From <https://towardsdatascience.com/introduction-to-knowledge-graph-embedding-with-dgl-ke-77ace6fb60ef> :

Introduction to Knowledge Graph Embeddings

Learn about Knowledge Graphs embeddings and two popular models to generate them with DGL-KE

Author: Cyrus Vahid, Da Zheng, George Karypis and Balaji Kamakoti: AWS AI

Jun 15, 2020

Knowledge Graphs (KGs) have emerged as an effective way to integrate disparate data sources and model underlying relationships for applications such as search. At Amazon, we use KGs to represent the hierarchical relationships among products; the relationships between creators and content on Amazon Music and Prime Video; and information for Alexa's question-answering service. Information extracted from KGs in the form of embeddings is used to improve search, recommend products, and infer missing information. It is however hard to train KG embeddings from graphs with millions of nodes and billions of edges.

Amazon recently launched DGL-KE, a software package that simplifies this process with simple command-line scripts. With DGL-KE, users can generate embeddings for very large graphs 2–5x faster than competing techniques. DGL-KE provides users the flexibility to select models used to generate embeddings and optimize performance by configuring hardware, data sampling parameters, and the loss function. To use this package effectively, however, it is important to understand how embeddings work and the optimizations available to compute them. This two-part blog series is designed to provide this information and get you ready to start taking advantage of DGL-KE.

What is a graph

A graph is a structure used to represent things and their relations. It is made of two sets — the set of nodes (also called vertices) and the set of edges (also called arcs). Each edge itself connects a pair of nodes indicating that there is a relation between them. This relation can either be *undirected*, e.g., capturing symmetric relations between nodes, or *directed*, capturing asymmetric relations. For example, if a graph is used to model the friendship relations of people in a social network, then the edges will be undirected as they are used to indicate that two people are friends; however, if the graph is used to model how people follow each other on Twitter, the edges will be directed. Depending on the edges' directionality, a graph can be *directed* or *undirected.*

Graphs can be either *homogeneous* or *heterogeneous*. In a homogeneous graph, all the nodes represent instances of the same type and all the edges represent relations of the same type. For instance, a social network is a graph consisting of people and their connections, all representing the same entity type. In contrast, in a heterogeneous graph, the nodes and edges can be of different

types. For instance, the graph for encoding the information in a marketplace will have *buyer*, *seller*, and *product* nodes that are connected via *wants-to-buy*, *has-bought*, *is-customer-of*, and *is-selling* edges.

Finally, another class of graphs that is especially important for knowledge graphs are *multigraphs*. These are graphs that can have multiple (directed) edges between the same pair of nodes and can also contain loops. These multiple edges are typically of different types and as such most multigraphs are heterogeneous. Note that graphs that do not allow these multiple edges and self-loops are called *simple* graphs.

What is a knowledge graph

In the earlier marketplace graph example, the labels assigned to the different node types (buyer, seller, product) and the different relation types (wants-to-buy, has-bought, is-customer-of, is-selling) convey precise information (often called *semantics*) about what the nodes and relations represent for that particular domain. Once this graph is populated, it will encode the knowledge that we have about that marketplace as it relates to types of nodes and relations included. Such a graph is an example of a knowledge graph.

A knowledge graph (KG) is a directed heterogeneous multigraph whose node and relation types have domain-specific semantics. KGs allow us to encode the knowledge into a form that is human interpretable and amenable to automated analysis and inference. KGs are becoming a popular approach to represent diverse types of information in the form of different types of entities connected via different types of relations.

When working with KGs, we adopt a different terminology than the traditional vertices and edges used in graphs. The vertices of the knowledge graph are often called *entities* and the directed edges are often called *triplets* and are represented as a (*h, r, t*) tuple, where *h* is the head entity, *t* is the tail entity, and *r* is the relation associating the head with the tail entities. Note that the term *relation* here refers to the type of the relation (e.g., one of wants-to-buy, has-bought, is-customer-of, and is-selling).

Let’s now examine a KG with cast of a people and the world in which they live.

Scenario:
Mary and **Tom** are ***siblings*** and they both are ***vegetarians***, who ***like*** **potatoes** and **cheese**. Mary and Tom both ***work*** at **Amazon**. **Joe** is a bloke who is a ***colleague*** of Tom. To make matters complicated, Joe ***loves*** Mary, but we do not know if the feeling is reciprocated.

Joe ***is from*** **Quebec** and is proud of his native dish of **Poutine**, which is ***composed*** of potato, cheese, and **gravy**. We also know that gravy ***contains*** **meat** in some form.

Joe is excited to invite Tom for dinner and has sneakily included his sibling, Mary, in the invitation. His plans are doomed from the get-go as he is planning to serve the vegetarian siblings his favorite Quebecois dish, Poutine.

Oh! by the way, a piece of geography trivia: Quebec ***is located*** in a **province** of the same name which in turn ***is located*** in **Canada**.

There are several relationships in this scenario that are not explicitly mentioned but we can simply infer from what we are given:

- Mary is a colleague of Tom.
- Tom is a colleague of Mary.
- Mary is Tom's sister.
- Tom is Mary's brother.
- Poutine has meat.
- Poutine is not a vegetarian dish.
- Mary and Tom would not eat Poutine.
- Poutine is a Canadian dish.
- Joe is Canadian.
- Amazon is a workplace for Mary, Tom, and Joe.

Some interesting negative conclusions seem intuitive to us, but not to the machine:

- Potato *does not like* Mary.
- Canada *is not from* Joe.
- Canada *is not located* in Quebec.
- …
  What we have examined is a knowledge graph, a set of nodes with different types of relations:
- 1-to-1: Mary is a sibling of Tom.
- 1-to-N: Amazon is a workplace for Mary, Tom, and Joe.
- N-to-1: Joe, Tom, and Mary work at Amazon.
- N-to-N: Joe, Mary, and Tom are colleagues.

There are other categorization perspectives on the relationships as well:

- Symmetric: Joe is a colleague of Tom entails Tom is also a colleague of Joe.
- Antisymmetric: Quebec is located in Canada entails that Canada cannot be located in Quebec.

Figure 1 visualizes a KG that describes the *World of Mary*. For more information on how to use the examples, please refer to the code embedded.

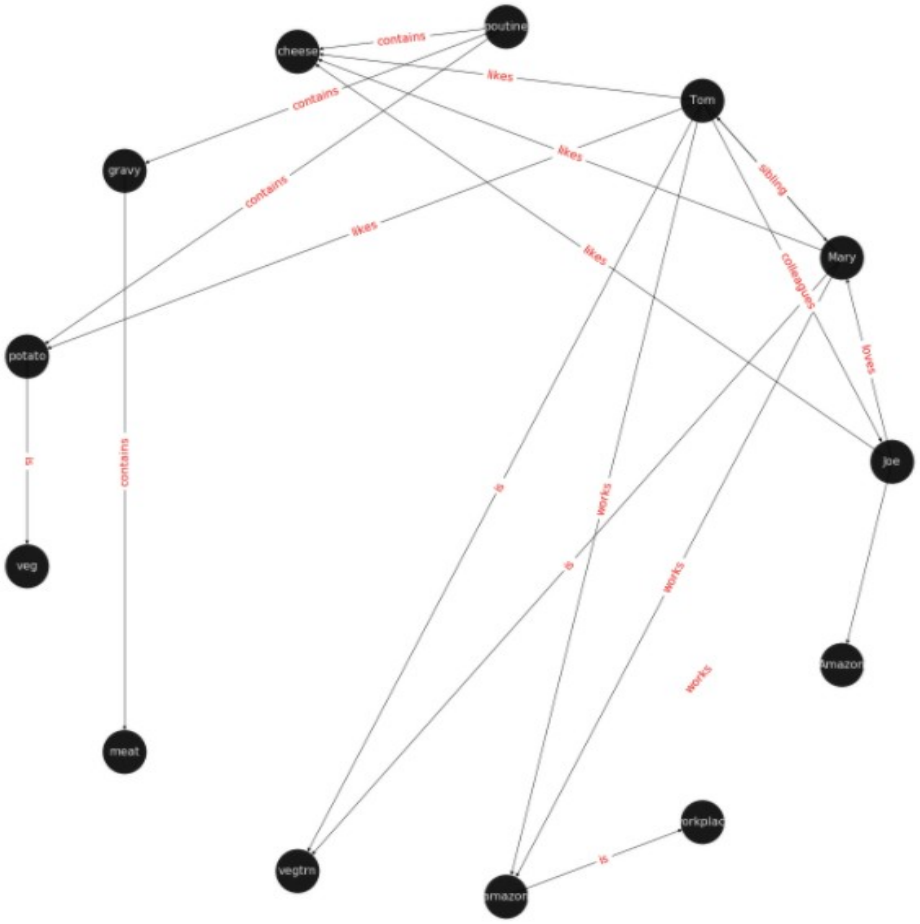

Figure1: World of Mary

Knowledge graph embeddings

Knowledge graph embeddings (KGEs) are low-dimensional representations of the entities and relations in a knowledge graph. They provide a generalizable context about the overall KG that can be used to infer relations. In the World of Mary visualized above, embeddings provide insights about relations among Mary, Joe, and Tom. In a larger and more dense KG, embeddings could provide insights about molecular property interactions to accelerate drug discovery or cluster user behaviors of scammers in a gaming network.

The knowledge graph embeddings are computed so that they satisfy certain properties; i.e., they follow a given KGE *model*. These KGE models define different *score functions* that measure the distance of two entities relative to its relation type in the low-dimensional embedding space. These score functions are used to train the KGE models so that the entities connected by relations are close to each other while the entities that are not connected are far away.

There are many popular KGE models such as TransE, TransR, RESCAL, DistMult, ComplEx, and RotatE, which define different score functions to learn entity and relation embeddings. DGL-KE makes these implementations accessible with a simple input argument in the command line script. In this post, we introduce and compare TransE and TransR, two common methods to provide

readers some intuition about the models and the tradeoffs

TransE

Translation based embedding model (TransE) is a representative translational distance model that represents entities and relations as vectors in the same semantic space of dimension Rd, where d is the dimension of the target space with reduced dimension. A fact in the source space is represented as a triplet (h,r,t) where h is short for the *head*, r is for the *relation*, and t is for the *tail*. The relationship is interpreted as a translation vector so that the embedded entities are connected by relation r have a short distance. [3, 4]

In terms of vector computation, it could mean adding a head to a relation should approximate to the relation's tail or h+r ≈ t.

For example if
$h_1 = emb(" \ Ottawa \ "), \ h_2 = emb(" \ Berlin \ "), t_1 = emb(" \ Canada \ "), t_2 = (" \ Germany \ ")$
, and finally $r =" \ CapilatOf \ "$, then $h_1 + r$ and $h_2 + r$ should approximate t_1 and t_2
respectively.

TransE performs linear transformation and the scoring function is negative distance between:

$h + r$ and t, or $f = -\|h + r - t\|_{\frac{1}{2}}$

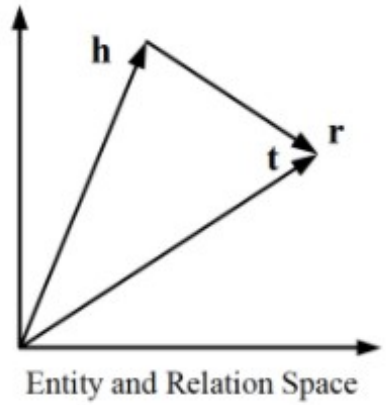

Figure 3: TransE

TransR

TransE cannot cover a relationship that is not 1-to-1 as it learns only one aspect of similarity. TransR addresses this issue with separating relationship space from entity space where h, t ∈ $\mathbb{R}^k$ and $r ∈ $\mathbb{R}^d$. The semantic spaces do not need to be of the same dimension. In the multi-relationship modeling, we learn a projection matrix $M \in \mathbb{R}^{kxd}$ for each relationship that can project an entity to different relationship semantic spaces. Each of these spaces captures a different aspect of an entity that is related to a distinct relationship. In this case, a head node, h, and tail node, t with a relationship r are projected into the relationship space using the learned projection matrix M_r as $h_r=hM_r$ and $t_r=tM_r$ respectively. Figure 4 illustrates this projection.

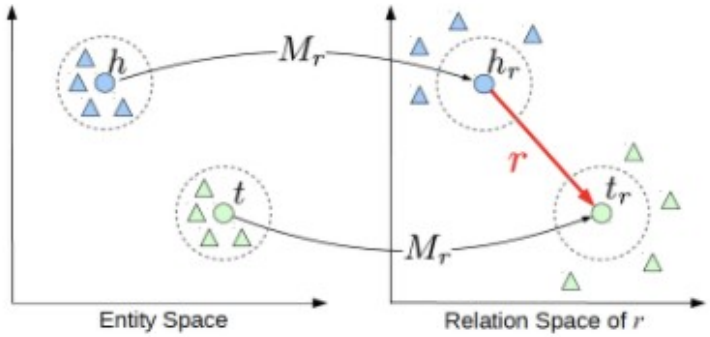

Figure 4: TransR projecting different aspects of an entity to a relationship space.

Let us explore this using an example. Mary and Tom are siblings and colleagues. They both are vegetarians. Joe also works for Amazon and is a colleague of Mary and Tom. TransE might end up learning very similar embeddings for Mary, Tom, and Joe because they are colleagues but cannot recognize the (not) sibling relationship. Using TransR, we learn projection matrices: M_sibling, M_colleague, and M_vegetarian that perform better at learning relationship like (not)sibling.

The score function in TransR is similar to the one used in TransE and measures the euclidean distance between h+r and t, but the distance measure is per relationship space. More formally:

$$f_r = \|h_r + r - t_r\|_2^2$$

Another advantage of TransR over TransE is its ability to extract compositional rules. The ability to extract rules has two major benefits. It offers richer information and has a smaller memory space as we can infer some rules from others.

Tradeoffs

The benefits from more expressive projections in TransR adds to the complexity of the model and a higher rate of data transfer, which has adversely affected distributed training. TransE requires O(*d*) parameters per relation, where dd is the dimension of semantic space in TransE and includes both entities and relationships. As TransR projects entities to a relationship space of dimension kk, it will require O(*kd*) parameters per relation. Depending on the size of k, this could potentially increase the number of parameters drastically. In exploring DGL-KE, we will examine the benefits of DGL-KE in making computation of knowledge embedding significantly more efficient. [5], [7]

For a detailed overview of other implementations in DGL-KE, please check out our documentation.

What's Next?

In this blog post, we introduced the concept of Knowledge Graph embeddings (KGEs), how they work and two popular methods to generate KGEs. In our next post, we will explore how options available to accelerate training with DGL-KE.

References

1. http://semantic-web-journal.net/system/files/swj1167.pdf
2. Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. CoRR, abs/1902.10197, 2019.
3. Knowledge Graph Embedding: A Survey of Approaches and Applications Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. DOI 10.1109/TKDE.2017.2754499, IEEE Transactions on Knowledge and Data Engineering
4. transE: Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, JasonWeston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Advances in Neural Information Processing Systems 26. 2013.
   5.TransR: Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
5. RESCAL: Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11, 2011.
6. Survey paper: Q. Wang, Z. Mao, B. Wang and L. Guo, "Knowledge Graph Embedding: A Survey of Approaches and Applications," in IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 12, pp. 2724–2743, 1 Dec. 2017.
7. DistMult: Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Proceedings of the International Conference on Learning Representations (ICLR) 2015, May 2015.
8. ComplEx: Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. CoRR, abs/1606.06357, 2016.
9. Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. CoRR, abs/1902.10197, 2019.