Recommendations using Knowledge Graph Embeddings

Simon Coessens Gabriel Lozano

October 21, 2024

Abstract

Around 80% of Netflix stream time comes from recommendations[7]. Recommendations play a key factor in today society and recent interest in graph embeddings have surge as an option to not only recommend content, but also clustering, classification, link prediction and machine learning applications. In this paper we review what Graph Embeddings are and how they are being used for recommendations.

1. Introduction

The idea behind Knowledge Graphs(KG) has changed through the years from the concept of semantic net in 1956[15], to the recent rise in popularity explained by D. Fensel [6] as: "Google coined the term Knowledge Graph in 2012 to build a model of the world. Meanwhile, it has become a hype term in the product and service industry". Google's famous implementation[17] and the ones we see today aim to solve the problem of **integrating data** in a big data context.

The main importance of KG is to[9] integrate, manage and extract value from diverse sources of data at large scale. Nowadays we have gone one extra step and used KG for Language Representation Learning as a component of NLP models, like K-BERT[11]. We can also see them being used in **recommendation models** which is the focus of this paper.

KG come in two main flavors: open and proprietary. Open KG include dbpedia[1], YAGO[2], and wikidata[3] which are open databases for web semantics mainly based on Wikipedia. The main idea behind web semantics[2] is to allow computers to understand and navigate the knowledge that has been uploaded to the internet. Closed KG correspond to companies creating datasets to supply their needs; some examples are Google[4] or Yahoo[19], that use KG to solve internal queries.

[1] https://www.dbpedia.org/
[2] https://yago-knowledge.org/
[3] https://www.wikidata.org/wiki/Wikidata:Main_Page
[4] https://developers.google.com/knowledge-graph/

We can think of KG as a finite set of triples subject-predicate-object. We can build some rules on what these triples mean by following a set of rules like RDFS[5] or OWL[6]. This allow us to express complex semantics like "Male" is a "type" of "Person".

A very important tool for KG is embeddings, that we explain in Section 2. We explain the specificities of KG embeddings (KGE) in Section 3. We explain the current state of KGE for the industry in Section 4 and give some practical examples in Section 4. Finally we delve in the different options for recommendations using graph embeddings in Section 5.

2. Embeddings

Computers have a hard time understanding words. For humans is intuitive that "queen" and "king" are similar while "king" and "kick" are different. For software there is no inherent meaning in words, there is no closeness concept. To solve this problem the NLP community decided to think of words as points in a high dimension. Meaning a word w can be represented as $w \in \mathbb{R}^d$ where d is an integer denoting the number of dimensions we use to represent a point. In this space making sense of words meaning was easier, we could apply the usual classification or regression models to text, since it was now a point in space. The problem was thus shifted, from understanding a set of characters, to finding a "good" way of embedding words.

2.1. Embeddings in Natural Language Processing

A very successful aproach to embeddings was proposed by Mikolov[13] known as **word2vec**, a shallow neural network model aimed to predict the context of a word or predict the word given its context. Here is important to understand that given a corpus (a set of phrases), we have a sequence. Words have a sense of closeness. The word "queen" would normally appear in contexts where the word "king" is used, and the embeddings like word2vec were able to identify this closeness.

Embeddings opened the possibility for many applications in NLP. Famous application include:

- Sentiment analysis: Given a review in natural text, is it a good review or a bad review?.
- Language translation: translate from spanish to english.
- chatbots: automated software that can chat with humans.

KGE have also resulted in a great variety of use cases that come from exploiting semantics. But our first challenge would be to find a way to serialize a graph. Since words are already serialized (one word comes after the other), there was no need to solve this problem, for KG it is a problem and there are multiple alternatives.

[5] https://www.w3.org/TR/rdf-schema/
[6] https://www.w3.org/OWL/

3. Graph Embeddings Theory

Just as we can generate embeddings for words in natural language, we can also generate embeddings for nodes in a graph. These embeddings encapsulate the characteristics of a specific node, including its neighborhood, connected nodes, and relationships.

This process is undertaken with the goal of efficiently providing this information to machine learning and computer algorithms, as these systems are specifically designed to work with vector representations of data.

Throughout this report, we will use the following notation: a graph is represented as $G = (V, E)$, where each vertex $v \in V$ corresponds to an entity in the knowledge graph, and each edge in E represents a relationship between entities. In the context of knowledge graphs, these relationships are typically represented as RDF triplets, structured as $(\text{subject}, \text{predicate}, \text{object})$.

For example, in a knowledge graph of scientific papers, vertices represent papers, authors, and institutions, while edges represent relationships such as *authored_by* and *cited_by*. A graph embedding for this knowledge graph maps each vertex to a vector in a high-dimensional space in such a way that semantic similarities and dissimilarities in the graph are preserved according to the RDF triplets.

Example: Consider a simple RDF graph with the following triples:

- $(\text{Paper1}, \text{authored\_by}, \text{AuthorA})$
- $(\text{AuthorA}, \text{works\_for}, \text{InstitutionX})$
- $(\text{Paper1}, \text{cited\_by}, \text{Paper2})$
- $(\text{Paper2}, \text{authored\_by}, \text{AuthorB})$

This example will be utilized to demonstrate how different graph embedding algorithms can capture and utilize such relationships to infer new knowledge, such as predicting future citations or discovering potential collaborations.

3.1. Formal definition

A graph embedding is a function $f : V \to \mathbb{R}^d$ that maps each vertex v of the graph $G = (V, E)$ to a point $f(v)$ in a d-dimensional vector space. The goal of this embedding is to preserve certain graph properties, such as distances or connectivity, in the embedding space.

3.2. Types of embeddings

In this section, we take a look into various embedding methods that were highlighted in our literature review. Each method offers unique advantages and is suited for different types of data representations.

Task	Pros	Cons	Recent Focus
Bipartite Graph Embedding	Well-suited for direct user-item interactions	Optimization can be challenging due to non-convex nature	Non-negative matrix factorization, Metric learning
Markov Processes	Captures sequential user behavior	Shallow learning models may miss deeper patterns	Automating feature generation
Deep Learning	Can discover non-linear patterns	Requires extensive computational resources	Integrating with matrix factorization
Translation	Preserves local topological features	May miss global topological features	Enhancing flexibility in handling multiple node and relation types
Meta Path (Random Walk)	Preserves global topological features	Requires expert knowledge for effective path design	Optimization of meta path construction
Graph Neural Networks (GNN)	Captures both node attributes and structural topology	Intensive computational demand	Enhancing propagation techniques
Multi-view Graph	Integrates multiple data views	Complex model architectures needed	Development of efficient algorithms
Multi-layered Graph	Can handle multi-modal data	Potentially high computational complexity	Leveraging inter-layer connections for deeper insights

Table 3.1: A comparison between different graph embedding-based recommendation methods. [5]

Based on this comprehensive overview, we have identified several methods that particularly captured our interest.

3.2.1. Sequence-Based Methods

One of the straightforward approaches to creating graph-based embeddings is the sequence-based RDF2Vec algorithm. This method significantly draws inspiration from the Word2Vec algorithm, which we previously described. RDF2Vec adapts the principles of natural language processing to graph data by transforming RDF graphs into sequences of RDF nodes. It leverages these sequences to generate vector representations that maintain the semantic relationships inherent in the knowledge graph. [16]

RDF2Vec employs unsupervised techniques from natural language processing to embed RDF graphs into vector spaces. The method involves:

1. **Graph to Sequence Conversion**: Using graph walks or Weisfeiler-Lehman Subtree RDF Graph Kernels, the RDF graph is transformed into a sequence of nodes, mimicking sentences in a textual context.

2. **Sequence Embedding**: These sequences are then embedded into a latent vector space using neural language models, akin to word embeddings in NLP.

Graph Walks Graph walks in RDF2Vec involve traversing the graph to generate sequences of nodes. These walks can be random, depth-first, or breadth-first, capturing different structural aspects of the graph. Each walk starts at a selected node and explores its neighbors, recursively or iteratively, to a specified depth. This method effectively captures local connectivity and can reveal patterns reflective of the graph's topology.

Possible sequences for the example 3 are given here:

- *Paper1, authored_by, AuthorA, works_for, InstitutionX*
- *Paper2, authored_by, AuthorB, cited_by, Paper1*

Weisfeiler-Lehman Subtree RDF Graph Kernels The Weisfeiler-Lehman algorithm for RDF graphs extends traditional graph kernels by adapting to the directed, labeled nature of RDF data. It iteratively aggregates and hashes neighborhood information to capture the local substructures around each node, effectively transforming these substructures into a fixed-length feature vector. This approach is particularly adept at capturing hierarchical and relational information embedded in RDF graphs, making it powerful for tasks requiring detailed semantic understanding.

3.3. Understanding Graph Neural Networks for Embedding Learning

Graph Neural Networks (GNNs) are an extension of traditional neural networks designed to directly work with the complexity of graph data. By integrating node features and graph structure, GNNs efficiently produce embeddings that capture both the properties of individual nodes and their relationships within the graph. In many applications, GNNs are employed in a supervised learning framework where they are trained on tasks like node label classification or link prediction, using labeled data to learn how to accurately predict similar labels on new, unseen graph data.

Initially, each node in a GNN starts with an embedding based on its own set of features, which could range from simple labels to complex attribute vectors. The learning process then begins by incorporating information from the node's neighborhood into the embedding:

Aggregation and Transformation: In each layer of a GNN, nodes aggregate information from their immediate neighbors, combining their features with their own to form a new, aggregated feature set. The specific method of aggregation can vary, ranging from

a straightforward sum to more sophisticated operations, tailored to the particular architecture of the GNN. Following aggregation, the feature set undergoes a transformation, typically through a neural network layer equipped with a non-linear activation function such as ReLU. This transformation refines the aggregated features, enhancing their ability to represent both the individual characteristics of the node and its relational context within the broader graph.

Layered Approach: The process of aggregation and transformation is not a singular occurrence within a GNN. Instead, it is iteratively repeated across multiple layers. Each successive layer enables a node to extend its reach, aggregating and integrating information from an increasingly wider neighborhood. This multilayer progression allows the embeddings to encapsulate a broader context and more profound connections within the graph. Consequently, these embeddings are particularly valuable for tasks that demand an intricate understanding of complex network structures.

Graph Neural Networks (GNNs) encompass a variety of architectures each tailored to specific tasks and data structures within graph-based modeling. As illustrated in Figure 3.1, Recurrent GNNs (RecGNNs) leverage iterative processes over the graph, refining the embeddings gradually by repeatedly processing the information to capture cyclic patterns within the graph structure. Similarly, Convolutional GNNs (ConvGNNs), shown in Figure 3.1b, draw inspiration from the convolutions used in traditional CNNs, adapting these operations to suit the irregular structures of graphs. These networks focus on capturing local neighborhood structures through layers that aggregate and transform node features based on their connectivity.

Graph Autoencoders (GAEs), as depicted in Figure 3.1c, take a different approach by aiming to reconstruct the graph or its adjacency matrix. This reconstruction task helps in learning embeddings that effectively summarize the graph's overall structure, providing a compressed yet informative representation of the entire graph. Spatial-Temporal GNNs (STGNNs), featured in Figure 3.1d, are designed for dynamic graphs, adapting to graphs that evolve over time by capturing the changes in their embeddings, which is crucial for applications like network traffic forecasting or dynamic social network analysis.

3.4. Learning and Optimization in GNNs

The learning process in GNNs is focused on minimizing a loss function that accurately reflects the performance of the network in specific tasks, such as link prediction or node classification, as exemplified by the networks shown in Figures 3.1b and 3.1d. This optimization is typically achieved through backpropagation and gradient descent methods, which adjust the network's weights to reduce prediction errors.

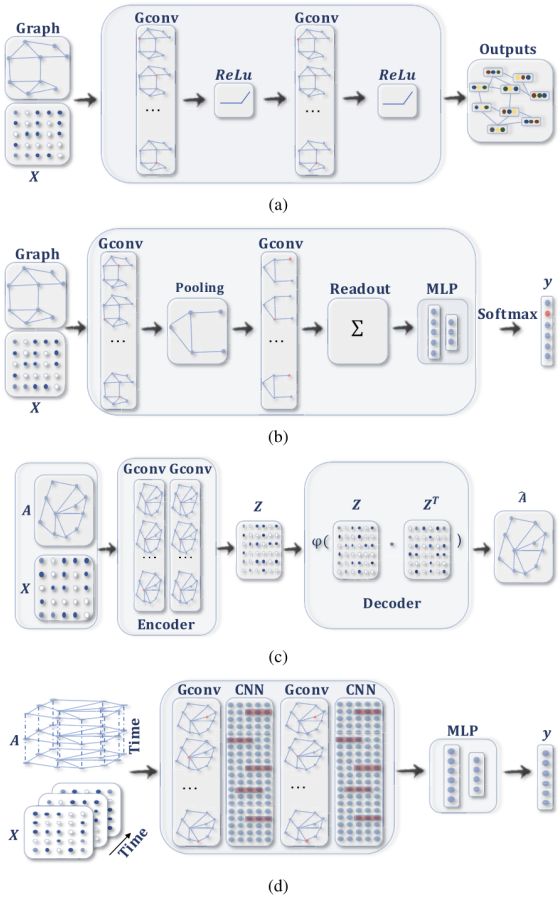

Figure 3.1: Various Graph Neural Network (GNN) models illustrating different architectural adaptations for graph data processing: (a) basic GNN model, (b) GNN with pooling and readout layers, (c) Graph Autoencoder model, and (d) Spatial-Temporal GNN model.

4. Graph Embeddings, Reality

Papers will usually do bold claims on how a method can be used to generate embeddings that can be used to make machine learning models for classification, or make link predictions. In reality, is hard to implement most methods in a fast pacing industry with a small margin for errors and downtime. When needing to use embeddings for an application developers need tools that make this process viable, deployable and maintainable.

We aim at showing tools that aid developers in using KGE in production environments. We show a Python library, an extension for Neo4j and a tool that manage embeddings Vector Databases.

4.1. Pykeen

Pykeen[1][7] is a Python library for KGE. The library serves as a commonground for multiple models, making it easy to compare and try out different models. It allows users to bring their own data in a format of csv of three columns where each column represents subject, predicate and object.

Pykeen contains over 40 models including translation models like TransD, TransE, TransF, TransH, TransR. It contains some datasets like Wikidata, DBpedia, Unified Medical Language System and WordNet allowing for enrichment of new datasets. It implements CUDA accelleration that drastically improves performance for KGE based on NN.

Pykeen has a very simplistic syntax. Given a graph containing information in triples like *slovakia locatedin europe*, *slovakia neighbor hungary*... embeddings can be achieved by using the following code:

```
from pykeen.pipeline import pipeline

result = pipeline(
    model='TransE',
    dataset='countries',
)
```

Predicting links is equally simple, since predicting a neighbor of a country would simply mean:

```
from pykeen import predict
predict.predict_target(
    model=result.model,
    head="colombia",
    relation="neighbor",
    triples_factory=result.training,
)
```

4.2. Neo4j

Neo4j is the most famous knowledge graph database. It currently support 4 different models FastRP, GraphSAGE, Node2Vec, and HashGNN. The only production ready embedding is FastRP, the remainder are in Beta.

If a company is already using Neo4j for its data, KGE can enrich their processes and algorithms with little overhead and great benefits. Being native to Neo4j allows for businesses to have one destiny for their graph needs, that although limited it is enough for their needs.

4.3. Vector Databases

To efficiently store and handle created embeddings, the best option is to use a vector database. This type of database is specifically designed for storing vectors and uses optimized

[7]Documentation can be found in https://pykeen.readthedocs.io/en/

algorithms for storing and indexing, which enhances querying. Popular vector databases include Pinecone[8], Milvus[9], and Faiss[10]. . A straightforward use case for vector databases in recommendation systems is as follows: Given a vector, the database is queried for the closest vectors to the provided one. This approach can be used for recommendations, as the database returns a list of similar entities.

5. Graph Recommendations

Even though this paper has focused on KGE, this is not the only way of producing recommendations using graphs. One of the most famous methods is **Matrix Factorization**[10]. It became famous on the Netflix Challenge where given some users and their ratings in movies we want to predict the users ratings in other movies. The problem is reduced to decomposing the sparse matrix of users and ratings into users and latent factor using optimization. This method is considered one of the first and simplest types of graph embedding[12].

Although a simple model, matrix factorization should not be discarded, research[5] shows that the performance of this model is similar to current neural networks approaches. There has been Bayesian approaches to matrix factorization such as probabilistic matrix factorization (PMF)[14] that aim to solve the optimization problems inherent to sparse matrices.

There is a different approach to graph recommendation based on embeddings. One example is YoutubeNet[4] that uses a combination of embeddings coupled with some extra layers to recommend content to users. Embeddings are not a final goal but a tool, we can combine information from different sources using KGE given that KG are specially good at integrating large volumes of data.

Another recommendation models that follow the trend of deep learning is **Autoencoders** [18], a model that aims at predict the ratings of a user by getting the latent information of the user as mentioned in section 3. For usage in a recommendation system Autoencoders start with a vector of dimension equaling the number of all items and use deep learning model to predict the missing ratings. Autoencoders are composed of an encoder and a decoder, the encoder learns an embedding of the information, that the decoder uses to recreate the information given.

Using only one relationship type can be quite limiting, Heterogeneus Information Network[20](HIN) builds on adding more information to the graph. Knowledge Graphs are good at combining multiple kinds of information, we can have information of what a user like, but also about what a content is. HIN takes a diffusion approach where random walks are taken with the user at the center, then it uses matrix factorization for recommendations.

Embeddings are interesting for allowing geometric interpretations of the elements we embed. In NLP there are classic examples of taking the embedding of "king" substracting "male" and getting "queen". In graph embeddings there are similar ideas where we get one *user* and add the last **item** he consumed and it gets us the *next item* he will consume, i.e. a

[8]https://www.pinecone.io/
[9]https://milvus.io/
[10]https://github.com/facebookresearch/faiss

recommendation. This **Translation-based Recommendation**[8] is based on **TransE**[11][3]. TransE is a model that understand the semantic of a relation as wanting to the relation to represent a translation in the embedded space.

Embeddings are one of the tools available to generate recommendations based on graphs, but it is neither the absolute best option or the most explainable. When creating a recommendation factor multiple options should be balanced with the available data to get the most optimal result.

6. Conclusion

KG are useful for data integration, and KGE can be more relevant in context where recommendations are not based on a bipartite graph of users and product. More simple methods of embeddings like matrix factorization, perform well and should not be discarded when building a recommendation system. KGE have plenty of applications since they allow including graph information to most machine learning models, when used in combination of classic clustering and classification techniques embeddings can greatly aid performance, such is the case of YoutubeNet. Research should not only aim at proving a point in academia, but bringing the tools to practitioners like Pykeen and Neo4j do, most embedding methods that are not easily implemented will hardly be used in the industry.

[11] Implemented in Pykeen `https://pykeen.readthedocs.io/en/stable/api/pykeen.models.TransE.html#pykeen.models.TransE`

References

[1] M. Ali, M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp, and J. Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82): 1–6, 2021.

[2] G. Antoniou and F. Van Harmelen. *A semantic web primer*. MIT press, 2004.

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

[4] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. *RecSys 2016 - Proceedings of the 10th ACM Conference on Recommender Systems*: 191–198, 2016.

[5] Y. Deng. *Recommender Systems Based on Graph Embedding Techniques: A Review*. Vol. 10. IEEE, 2022, pp. 51587–51633.

[6] D. Fensel, U. Simsek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich, and A. Wahler. *Knowledge graphs*. Springer, 2020.

[7] C. A. Gomez-Uribe and N. Hunt. The netflix recommender system: algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4): 1–19, 2015.

[8] R. He, W.-C. Kang, and J. McAuley. Translation-based recommendation. *Proceedings of the eleventh ACM conference on recommender systems,* 161–169, 2017.

[9] S. Ji, S. Pan, E. Cambria, P. Marttinen, and S. Y. Philip. A survey on knowledge graphs: representation, acquisition, and applications. *IEEE transactions on neural networks and learning systems*, 33(2): 494–514, 2021.

[10] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37, 2009.

[11] W. Liu, P. Zhou, Z. Zhao, Z. Wang, Q. Ju, H. Deng, and P. Wang. K-bert: enabling language representation with knowledge graph. *Proceedings of the AAAI Conference on Artificial Intelligence,* vol. 34. 03. 2901–2908, 2020.

[12] I. Makarov, D. Kiselev, N. Nikitinsky, and L. Subelj. Survey on graph embeddings and their applications to machine learning problems on graphs. *PeerJ Computer Science*, 7: 1–62, 2021.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[14] A. Mnih and R. R. Salakhutdinov. Probabilistic matrix factorization. *Advances in neural information processing systems*, 20, 2007.

[15] R. H. Richens. Preprogramming for mechanical translation. *Mech. Transl. Comput. Linguistics*, 3(1): 20–25, 1956.

[16] P. Ristoski and H. Paulheim. Rdf2vec: rdf graph embeddings for data mining bt - the semantic web – iswc 2016: 498–514, 2016.

[17] M. A. Rodriguez. The gremlin graph traversal machine and language (invited talk). *Proceedings of the 15th Symposium on Database Programming Languages,* 1–10, 2015.

[18] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: autoencoders meet collaborative filtering. *Proceedings of the 24th international conference on World Wide Web,* 111–112, 2015.

[19] N. Torzec. The yahoo! knowledge graph. *Proceedings of Semantic Technology and Business Conference,*

[20] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han. Personalized entity recommendation: a heterogeneous information network approach. *Proceedings of the 7th ACM international conference on Web search and data mining,* 283–292, 2014.