**Knowledge Graph Embeddings**

**Types of graphs (Revision):**

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

In a heterogeneous graph, the nodes and edges can be of different types

ex.: 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.

Another class of graphs: multigraphs

graphs that can have multiple (directed) edges between the same pair of nodes and can also contain loops

**What is 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

**Terminology:**

The vertices of the knowledge graph are often called entities and the directed edges are often called triplets (I think this is a mistake from the article, it is just called a relation) and are represented as a (h, r, t) (this tuple is called triples) tuple

**Knowledge graph embeddings**

low-dimensional representations of the entities and relations in a knowledge graph.

provide a generalizable context about the overall KG that can be used to infer relations.

**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 taking into consideration the relation 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.

popular KGE models such as

1. TransE
2. TransR
3. RESCAL
4. DistMult
5. ComplEx
6. RotatE

**TransE Model:**

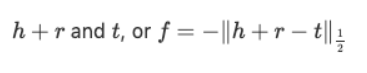
translational distance model that represents entities and relations as **vectors** in the same semantic space of dimension **Rd**, where **d is the dimension of the target space with reduced dimension**

**(don’t remember reading about how do we decide the dimensionality of the target space -> still to research)**

The relationship is interpreted as a translation vector so that the embedded entities are connected by relation r have a short distance for positive samples and longer distances for negative samples.

adding a head to a relation should approximate to the relation’s tail or h+r ≈ t

(sounds similar to the idea of why we represent words as vectors using word embeddings (to possibly be able to find similar words and do arithmetic))

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

(find it interesting that we take the negative of the distance, although possibly it has to do with how it is linked to the loss functions later,

Also I have read that the higher the score the better, but that doesn’t make much sense, as the distance approaches zero the better the model (in a good model, the entities that are linked by a relation are close to one another using the for h+r ~= t))

**TransR** **Model:**

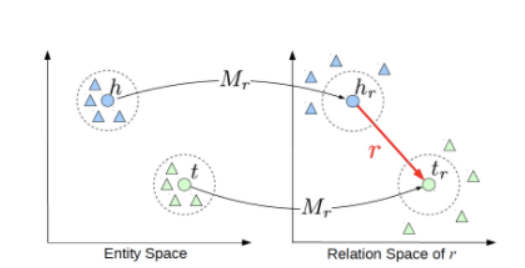
However, an entity may have multiple aspects, and various relations focus on different aspects of entities. Hence, it is intuitive that some entities are similar and thus close to each other in the entity space, but are comparably different in some specific aspects and thus far away from each other in the corresponding relation spaces.

TransR addresses this issue with separating relationship space from entity space where h, t ∈ ℝᵏ and $r ∈ ℝᵈ.

In the multi-relationship modeling, we learn a projection matrix M∈ℝᵏˣᵈ 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 Mᵣ as hᵣ=hMᵣ and tᵣ=tM ᵣrespectively.



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:



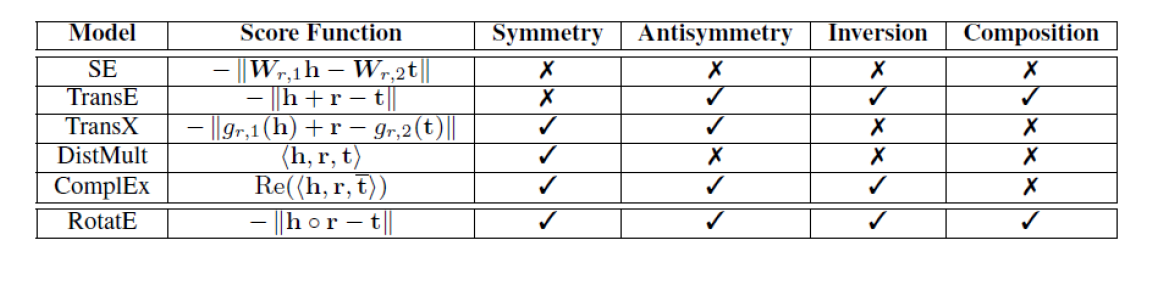
**(so the model is similar to the transE model In that we check the distance between the head and the tail given the relation, but here we map the head and the tail entities using a matrix specific to each relation Mr, and I’m still not sure how do we decide the dimensionality of the spaces of Rk & Rd, so that for 2 entities h1 and t1, if they have a relation R1, they would be close after mapping them using Mr1 and for any other relation, they would be further, when using the mappings for these relations Mri)**

**(I think though maybe, if we can change the distance between the entities according to how many relations they have between them, so the more relations h and t have the closer they are, and vice versa, being the farthest when having no relations between them, that can maybe solve the problem of transE model, and maybe while generating the negative sampling we also generate a false sample with (h, t, r’),where r’ is a random relation, I’m not sure if it has been implemented before or if it would actually work (haven’t had the chance to test it in code))**

**RotatE Model:**

the model leverages the idea that if we have a unitary complex vector (Euler’s identity eitheta = cos(theta)+ i sin(theta), which) and we multiply a vector by it, it is as if we have rotated by theta angles, so the idea is that if we represent the relation vector as a unitary complex vector and multiply it by the head entity h, then for a well-trained model, it should be close to tail entity t.

* the scoring function is as follows:****
* comparision of RotatE to other models

****

**Training:**

for some reason, I was confused on whether we change the position of the entities vectors or the relation vectors during training using the loss function, so I tried to search for it.

For example, training on a triple (s, p, o) is achieved by learning the right classes "s" and "o" for the pairs (?, p, o) and (s, p, ?)

so it means that we adjust the position of the entities when using gradient descent not the length of the relations in the graph, makes sense especially for RotatE, as the relation vector length is 1, and also the false samples only consider false samples of the (head entity) for ex

**Loss functions: (still todo)**

**About the code:**

* I have reviewed the code in this github first (<https://github.com/Lapis-Hong/TransE-Knowledge-Graph-Embedding>), wanting to try to understand see how transE was implemented, and then extending it to support the RotatE model, unfortunately I wasn’t able to make it work, don’t remember the exact reason, and also found the code written in tensorflow, fairly hard to understand (maybe it’s my inexperience or maybe the code was written in an unnecessarily complex way)
* So I later I reviewed the code in (<https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding>), found to be a lot more understandable, also coincidentally, found out later that it was the code used by Shayan in his thesis, still haven’t understood 100% of the code, but I think I have a fairly good grasp of how it was written, also it didn’t work because of **CUDA out of memory problems** on my personal laptop, even after decreasing the batch size, not sure it can be solved, or is it a limitation of the my laptop and the solution is to use a GPU with more memory and CUDA cores

**Shayan’s thesis:**

* **Done:**

**Chapter 1**

**Chapter 2 (excluding 2.2 (bash files) 2.53 & 2.54 (the SDA Servers & HPC parts))**

**Chapter 3 (semi done, would have to try myself later)**

**Chapter 4 (excluding**

**4.2.2 (the part about getting detailed information about each relation’s characteristics))**

**4.3 (excluding models other than of RotatE & TransE)**

**4.4 (loss functions)**

**Chapter 5 & 6**

Have seen the code from the RotatE paper, so have familiarised myself with the code from model.py (arguments for the class (including the bash file for run), RotatE, TransE, and training process).

Sources:

1. **Introduction to Knowledge Graph Embeddings**

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

1. **Learning Entity and Relation Embeddings for Knowledge Graph Completion**

Yankai Lin1, Zhiyuan Liu1\_, Maosong Sun1;2, Yang Liu3, Xuan Zhu3

1. **Loss Functions in Knowledge Graph Embedding Models**<https://alammehwish.github.io/dl4kg-eswc/papers/paper%201.pdf>
2. **RotatE: Knowledge graph embedding by Relational Rotation in Complex Space**

Zhiqing Sun 1, Zhi-Hong Deng1, Jian-Yun Nie3, Jian Tang2;4;5