in the provided abstracts' texts of each research paper. This data is augmented by the information about authorship, categories, and journals. The information can be expanded as entities in the graph, linking concepts together. Graph Neural Networks then allow us to learn representations, taking into consideration the knowledge about all entity types. In contrast to methods like TF-IDF, authors can not only be represented by the average of their documents embeddings, but the Graph Neural Networks may consider the representations of co-authors or the journals in which papers are published as well.

3.2 Assign edge weights

In this paper the term frequency-inverse document frequency (TF-IDF) metric is used for document-word relations and point-wise mutual information (PMI) for word-word co occurrencies as in [5].

TF is the count of a word in a document. IDF measures the rareness of a word across all documents and is calculated as follows:

$$\mathrm{IDF}(t) = \ln\left(\frac{1+n}{1+df_k(docs)}\right)$$

where n is the number of documents in the corpus and df_k the number of documents including term k. The TF-IDF score is the product of TF and IDF. [12, p. 2][13]

PMI with a fixed-size sliding window can be used to analyze word relationships across a corpus. In this paper the normalized PMI metric NPMI is being used. The weight of the connection between two word nodes i and j is thus defined using the following approach:

$$\mathrm{NPMI}(x,y) = \ln\left(\frac{p(x,y)}{p(x)p(y)}\right) / -\ln p(x,y)$$

One side effect is the potential reduction of low frequency bias. [14, p. 5] The NPMI value between two words of the corpus is based on their co-occurence. A value of 1 indicates that the words always appear together, a value of 0 signifies that they occur independently, and a value of -1 means that they never appear together.[14, p. 6]

3.3 Training

Training uses a two layer deep HGT [7]. Each layer is of size 256 and includes 8 attention-heads. To allow nodes to attend to themselves, the input graph is augmented by self-edges. First, the node attributes are fed through a linear layer to downsize to 256 dimensions, then through two HGT-layers and a preceding linear layer with the same dimensionality. Finally, the node representations are learned with an additional TransE [10] head with Minkowski distance $p = 2$ in combination with pairwise margin loss [15] as proposed in [10], with $\gamma = 0.5$:

$$\mathcal{L} = \sum_{(s,r,t)\in S} \sum_{(s',r,t')\in S'} [\gamma + d(h_s + h_r, h_t) - d(h_{s'} + h_r, h_{t'})]_+$$

$$[a]_+ = \begin{cases} a, & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

In detail, existing edges and corresponding node pairs are sampled to create the set of positive examples S and the source nodes are connected to random target nodes for the negative examples $\in S'$.

4 Experiment Setup

In this section, the application of the proposed method to the use case of semantic document-search is presented.

4.1 Dataset

The arXiv dataset [4] is a continually updatet repository containing 1.7 million scholarly articles in various fields such as computer science, physics, quantitive biology and economics. After extracting timestamp and page count from the dataset we are left with a refined set of features including license, doi, pages, jornal, date, id, title, author, category, journal and abstracts. The data is being preprocessed by lemmatization and the deletion of stopwords. The spacy lemmatizer assigns the base forms to each token in a corpus. [16] Additionally, stopwords are removed using the NLTK package[17].

4.2 Implementation Details

The dataset is represented using a heterogenous graph-object from the PyTorch Geometric library [18] consisting of five node types: paper, author, category, journal, and word. The paper node type has attributes like license, DOI, pages, journal, date, id, and title and forms edges with author, category, word, titleword and journal nodes. In order to create the nodes OrderedSet [19] was used. This way the representation of every word without duplicates within a list is possible while keeping the order of the entries. The nodes are assigned numerical representations based on created dictionaries. After creating nodes using an OrderedSet to ensure a unique and ordered representation of each word, edges between the nodes are generated through the iterative mapping of the lists. Edges between two word nodes are established if their PMI score is greater than 0, signifying a positive correlation. Additionally, the edge weights connecting paper nodes to word nodes are determined by their TF-IDF scores, reflecting the importance of words in the papers. The NPMI calculation in this study uses a window size of 10, as opposed to the window size of 20 used by Han et al. in [20], reflecting the smaller word count in abstracts opposed to entire texts. Because of practicality, for the word-word edges, only approximately the 50 edges with the highest PMI score are used. Similarly, each paper is only represented by the 50 words with the highest TF-IDF score. To provide initial node features for the GNN to learn on, Sentence-BERT [3] embeddings encode the identifying text attribute of each node type. For authors, papers, journals, categories and words, it is their respective name.

Data pre-processing is conducted on an 124GB 22-core machine for 5 hours, although code-optimizations would allow the pre-processing with arbitrarily small resources. The model is trained on a single Nvidia P100 GPU. The GNN learns over 36 hours from 900000 target edges with the additional amount of negative examples. The Adam optimizer [21] with a learning rate of 2e-4 is used. Sampling is conducted with minibatch size 32, amounting to two separate sampling operations by the heterogeneous graph sampling of 32 target nodes respectively, in case the target edge type contains two different entity types. Otherwise a subgraph of 64 nodes is sampled. Compared to simple random neighborhood sampling , the employed graph sampling approach limits training iteration speed, although obtained subgraphs might hold more information. The inference of the final model amounts to 12 hours, to produce approximately 7 million entity embeddings. Ideally, inference should be conducted multiple times to average the obtained embeddings, as representations vary based on the sampled subgraph. Pre-processing, training and deployment code can be found in the repository [22].

4.3 Results

Query results from pure TF-IDF embeddings extracted from the papers' abstracts, and reduced to 256 dimensions with principal component analysis are compared against the proposed methods' results. For a query paper title, relevant query results are annotated. The scoring is a simple average across the ranking of query results which should be placed at the top for both methods respectively. TF-IDF achieves an average rank of 189037 out of 2381173 whereas the proposed approach achieves an average ranking of 81752 out of 2381173 (lower is better).

5 Implementation

The vectors that the GNN is learning are stored in a Weaviate [23] vector database. Along with the vectors, the original attributes of the data are also stored. For instance, for papers also their titles are saved. The frontend application is realized with the Python library Streamlit. The user can make an entry via a text input field. Below this is a drop-down menu where the user can select which class to search in. The user is then shown the 10 best hits for their search. The user can then select one of these and similar entries for the selected class are returned. The similarity search searches similar vectors in the database using the Euclidean distance and returns the best hits.

Some difficulties are encountered during implementation, of which the lack of working memory proves to be particularly challenging. A technique called Product Quantization (PQ) is being used to reduce the required memory space. This method is being utilized for the compression of vectors. It is exhibiting high efficiency when being applied to the compression of high dimensional vectors, specifically for the purpose of nearest neighbor search [24].

PQ is a method used to speed up the process of finding the nearest neighbors in high-dimensional spaces. PQ works by dividing the original high-dimensional vector space into a Cartesian product of low-dimensional subspaces. Each subspace is then quantized separately, leading to a set of quantization indices for each vector. This approach results in a compact code for each vector, which is used for storage and comparison purposes [24].

The advantages of product quantization are twofold: it reduces the memory usage significantly, as each vector is represented by a short code rather than a high-dimensional vector; and it speeds up the nearest neighbor search, as the distance between two vectors can be computed efficiently using the precomputed lookup tables.

6 Conclusion

This paper presents the idea of the application of Graph Neural Networks in combination with text-based statistics. The Graph Neural Network enables the approach to utilize the text-based data in order to inform the representations of entities linked to the main body of documents. Thereby, a holistic view on the relations naturally found in the data becomes possible, empowering advanced semantic search capabilities. By example the setup of semantic search is demonstrated on the large corpus of the arXiv dataset. While the results show feasibility of the proposed approach, further work is needed to validate and consequently improve the method. Different GNN models and training approaches could be experimented with. The margin loss could be replaced with unsupervised loss methods like attribute prediction or explicit link prediction.

Future work could include exploring various Graph Neural Network (GNN) models and training methodologies, not only focusing on ranking loss but also incorporating link prediction.

References

[1] T. Mikolov, K. Chen, G. Corrado, J. Dean, *Efficient estimation of word representations in vector space*,

arXiv preprint arXiv:1301.3781 (2013)

[2] J. Pennington, R. Socher, C.D. Manning, *Glove: Global vectors for word representation*, in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (2014), pp. 1532–1543

[3] N. Reimers, I. Gurevych, *Sentence-bert: Sentence embeddings using siamese bert-networks*, arXiv preprint arXiv:1908.10084 (2019)

[4] C. University, *Arxiv dataset*, https://www.kaggle.com/datasets/Cornell-University/arxiv (2023), rEPLACE Accessed: January 25, 2024

[5] L. Yao, C. Mao, Y. Luo, *Graph convolutional networks for text classification*, in *Proceedings of the AAAI conference on artificial intelligence* (2019), Vol. 33, pp. 7370–7377

[6] T.N. Kipf, M. Welling, *Semi-supervised classification with graph convolutional networks*, arXiv preprint arXiv:1609.02907 (2016)

[7] Z. Hu, Y. Dong, K. Wang, Y. Sun, *Heterogeneous graph transformer*, in *Proceedings of the web conference 2020* (2020), pp. 2704–2710

[8] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, M. Welling, *Modeling relational data with graph convolutional networks*, in *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15* (Springer, 2018), pp. 593–607

[9] D. Bahdanau, K. Cho, Y. Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473 (2014)

[10] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, *Translating embeddings for modeling multi-relational data*, Advances in neural information processing systems **26** (2013)

[11] B. Yang, W.t. Yih, X. He, J. Gao, L. Deng, *Embedding entities and relations for learning and inference in knowledge bases*, arXiv preprint arXiv:1412.6575 (2014)

[12] J. Ramos et al., *Using tf-idf to determine word relevance in document queries*, in *Proceedings of the first instructional conference on machine learning* (New Jersey, USA, 2003), Vol. 242, pp. 29–48

[13] *sklearn tfidfvectorizer* (2024), accessed: 2024-01-21, `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`

[14] G. Bouma, *Normalized (pointwise) mutual information in collocation extraction*, Proceedings of GSCL **30**, 31 (2009)

[15] *pytorch, margin ranking loss* (2023), 22.01.2024, `https://pytorch.org/docs/stable/generated/torch.nn.MarginRankingLoss.html`

[16] *spacy lemmatizer documentation* (2024), accessed: 2024-01-20, `https://spacy.io/api/lemmatizer`

[17] S. Bird, E. Loper, E. Klein, *Natural Language Processing with Python* (O'Reilly Media Inc., 2009)

[18] *Heterodata* (2024), accessed: 2024-01-19, `https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.data.HeteroData.html`

[19] *Ordered set* (2024), accessed: 2024-01-19, `https://pypi.org/project/ordered-set/`

[20] S.C. Han, Z. Yuan, K. Wang, S. Long, J. Poon, *sklearn tfidfvectorizer* (2022), `2203.16060`

[21] D.P. Kingma, J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014)

[22] H.R. Ahmet Korkmaz, Amos Dinh, M. Fast, *GNNPapersSearch: Graph neural networks for semantic search*, `https://github.com/AmosDinh/GNNpapersearch` (2024)

[23] *Weaviate* (2018), 22.01.2024, `https://weaviate.io/`

[24] H. Jegou, M. Douze, C. Schmid, *Product quantization for nearest neighbor search*, IEEE transactions on pattern analysis and machine intelligence **33**, 117 (2010)

Arbeitsteilung im Projekt GNNPaperSearch

1. Preprocessing and Graph Modeling: Ahmet Korkmaz und Henrik Rathai
2. Training: Amos Dinh
3. Applikation und Evaluation: Matthias Fast