What is a Knowledge Graph

“A knowledge graph (i) mainly describes real world entities and their interrelations, organized in a graph, (ii) defines possible classes and relations of entities in a schema, (iii) allows for potentially interrelating arbitrary entities with each other and (iv) covers various topical domains.” [Paulheim- 1]

Knowledge Graphs are often modeled as directed multigraphs. As a quick reminder let us define Multigrpah and directed graphs: - In a directed graph edges are called arcs and they can direct information from their head to their tail but not in the opposite direction. - A multigraph is a graph that can includes loops as well as multiple relationships between the same nodes. for more information on Graphs you can check this link

Let us examine a directed multigraph in an example, which include a cast of characters and the world in which they live.

scenario:

Mary and **Tom** are ***siblings*** and they both are ***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 the matter 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 and has sneakily included his sibling, Mary, in the invitation. His plans are doomed from get go as he is planning to serve the vegetarian siblings his favourite 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**.

end of scenario

There is 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.

There are also some interesting negative conclusions that seems 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 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: visualized a knowledgebase that describes *World of Mary*. For more information on how to use the examples, please refer to the code that draws the examples.

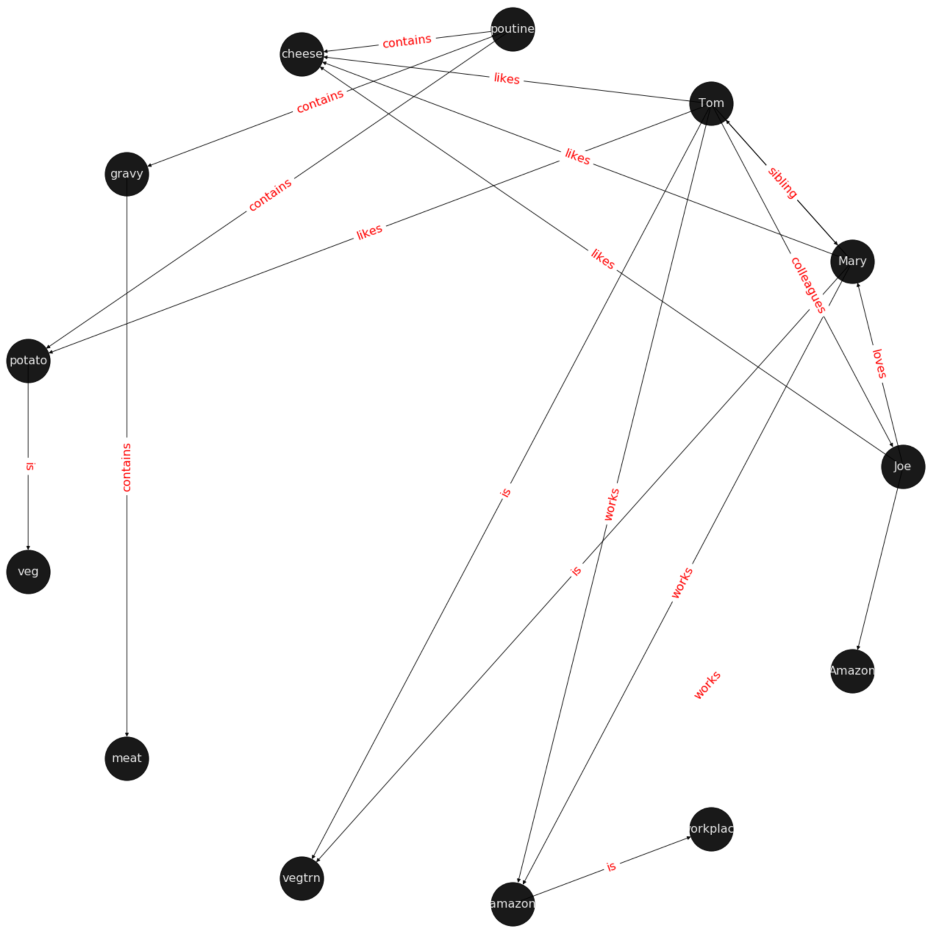


Figure1: World of Mary

What is the task of Knowledge Graph Embedding?

Knowledge graph embedding is the task of completing the knowledge graphs by probabilistically inferring the missing arcs from the existing graph structure. KGE differs from ordinary relation inference as the information in a knowledge graph are multi-relational and more complex to model and computationally expensive. For this rest of this blog, we examine fundamentals of KGE.

Common connectivity patterns:

The connectivity or relational pattern are commonly observed in KG. A Knowledge Graph Embedding model intends to predicts missing connections that are often one of the below types.

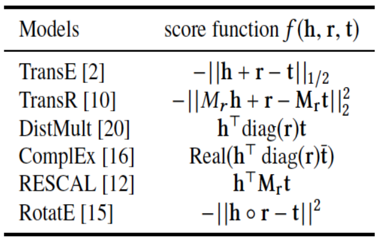
- ***symmetric***
  - **Definision:** A relation r is ***symmetric** if $\forall x, y : (x, r, y) \implies (y, r, x)$
  - **Example:** $\text{x=Mary and y=Tom and r="is a sibling of"}; (x, r, y) = \text{Mary is a sibling of Tom} \implies (y, r, x) = \text{Tom is a sibling of Mary}$
- ***antisymmetric***
  - **Definision:** A relation r is ***antisymmetric** if $\forall x, y : (x, r, y) \implies \neg(y, r, x)$
  - **Example:** $\text{x=Quebec and y=Canada and r="is located in"}; (x, r, y) = \text{Quebec is located in Canada} \implies (y, \neg r, x) = \text{Canada is not located in Quebec}$
- ***inversion***
  - **Definition:** A relation r_1 is ***inverse*** to relation r_2 if $\forall x, y : r_2(x, y) \implies r_1(y, x)$.
  - **Example:** $x = Mary,\ y = Tom,\ r_1 = \text{"is a sister of "}\ and r_2 = \text{"is a brother of"}$
    $(x, r_1, y) = \text{Mary is a sister of Tom} \implies (y, r_2, x) = \text{Tom is a brother of Mary}$
- ***composition***
  - **Definition**: relation r_1 is composed of relation r_2 and relation r_3 if $\forall x, y, z : (x, r_2, y) \wedge (y, r_3, z) \implies (x, r_1, z)$

- **Example:** x=Tom, y=Quebec, z=Canada, r_2 = "is born in", r_3 = "is located in", r_1 = "is from"
  (x, r_2, y) = Tom is born in Quebec $\wedge$ (y, r_3, z) = Quebec is located in Canada $\implies$ (x, r_1, z) = Tom is from Canada

ref: RotateE[2]

Score Function

There are different flavours of KGE that have been developed over the course of the past few years. What most of them have in common is a score function. The score function measures how distant two nodes in relations to a relation are. As we are setting the stage to introduce the reader to DGL-KE, an open source knowledge graph embedding library, we limit the scope only to those methods that are implemented by DGL-KE and are listed in figure 2.



Models	score function $f(\mathbf{h}, \mathbf{r}, \mathbf{t})$
TransE [2]	$-\|\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|\|_{1/2}$
TransR [10]	$-\|\|M_r\mathbf{h} + \mathbf{r} - \mathbf{M_r t}\|\|_2^2$
DistMult [20]	$\mathbf{h}^\top \mathrm{diag}(\mathbf{r})\mathbf{t}$
ComplEx [16]	$\mathrm{Real}(\mathbf{h}^\top \mathrm{diag}(\mathbf{r})\bar{\mathbf{t}})$
RESCAL [12]	$\mathbf{h}^\top \mathbf{M_r t}$
RotatE [15]	$-\|\|\mathbf{h} \circ \mathbf{r} - \mathbf{t}\|\|^2$

Figure2: A list of score functions for KE papers implemented by DGL-KE

A short explanation of the score functions

Knowledge graphs that are beyond toy examples are always large, high dimensional, and sparse. High dimensionality and sparsity are the results of the amount of information that the KG holds that can be represented with 1-hot or n-hot vectors. The fact that most of the items have no relationship withe one another is another major contributor to sparsity of KG representations. We, therefore desire to project the sparse and high dimensional graph representational vector space onto a lower dimensional dense space. This process is similar to to word embedding and dimensionality reduction used for recommender systems baesd on matrix factorization models. I will provide a detailed account of all the methods in a different post, but here I will provide a short explanation of how projections differ in each paper, what the score functions do, and what consequences the choices have on on relationship inference and computational complexity.

TransE:

TransE is a representative translational distance model that represents entities and relations as vectors in the same semantic space of dimension $\mathbb{R}^\mathrm{d}$, 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 *head*, r is for *relation*, and t is for *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 $h + r \approx t$. For example if $h_1 = emb(\text{" } Ottawa \text{ "}),\ h_2 = emb(\text{" } Berlin \text{ "}), t_1 = emb(\text{" } Canada \text{ "}), t_2 = (\text{" } Germany \text{ "})$, and finally $r = \text{" } CapilatOf \text{ "}$, 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 $f = -\|h + r - t\|_{\frac{1}{2}}$

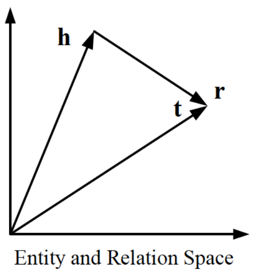


Figure 3: TransE

TransR

TransE cannot cover relationship that are not 1-to-1 as it learns only one aspect of similarity. TransR addresses this issue with separating relationship space from entity space where $h, t \in \mathbb{R}^k$ and $r \in \mathbb{R}^d$. The semantic spaces do not need to be of the same dimension. In the multi-relationship modeling we learn a projection matrix $M \in \mathbb{R}^{k \times d}$ for each relationship that can project an entity to different relationship semantic spaces. Each of these spaces capture a different aspect of an entity that is related to a distinct relationship. In this case a head node h and a tail node t in relation to relationship r is projected into the relationship space using the learned projection matrix M_r as $h_r = hM_r$ and $t_r = tM_r$ respectively.Figure 5 illustrates this projection.

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 embedding for Mary, Tom, and Joe from the fact that they are colleagues but fail to recognize the (not)sibling relationship. Using TransR, we learn projection matrices: $M_{sib},\ M_{clg}$ and M_{vgt} that perform better on learning relationship like (not)sibling. The score function in TransR is similar to TransE and measures euclidean distance between $h + r$ and t, but the distance measure is per relationship space. More formally: $f_r = \|h_r + r - t_r\|_2^2$

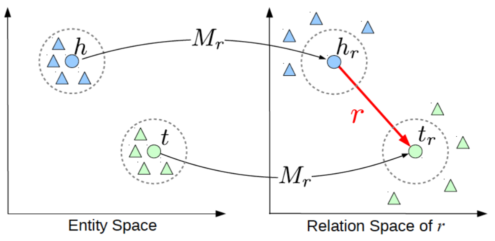


Figure 4: TransR projecting different aspects of an entity to a relationship space.

Another advantage of TransR over TransE is its ability to extract compositional rules. Ability to extract rules has two major benefits: 1) Richer information, and 2) smaller memory space as we can infer some of the rules from others.

Drawbacks

The benefits from more expressive projections in TransR adds to the complexity of the model as well as higher rate of moving data around which has an adverse effect on distributed training. TransE requires $O(d)$ parameters per relation, where d is the dimension of semantic space in TransE which includes both entities and relationships. As TransR projects entities to a relationship space of dimension k, 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 benefits of DGL-KE in making computation of knowledge embedding significantly more efficient.

ref: TransR[5], 7

TransE and its variantes such as TransR are generall called *translational distance models* as they translate the entities and relationships and measure distance in the the target semantic spaces. A second category of KE models is called *semantic matching* that includes models such as RESCAL, DistMulti, and ComplEx.These models make use of a similarity-based scoring function. The first of semantic matching models we explore is RESCAL.

RESCAL

RESCAL is a **bilinear** model that captures latent semantics of a knowledge graph through associate entities with vectors and represents each relation as a matrix that **models pairwise interaction** between entities.

Multiple relations of any order can be represented as tensors. In fact $n-dimensional$ tensors are by definitions representation of multi-dimensional vector spaces. RESCAL, therefore, proposes to capture entities and relationships as multidimensional tensors as illustrated in figure 5.

RESCAL uses semantic web's RDF formation where relationships are modeled as $(subject, predicate, object)$. Tensor $\mathcal{X}$ contains such relationships as $\mathcal{X}_{ijk}$ between i^{th} and j^{th} entities through k^{th} relation. Value of $\mathcal{X}_{ijk}$ is determined as:

$$\mathcal{X}_{ijk} = \begin{cases} 1 & \text{if } (e_i, r_k, e_j) \text{ holds} \\ 0 & \text{if } (e_i, r_k, e_j) \text{ does not hold} \end{cases}$$

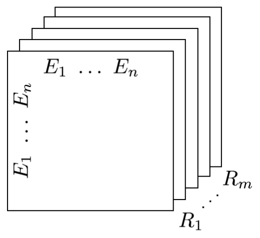


Figure 5: RESCAL captures entities and their relations as multi-dimensional tensor

As entity relationship tensors tends to be sparse, the authors of RESCAL, propose a dyadic decomposition to capture the inherent structure of the relations in the form of a latent vector representation of the entities and an asymmetric square matrix that captures the relationships. More formally each slice of $\mathcal{X}_k$ is decomposed as a rank$-r$ factorization:

$$\mathcal{X}_k \approx AR_k\mathbf{A}^\top, \text{ for } k = 1, \dots, m$$

where A is an $n \times r$ matrix of latent-component representation of entities and asymmetrical $r \times r$ square matrix R_k that models interaction for k_th predicate component in $\mathcal{X}$. To make sense of it all, let's take a look at an example:

$Entities = \{\text{Mary}:0, \text{Tom}:1, \text{Joe}:2\}$
$Relationships = \{\text{sibling, colleague}\}$
$Relation_{k=0}^{sibling}$: Mary and Tom are siblings but Joe is not their sibling.
$Relations_{k=1}^{colleague}$: Mary,Tom, and Joe are colleagues

relationship matrices will model: $\mathcal{X}_k = \begin{bmatrix} Mary & Tom & Joe \\ Tom & Joe & Mary \\ Joe & Mary & Tom \end{bmatrix}$

$\mathcal{X}_{0:sibling} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$

$\mathcal{X}_{1:colleague} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

Note that even in such a small knowledge graph where two of the three entities have even a symmetrical relationship, matrix $\mathcal{X}_k$ is sparse and asymmetrical. Obviously colleague relationship in this example is not representative of a real world problem. Even though such relationships can be created they contain no information as probability of occurring is hight. For instance if we are creating a knowledge graph for for registered members of a website is a specific country, we do not model relations like "is countryman of" as it contains little information and has very low entropy.

Next step in RESCAL is decomposing matrices $\mathcal{X}_k$ using a rank_k decomposition as illustrated in figure 6.

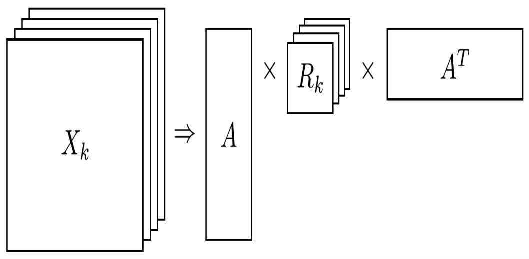


Figure 6: Each of the k slices of martix $\mathcal{X}$ is factorized to its k-rank components in form of a $n \times r$ entity-latent component and an asymmetric $r \times r$ that specifies interactions of entity-latent components per relation.

A and R_k are computed through solving an optimization problem that is correlated to minimizing the distance between $\mathcal{X}_k$ and $AR_k\mathbf{A}^\top$.

Now that the structural decomposition of entities and their relationships are modeled we need to create a score function that can predict existence of relationship for those entities we lack their mutual connection information.

The score function $f_r(h, t)$ for $h, t \in \mathbb{R}^d$, where h and t are representation of *head* and *tail* entities, captures pairwise interactions between entities in h and t through relationship matrix M_r that is the collection of all individual R_k matrices and is of dimension $d \times d$.

$$f_r(h, t) = \mathbf{h}^\top M_r t = \sum_{i=0}^{d-1} \sum_{j=0}^{d-1} [M_r]_{ij} . [h]_i . [t]_j$$

Figure 7 illustrates computation of the the score for RESCAL method.

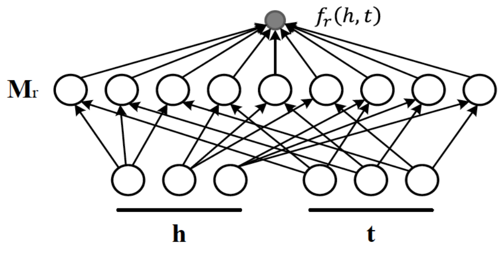


Figure 7: RESCAL

Score function f requires $O(d^2)$ parameters per relation.

Ref: 6,7

DistMult

If we want to speed up the computation of RESCAL and limit the relationships only to symmetric relations, then we can take advantage of the proposal put forth by DistMult[8], which simplifies RESCAL by restricting M_r from a general asymmetric $r \times r$ matrix to a diagonal square matrix and reducing the number of parameters per relation to $O(d)$. DistMulti introduces vector embedding $r \in \mathcal{R}^d$.the score function for DistMult where $M_r = diag(r)$ is computed as:

$$f_r(h,t) = \mathbf{h}^\top diag(r) t = \sum_{i=0}^{d-1} [r]_i . [h]_i . [t]_i$$

Figure 8 illustrates how DistMulti computes the score by capturing the pairwise interaction only along the same dimensions of components of h and t.

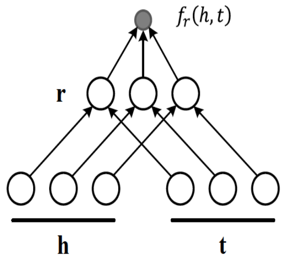


Figure 8: DistMulti

A basic refresher on linear algebra

$if\ A = [a_{ij}]_{m\times n} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}_{m\times n}$ and $B = [b_{ij}]_{n\times k} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{bmatrix}_{n\times k}$ $then\ C = [c_{mk}]_{m\times k}\ such\ that\ c_{mk} = \sum_{p=1}^{k} a_{mp}b_{pk}\ thus:$ We know that a $C_{m\times k} = \begin{bmatrix} a_{11}b_{11} + \dots + a_{1n}b_{n1} & a_{11}b_{12} + \dots + a_{1n}b_{n2} & \dots & a_{11}b_{1k} + \dots + a_{1n}b_{nk} \\ a_{21}b_{11} + \dots + a_{2n}b_{n1} & a_{21}b_{12} + \dots + a_{2n}b_{n2} & \dots & a_{21}b_{1k} + \dots + a_{2n}b_{nk} \\ \vdots & \vdots & \ddots & \dots \\ a_{m1}b_{11} + \dots + a_{mn}b_{n1} & a_{m1}b_{12} + \dots + a_{mn}b_{n2} & \dots & a_{m1}b_{1k} + \dots + a_{mn}b_{nk} \end{bmatrix}_{n\times k}$
diagonal matrix is a matrix in which all non diagonal elements, $(i \neq j)$, are zero. This reduces complexity of matrix multiplication as for diagonal matrix multiplication for diagonal matrices $A_{m\times n}$ and $B_{n\times k}$, $C = AB = [c_{mk}]_{m\times k}$ where

$$c_{mk} = \begin{cases} 0 & \text{for } m \neq k \\ a_m b_k & \text{for } m = k \end{cases}$$

This is basically multiplying to numbers a_{ii} and b_{ii} to get the value for the corresponding diagonal element on C. This complexity reduction is the reason that whenever possible we would like to reduce matrices to diagonal matrices.

ComplEx

In order to model a KB effectively models need to be able to identify most common relationship patters as laid out earlier in this blog. relations can be reflexive/irreflexive, symmetric/antisymmetric, and transitive/intransitive. We have also seen two classes of semantic matching models, RESCAL and DistMulti. RESCAL is expressive but has an exponential complexity, whiled DistMulti has linear complexity but is limited to symmetric relations. An ideal model needs to keep linear complexity while being able to capture antisymmetric relations. Let us go back to what is good at DistMulti. It is using a rank-decomposition based on a diagonal matrix. We know that dot product of embedding scale well and handles symmetry, reflexity, and irreflexivity effectively. Matrix factorization methods have been very successful in recommender systems based on some variation of MF. MF works based on factorizing a relation matrix to dot product of lower dimensional matrices $\mathbf{UV}^\top$ where $\mathbf{UV} \in \mathbb{R}^{n\times K}$. The underlying assumption here is that the same entity would be taken to be different depending on whether it appears as a subject or an object in a relationship. For instance "Quebec" in "Quebec is located in Canada" and "Joe is from Quebec" appears as subject and object respectively. In many link prediction tasks the same entity can assume both roles as we perform graph embedding through adjacency matrix computation. Dealing with antisymmetric relationships, consequently, has resulted in an explosion of parameters and increased complexity and memory requirements.

The goal ComplEx is set to achieve is perform embedding while reducing the number of required parameters, scale well, and capture antisymmetric relations. One essential strategy is to compute a joint representation for the entities regardless of their role as subject or object and perform dot product on those embeddings.

Such embedding cannot be achieved in the real vector spaces, so ComplEx authors propose complex embedding.

But first a quick reminder about complex vectors.

Complex Vector Space

As 1 is the unit for real numbers, $i = \sqrt{-1}$ is the **imaginary unit** of complex numbers. Each complex number has two parts, a real and an imaginary part as is represented as $c = a + bi \in \mathbb{C}$. As expected the complex plane has a horizontal and a vertical axis. The horizontal is the real and the vertical represents the imaginary part of a number in the same way that x and y are represented on Cartesian plane. An n-dimensional complex vector $\mathcal{V} \in \mathbb{C}^n$ is a vector whose elements $v_i \in \mathbb{C}$ are complex numbers.

Example: $V_1 = \begin{bmatrix} 2+3i \\ 1+5i \end{bmatrix}$ and $V_2 = \begin{bmatrix} 2+3i \\ 1+5i \\ 3 \end{bmatrix}$ are in $\mathbb{C}^2$ and $\mathbb{C}^3$ respectively.

$\mathbb{R} \subset \mathbb{C}$ and $\mathbb{R}^n \subset \mathbb{C}^n$. Basically a real number is a complex number whose imaginary part gas a coefficient of zero. **Complex Conjugate** The conjugate of complex number $z = a + bi$ is denoted by $\bar{z}$ and is given by $\bar{z} = a - bi$.

Example: $\bar{V}_1 = \begin{bmatrix} 2-3i \\ 1-5i \end{bmatrix}$ and $\bar{V}_2 = \begin{bmatrix} 2-3i \\ 1-5i \\ 3 \end{bmatrix}$ are in $\mathbb{C}^2$ and $\mathbb{C}^3$ respectively.

Conjugate Transpose The conjugate transpose of a complex matrix $\mathcal{A}$, is denoted as $\mathcal{A}^*$ and is given by $\mathcal{A}^* = \bar{\mathcal{A}}^\top$ where elements of $\bar{\mathcal{A}}$ are complex conjugates of $\mathcal{A}$.

Example: $V_1^* = \begin{bmatrix} 2-3i & 1-5i \end{bmatrix}$ and $V_2^* = \begin{bmatrix} 2-3i & 1-5i & 3 \end{bmatrix}$ are in $\mathbb{C}^2$ and $\mathbb{C}^3$ respectively.

Complex dot product. aka Hermitian inner product if $\mathbf{u}$ and $\mathbf{c}$ are complex vectors, then their inner product is defined as $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^* \mathbf{v}$.

Example:
$u = \begin{bmatrix} 2+3i \\ 1+5i \end{bmatrix}$ and $v = \begin{bmatrix} 1+i \\ 2+2i \end{bmatrix}$ are in $\mathbb{C}^2$ and $\mathbb{C}^3$ respectively. then $u^* = \begin{bmatrix} 2-3i & 1-5i \end{bmatrix}$ and $\langle u, v \rangle = u^* v = \begin{bmatrix} 2-3i & 1-5i \end{bmatrix} \begin{bmatrix} 1+i \\ 2+2i \end{bmatrix} = (2-3i)(1+i) + (1-5i)$

Definition: A complex matrix A us **unitary** when $A^{-1} = A^*$

Example: $A = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$

Theorm: An $n \times n$ complex matrix A is unitary $\iff$ its rows or columns form an orthanormal set in C^n

Definition: A square matrix A is **Hermitian** when $A = A^*$

Example:$A = \begin{bmatrix} a_1 & b_1 + b_2 i \\ b_1 + b_2 i & d+1 \end{bmatrix}$

Theorm: Matrix A is Hermitian $\iff$: 1. $a_{ii} \in \mathbb{R}$ 2. $a_{ij} iscomplexconjugateof a_{ji}$

Theorm: If A is a Hermirian matrix, then its eigenvalues are real numbers.

Theorm: Hermitian matrices are **unitarity diagonizable**.

Definitions: A squared matrix A is unitarily diagonizable when there exists a unitary matrix P such that $P^{-1}AP$.

Diagonizability can be extended to a larger class of matrices, called normal matrices.

Definition: A square complex matrix A is called **normal** when it commutes with its conjugate transpose. $AA^* = A^*A$.

Theorm: A complex matrix A is **normal** $\iff$ A is **diagonizable**.

This theorm plays a crucial role in ComplEx paper.

ref: https://www.cengage.com/resource*uploads/downloads/1133110878*339554.pdf

Eigen decomposition for entity embedding

The matrix decomposition methods have a long history in machine learning are used. Using embedding based decomposition in the form of $X = EWE^{-1}$ for square symmetric matrices can be represented as eigen decomposition $X = Q\Lambda Q^{-1}$ where Q is orthogonal ($\vDash Q^{-1} = Q^\top$) and $\Lambda = diag(\lambda)$ and λ_i is an eigenvector of X.

As ComplEx targets to learn antisymmetric relations, and eigen decomposition for asymmetric matrices does not exist in real space, the authors extend the embedding representation to complex numbers, where they can factorize complex matrices and benefit from efficient scaling and distribution of matrix multiplication while being able to capture antisymmetric relations. This asymmetry is resulted from the fact that dot product of complex matrices involves conjugate transpose.

We are not done yet. Do you remember in RESCAL the number of parameters was $O(d^2)$ and DistMulti reduce that to a linear relation of $O(d)$ by limiting matrix M_r to be diagonal?. Here even with complex eigenvectors $E \in C^{n \times n}$, inversion of E in $X = EWE^*$ explodes the number of parameters. As a result we need to find a solutions in which W is a diagonal matrix, and $E = E^*$, and X is asymmetric, so that we 1) computation is minimized, 2) there is no need to compute inverse of E, and 3) antisymmetric relations can be captures. We have already seen the solution in the complex vector space section. The paper does construct the decomposition in a normal space, a vector space composed of complex normal vectors.

The Score Function

A relation between two entities can be modeled as a sign function, meaning that if there is a relation between subject and object, then the outcome is 1, otherwise it is 0. More formally, $Y_{so} \in \{-1, 1\}$. the probability of of a relation to exist then is given by sigmoid function: $P(Y_{so} = 1) = \sigma($\X$_{(}so))$.

This probability score requires X to be real, while EWE^* includes both real and imaginary components. We can simply project the decomposition to the real space so that $X = Re(EWE^*)$. the score function of ComlEx, therefore is given by:

$$f_r(h, t) = Re(h^\top diag(r)\bar{t}) = Re(\sum_{i=0}^{d-1} [r]_i . [h]_i . [\bar{t}]_i)$$

and since there are no nested loops, the number of parameters is linear and is given by $O(d)$.

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 5.TransR
5. RESCAL
6. Survey paper
7. DistMult

`y - (y^2 - k) / 2y`

i^2 ?

$$if\ A = [a_{ij}]_{m\times n} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}_{m\times n} \text{ and } B = [b_{ij}]_{n\times k} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{bmatrix}_{n\times k} then\ C = [c_{mk}]_{m\times k}\ such\ that\ c_{mk} = \sum_{p=1}^{k} a_{mp} b_{pk}$$

$$f$$