Recommendations using Knowledge Graph Embeddings

Simon Coessens, Gabriel Lozano - Universitat Politecnica de Catalunya

Introduction

Around 80% of Netflix hours come from a recommendation. Usage of graphs in recommendation has proven to be successful, but using the information contained in a graph database is not straightforward. Graphs are complicated and hard to summarize, in this poster we see some graph embeddings techniques that should breach the gap between having graph information and using it to build a recommender system.

Embeddings

Embeddings are a transformation we make to data so that we can represent it in a $\mathbb{R}^d$ vector space where d is the dimension of our embedded data.

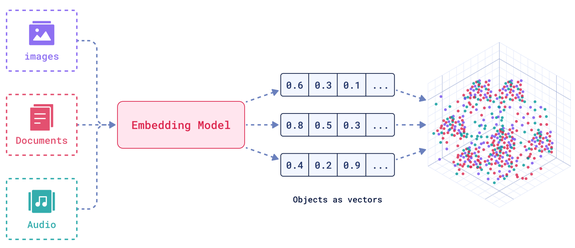


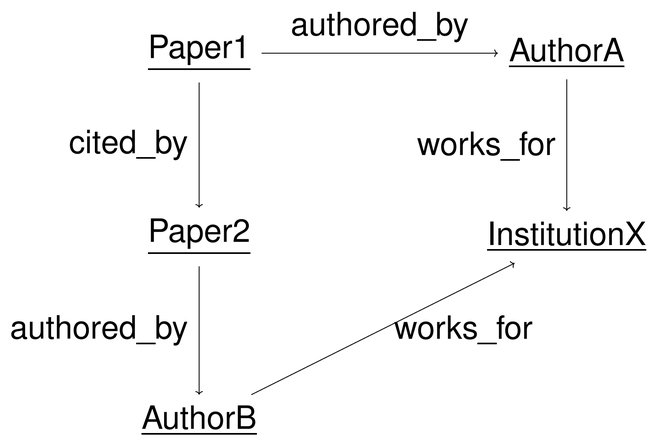
Fig. 1: Image taken from https://qdrant.tech/articles/what-are-embeddings/

Embeddings are commonly used in NLP to represent the abstract concept of a word, in a way that makes sense to a computer. The advantage words have over nodes is that they are sequenced, thus embeddings tend to use word proximity and order to understand their meaning. There is no order or sequencing in graphs, thus current embedding models work on how to make the embedding make sense.

Graph Embeddings

Just as we embed words in natural language, we can embed nodes in a graph. These embeddings capture node characteristics and relationships, crucial for machine learning algorithms.

A graph embedding $f : V \to \mathbb{R}^d$ maps each vertex v in graph $G = (V, E)$ to a d-dimensional vector, preserving graph properties.



RDF2Vec Explained:

- **Graph Walks:** RDF2Vec employs graph walks to traverse the RDF graph.They generate sequences of nodes that reflect the connectivity and relationships within the graph.
- **Sequence Embedding:** The sequences generated from graph walks are then embedded into a latent vector space using similar techniques to word2vec.

Example Sequences Derived from RDF2Vec:

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

Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are advanced extensions of traditional neural networks designed to handle the complexity of graph data. By integrating node features and graph structure, GNNs generate embeddings that capture both individual properties of nodes and their relationships within the graph.

Key Processes in GNNs:

- **Aggregation and Transformation:** Nodes in each layer aggregate information from their immediate neighbors, combining features to form a new aggregated set. This set is then transformed through a neural network layer with non-linear activation, typically ReLU, enhancing their ability to represent node and relational properties.

- **Layered Approach:** Aggregation and transformation are iteratively repeated across multiple layers. This enables nodes to incorporate information from an increasingly wider neighborhood, enriching the embeddings and allowing them to represent complex network structures more effectively.

Diverse Architectures: GNN architectures include Recurrent GNNs (RecGNNs), Convolutional GNNs (ConvGNNs), Graph Autoencoders (GAEs), and Spatial-Temporal GNNs (STGNNs), each tailored for specific tasks and dynamic graph structures.

Learning and Optimization: GNNs minimize a loss function to improve task-specific performances such as node classification or link prediction, employing methods like backpropagation and gradient descent.

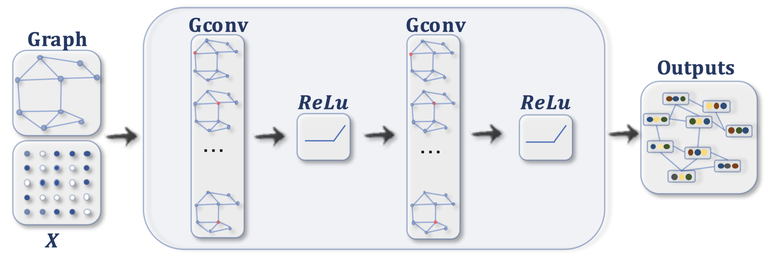


Fig. 2: Image showing a ConvGNN model with multiple graph convolutional layers that aggregate and transform neighbor features, extending each node's influence across the network.

Graph Embedding Tools

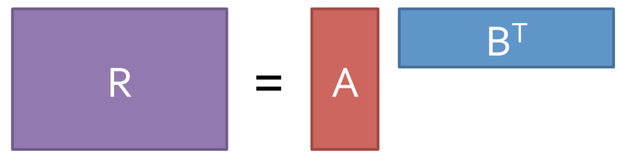
We have identified three main tools to make graph embeddings a viable tool for practitioners.

- **Pykeen**: A python library that integrates multiple state of the art embeddings. It brings them to a common format to easily swap across them.
- **Neo4j Libraries**: Neo4j is the most popular graph database. It has integrated embeddings that allow implementing embeddings into production without adding additional dependencies.
- **Vector Databases**: Pinecone and similar databases allow users to manage embeddings with easy. Calculating common neighbours and updating embeddings becomes a fast a reliable operation.

Graph Recommendation Using Embeddings

We start with simple models and then we can make more complicated models.

- **Matrix factorization**: Let R be a $c \times u$ **sparse** matrix where the columns are the content and the rows are the users. Then matrix factorization is finding A $(u \times k)$ and B $(k \times c)$ such that:



- **Autoencoders**: Autoencoders like AutoRec start with a vector of dimension equaling the number of all content and use deep learning model to predict the missing ratings.
- **Translation methods**: We can give meaning to the embeddings. Operations can have a significance in what we do. If the triple spo exists, then the embedding should respect $s + p = o$. Similarly *TransR* makes it so that $user + currentContent = nextContent$.

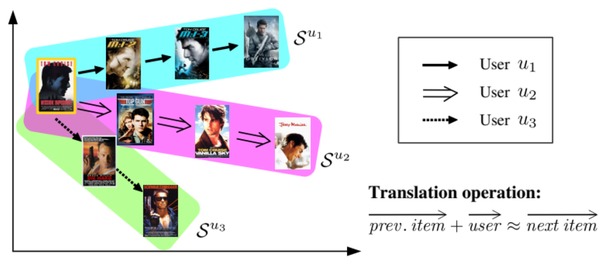


Fig. 4: He, Ruining "Translation-based recommendation." 2017.

- **Link prediction**: Given a user we can predict the content he might like using the loss function of the embedding.
- **Machine learning**: Given embeddings we can use any ML method we want, like kNN.

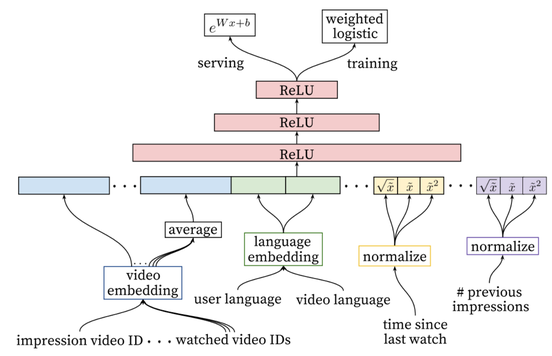


Fig. 5: Covington, Paul "Deep neural networks for youtube recommendations." 2016.

Conclusions

- **Enhanced Capabilities:** Knowledge Graph Embeddings (KGE) significantly improve recommendation systems by providing deeper interaction analysis.
- **Embedding Techniques:** Methods such as GNNs and RDF2Vec leverage complex relational data, offering insights that go beyond simpler methods like matrix factorization.
- **Practical Implementation:** Tools like Pykeen and Neo4j facilitate the integration of KGE into production environments, demonstrating the transition from theoretical models to practical applications.