CS224W Homework 2

yeeboxie

March 6, 2025

1 Node Embeddings with TransE [21 points]

While many real world systems are effectively modeled as graphs, graphs can be a cumbersome format for certain downstream applications, such as machine learning models. It is often useful to represent each node of a graph as a vector in a continuous low dimensional space. The goal is to preserve information about the structure of the graph in the vectors assigned to each node. For instance, the spectral embedding preserved structure in the sense that nodes connected by an edge were usually close together in the (one-dimensional) embedding x.

Multi-relational graphs are graphs with multiple types of edges. They are incredibly useful for representing structured information, as in knowledge graphs. There may be one node representing "Washington, DC" and another representing "United States", and an edge between them with the type "Is capital of". In order to create an embedding for this type of graph, we need to capture information about not just which edges exist, but what the types of those edges are. In this problem, we will explore a particular algorithm designed to learn node embeddings for multi-relational graphs.

The algorithm we will look at is TransE.[1]

We will first introduce some notation used in the paper describing this algorithm. We'll let a multi-relational graph $G = (E, S, L)$ consist of the set of *entities* E (i.e., nodes), a set of edges S, and a set of possible relationships L. The set S consists of triples (h, l, t), where $h \in E$ is the *head* or source-node, $l \in L$ is the relationship, and $t \in E$ is the *tail* or destination-node. As a node embedding, TransE tries to learn embeddings of each entity $e \in E$ into $\mathbb{R}^k$ (k-dimensional vectors), which we will notate by $\mathbf{e}$. The main innovation of TransE is that each relationship ℓ is also embedded as a vector $\ell \in \mathbb{R}^k$, such that the difference between the embeddings of entities linked via the relationship ℓ is approximately ℓ. That is, if $(h, \ell, t) \in S$, TransE tries to ensure that $\mathbf{h} + \boldsymbol{\ell} \approx \mathbf{t}$. Simultanesouly, TransE tries to make sure that $\mathbf{h} + \boldsymbol{\ell} \not\approx \mathbf{t}$ if the edge (h, ℓ, t) does not exist.

Note on notation: we will use unbolded letters e, ℓ, etc. to denote the entities and

[1] See the 2013 NeurIPS paper by Bordes et al: https://papers.nips.cc/paper/5071-translating-embeddings-formodeling-multi-relational-data.pdf

relationships in the graph, and bold letters $\mathbf{e}, \boldsymbol{\ell}$, etc., to denote their corresponding embeddings. TransE accomplishes this by minimizing the following loss:

$$\mathcal{L} = \sum_{(h,\ell,t)\in S} \left(\sum_{(h',\ell,t')\in S'_{(h,\ell,t)}} \left[\gamma + d(\mathbf{h}+\boldsymbol{\ell}, \mathbf{t}) - d\left(\mathbf{h}'+\boldsymbol{\ell}, \mathbf{t}'\right)\right]_+ \right) \tag{1}$$

Here (h', ℓ, t') are "corrupted" triplets, chosen from the set $S'_{(h,\ell,t)}$ of corruptions of (h, ℓ, t), which are all triples where either h or t (but not both) is replaced by a random entity, and ℓ remains the same as the one in the original triplets.

$$S'_{(h,\ell,t)} = \left\{\left(h', \ell, t\right) \mid h' \in E\right\} \cup \left\{\left(h, \ell, t'\right) \mid t' \in E\right\}$$

Additionally, $\gamma > 0$ is a fixed scalar called the *margin*, the function $d(\cdot, \cdot)$ is the Euclidean distance, and $[\cdot]_+$ is the positive part function (defined as $\max(0, \cdot)$). Finally, TransE restricts **all the entity embeddings to have length** $1 : \|\mathbf{e}\|_2 = 1$ **for every** $e \in E$**.**

For reference, here is the TransE algorithm, as described in the original paper on page 3:

Algorithm 1 Learning TransE

input Training set $S = \{(h, \ell, t)\}$, entities and rel. sets E and L, margin γ, embeddings dim. k.

initialize $\boldsymbol{\ell} \leftarrow$ uniform$(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each $\ell \in L$

  $\boldsymbol{\ell} \leftarrow \boldsymbol{\ell} / \|\boldsymbol{\ell}\|$ for each $\ell \in L$

  $\mathbf{e} \leftarrow$ uniform$(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each entity $e \in E$

loop

  $\mathbf{e} \leftarrow \mathbf{e} / \|\mathbf{e}\|$ for each entity $e \in E$

  $S_{batch} \leftarrow$sample(S, b) // sample a minibatch of size b

  $T_{batch} \leftarrow \emptyset$ // initialize the set of pairs of triplets

  for $(h, \ell, t) \in S_{batch}$ **do**

    $(h', \ell, t') \leftarrow$sample$(S'_{(h,\ell,t)})$ // sample a corrupted triplet

    $T_{batch} \leftarrow T_{batch} \cup \left\{\left((h, \ell, t), (h', \ell, t')\right)\right\}$

  end for

  Update embeddings w.r.t. $\sum_{\left((h,\ell,t),(h',\ell,t')\right)\in T_{batch}} \nabla\left[\gamma + d(\boldsymbol{h}+\boldsymbol{\ell}, \boldsymbol{t}) - d(\boldsymbol{h'}+\boldsymbol{\ell}, \boldsymbol{t'})\right]_+$

end loop

1.1 Simplified Objective [3 points]

Say we were intent on using a simpler loss function. Our objective function (1) includes a term maximizing the distance between $\mathbf{h}' + \boldsymbol{\ell}$ and $\mathbf{t}'$. If we instead simplified the objective, and just tried to minimize

$$\mathcal{L}_{\text{simple}} = \sum_{(h,\ell,t)\in S} d(\mathbf{h}+\boldsymbol{\ell}, \mathbf{t}), \tag{2}$$

we would obtain a useless embedding. Give an example of a simple graph and corresponding embeddings which will minimize the new objective function (2) all the way to zero, but still give a completely useless embedding.
Hint: Your graph should be non-trivial, i.e., it should include at least two nodes and at least one edge. Assume the embeddings are in 2 dimensions, i.e., $k = 2$. What happens if $\boldsymbol{\ell} = \mathbf{0}$?

★ **Solution** ★

We can embed all the entities with $(0, 1)$ and get the minimum loss function value of 0 although this kind of embeding is useless.

1.2 Utility of γ [5 points]

We are interested in understanding what the margin term γ accomplishes. If we removed the margin term γ from our loss, and instead optimized

$$\mathcal{L}_{\text{no margin}} = \sum_{(h,\ell,t)\in S} \sum_{(h',\ell t')\in S'_{(h,\ell,t)}} \left[d(\mathbf{h}+\boldsymbol{\ell},\mathbf{t}) - d\left(\mathbf{h}'+\boldsymbol{\ell},\mathbf{t}'\right)\right]_+, \tag{3}$$

it turns out that we would again obtain a useless embedding. Give an example of a simple graph and corresponding embeddings which will minimize the new objective function (3) all the way to zero, but still give a completely useless embedding. By useless, we mean that in your example, you cannot tell just from the embeddings whether two nodes are linked by a particular relation (Note: your graph should be non-trivial, i.e., it should include at least two nodes and at least one edge. Assume the embeddings are in 2 dimensions, i.e., $k = 2$.)

★ **Solution** ★

We can embed all the entities with $(0, 1)$ and get the minimum loss function value of 0 although this kind of embeding is useless.

Without γ, the loss function does not enforce a sufficient gap between positive and negative pairs, allowing the embeddings to be degenerate. Even though the loss function reaches zero, the embeddings may not encode any meaningful relational information, making them "useless" in practice.

1.3 Normalizing the embeddings [5 points]

Recall that TransE normalizes every entity embedding to have unit length (see line 5 of the algorithm). The quality of our embeddings would be much worse if we did not have this step. To understand why, imagine running the algorithm with line 5 omitted. What could the algorithm do to trivially minimize the loss in this case? What would the embeddings it generates look like?

★ Solution ★

Without the normalization step, the embeddings could look like the following:

- Arbitrary scaling: The embeddings could become arbitrarily large. For instance, instead of embeddings like $h = (0.5, 0.3)$, you could end up with h=(10,6). While this might still respect the relative distances (i.e., $d(h, t)$ might remain the same), the actual embeddings would be unnecessarily large, leading to poor generalization and lack of structure in the learned representation.
- Identical embeddings: The embeddings of all entities might collapse into a single point, e.g., $h = t = h' = t' = (0, 0)$. This would trivially minimize the loss by making all distances zero, but it would obviously result in useless embeddings for distinguishing relationships.

1.4 Expressiveness of TransE embeddings [8 points]

Give an example of a simple graph for which no perfect embedding exists, i.e., no embedding perfectly satisfies $\mathbf{u} + \boldsymbol{\ell} = \mathbf{v}$ for all $(u, \ell, v) \in S$ and $\mathbf{u} + \boldsymbol{\ell} \neq \mathbf{v}$ for $(u, \ell, v) \notin S$, for any choice of entity embeddings ($\mathbf{e}$ for $e \in E$) and relationship embeddings ($\boldsymbol{\ell}$ for $\ell \in L$). Explain why this graph has no perfect embedding in this system, and what that means about the expressiveness of TransE embeddings. As before, assume the embeddings are in 2 dimensions ($k = 2$).
Hint: By expressiveness of TransE embeddings, we want you to talk about which type of relationships TransE can/cannot model with an example. (Note that the condition for this question is slightly different from that for Question 2.1 and what we ask you to answer is different as well).

★ Solution ★ TransE struggles with modeling **1-n (one-to-many)** and **symmetric relations** due to its simplistic approach.

1-n (One-to-Many) Relations

In a one-to-many relation, one entity (the head) is connected to multiple entities (the tails). For example, consider the relation "author of":

$$\text{"J.K. Rowling"} \xrightarrow{\text{author of}} \text{"Harry Potter"}$$

$$\text{"J.K. Rowling"} \xrightarrow{\text{author of}} \text{"Fantastic Beasts"}$$

For TransE to model this correctly, it needs to learn that the vector for "J.K. Rowling" plus the relation vector for "author of" should match different tail vectors, such as "Harry Potter" and "Fantastic Beasts." However, because TransE represents relations as translations, it assumes that the relation vector remains constant. This is problematic because one relation cannot simultaneously map to multiple different

tail entities in the same way, making it difficult for TransE to handle one-to-many relations properly.

Symmetric Relations

For symmetric relations, the order of the entities doesn't matter. For example, consider the relation "is friend of":

$$\text{“Alice”} \xrightarrow{\text{is friend of}} \text{“Bob”}$$

$$\text{“Bob”} \xrightarrow{\text{is friend of}} \text{“Alice”}$$

In this case, the relationship between Alice and Bob is bidirectional, meaning if the triplet $(\text{Alice}, \text{is friend of}, \text{Bob})$ holds true, so does $(\text{Bob}, \text{is friend of}, \text{Alice})$.

However, TransE models the relation r as a translation, so it expects that $h + r \approx t$. This implies that for a symmetric relation, the vectors for h and t should be identical, or the relation vector r should be zero (i.e., no translation), which contradicts the assumption of a non-zero relation vector for the translation. Thus, TransE struggles to model symmetric relations effectively because it cannot capture the bidirectionality properly.

Summary

- **1-n relations**: TransE cannot handle these well because it assumes a fixed translation for a relation, which fails when one entity is connected to multiple others.
- **Symmetric relations**: TransE cannot handle these effectively because it assumes a directional translation, which conflicts with the bidirectionality inherent in symmetric relations.

These limitations highlight why more advanced models like TransH or TransR, which introduce more flexibility in representing relations, are often preferred for handling such cases.

2 Expressive Power of Knowledge Graph Embeddings [10 points]

TransE is a common method for learning representations of entities and relations in a knowledge graph. Given a triplet (h, ℓ, t), where entities embedded as h and t are related by a relation embedded as ℓ, TransE trains entity and relation embeddings to make $h + \ell$ close to t. There are some common patterns that relations form:

- Symmetry: A is married to B, and B is married to A.
- Inverse: A is teacher of B, and B is student of A. Note that teacher and student are 2 different relations and have their own embeddings.
- Composition: A is son of B; C is sister of B, then C is aunt of A. Again note that son, sister, and aunt are 3 different relations and have their own embeddings.

2.1 TransE Modeling [3 points]

For each of the above relational patterns, can TransE model it perfectly, such that $h + \ell = t$ for all relations? Explain why or why not. Note that here $\mathbf{0}$ embeddings for relation are undesirable since that means two entities related by that relation are identical and not distinguishable.

★ Solution ★

- **Symmetry**: TransE cannot model symmetric relations perfectly. For a symmetric relation ℓ, both (h, ℓ, t) and (t, ℓ, h) must hold, implying $h + \ell \approx t$ and $t + \ell \approx h$. Adding these equations gives $2\ell \approx 0$, forcing $\ell \approx 0$. However, a zero relation embedding is undesirable, as it collapses h and t into the same vector, making them indistinguishable. Thus, symmetry cannot be modeled without violating the non-zero constraint.
- **Inverse**: TransE can model inverse relations. If ℓ_1 and ℓ_2 are inverses (e.g., "teacher" and "student"), then (h, ℓ_1, t) implies $h+\ell_1 \approx t$ and (t, ℓ_2, h) implies $t+\ell_2 \approx h$. Substituting the first into the second gives $\ell_1+\ell_2 \approx 0$, meaning $\ell_2 \approx \neg\ell_1$. This allows distinct non-zero embeddings for ℓ_1 and ℓ_2 while preserving invertibility.
- **Composition** TransE can model compositional relations. Suppose (h, ℓ_1, t_1) and (t_1, ℓ_2, t) imply (h, ℓ_3, t). TransE enforces $h+\ell_1 \approx t_1$ and $t_1+\ell_2 \approx t$,leading to $h+\ell_1+\ell_2 \approx t$. If $\ell_3 = \ell_1+\ell_2$, the composition holds. For example, if "son" (ℓ_1) and "sister" (ℓ_2) compose to "aunt" (ℓ_3), TransE can learn $\ell_3 \approx \ell_2 - \ell_1$ (since $t_1 = h+\ell_1$ and $t = t_1+\ell_2 = h+\ell_1+\ell_2$). Thus, composition is achievable through linear combinations of relation vectors.

Summary:

1. Symmetry: No (requires zero relation embedding).
2. Inverse: Yes (via negation).
3. Composition: Yes (via additive relation embeddings).

2.2 RotatE Modeling [3 points]

Consider a new model, RotatE. Instead of training embeddings such that $h + \ell \approx t$, we train embeddings such that $h \circ \ell \approx t$. Here $\circ$ means rotation. You can think of h as a vector of dimension $2d$, representing d 2D points. ℓ is a d-dimensional vector specifying rotation angles. When applying $\circ$, For all $i \in 0 \ldots d-1, (h_{2i}, h_{2i+1})$ is rotated clockwise by l_i. Similar to TransE, the entity embeddings are also normalized to L2 norm 1. Can RotatE model the above 3 relation patterns perfectly? Why or why not?

★ Solution ★

RotatE can model all three relational patterns (symmetry, inverse, and composition) **perfectly**, unlike TransE. Here's why:

1. Symmetry

Example: "A is married to B" and "B is married to A."

- **Mechanism**: A symmetric relation requires $h \circ \ell \approx t$ and $t \circ \ell \approx h$.
- **Solution**: RotatE uses a rotation angle $\ell = \pi$ (180°). Rotating h by π gives t, and rotating t by π recovers h:

$$h \circ \pi \approx t \quad \text{and} \quad t \circ \pi \approx h.$$

  This satisfies symmetry without collapsing embeddings (unlike TransE's zero-vector issue).

- **Why it works**: Rotations by π flip the vector direction while preserving its norm (L2=1).

2. Inverse

Example: "A is teacher of B" (ℓ_1) and "B is student of A" (ℓ_2).

- **Mechanism**: Inverse relations require $h \circ \ell_1 \approx t$ and $t \circ \ell_2 \approx h$.

- **Solution**: Set $\ell_2 = -\ell_1$ (negative angle). Rotating t by $-\ell_1$ cancels the original rotation:
$$h \circ \ell_1 \approx t \quad \Rightarrow \quad t \circ (-\ell_1) \approx h.$$
- **Why it works**: Rotations are invertible via angle negation, preserving distinct embeddings for both relations.

3. Composition

Example: "A is son of B" (ℓ_1), "C is sister of B" (ℓ_2), so "C is aunt of A" (ℓ_3).

- **Mechanism**: Composition requires $h \circ \ell_1 \circ \ell_2 \approx t$ for $\ell_3 = \ell_1 + \ell_2$.
- **Solution**: Define ℓ_3 as the sum of angles from ℓ_1 and ℓ_2. For instance:
$$\text{Parent: } \ell_{\text{parent}} = -\ell_{\text{son}}, \quad \text{Aunt: } \ell_{\text{aunt}} = \ell_{\text{sister}} + \ell_{\text{parent}}.$$
This ensures $h \circ \ell_{\text{aunt}} \approx t$ through additive angle composition.
- **Why it works**: Rotations are closed under addition (angles compose linearly), enabling hierarchical relationships.

Key Strengths of RotatE

- **Non-Collapsing Embeddings**: Normalization to L2=1 ensures entities remain distinct after rotation.
- **Rotation Invariance**: Rotations preserve vector norms, avoiding TransE's limitations with symmetric relations.
- **Flexibility**: Angles can represent complex relational patterns (e.g., inverse via negation, composition via addition).

Conclusion: RotatE successfully models symmetry, inverse, and composition by leveraging rotational transformations, which are inherently reversible and composable.

2.3 Failure Cases [4 points]

Give an example of a graph that RotatE cannot model perfectly. Can TransE model this graph perfectly? Assume that relation embeddings cannot be $\mathbf{0}$ in either model.

★ Solution ★

Answer

Consider a graph with the following entities and relation:

- Entities:

$$A = (1, 0, 0, 0) \quad \text{(normalized 4D vector)},$$
$$B = (0, 0, 1, 0) \quad \text{(normalized 4D vector)}.$$

- Relation: ℓ such that (A, ℓ, B) holds.

Analysis for RotatE

- **Rotation Constraints**:
  - RotatE splits A into two 2D components: $(1, 0)$ and $(0, 0)$.
  - Applying rotation angles ℓ_i to these components:

$$\text{First pair: } (1, 0) \xrightarrow{\text{rotate } \ell_1} (\cos \ell_1, \sin \ell_1),$$
$$\text{Second pair: } (0, 0) \xrightarrow{\text{rotate } \ell_2} (0, 0).$$

- **Failure**: The rotated result is $(\cos \ell_1, \sin \ell_1, 0, 0)$. To match $B = (0, 0, 1, 0)$, we need:

$$\cos \ell_1 = 0 \quad \text{and} \quad \sin \ell_1 = 0 \quad \text{(impossible)}.$$

- **Key Issue**: RotatE cannot transfer magnitude between separate 2D planes.

Analysis for TransE

- **Translation**:

$$A + \ell = (1, 0, 0, 0) + (-1, 0, 1, 0) = (0, 0, 1, 0) = B.$$

- **Success**: The relation embedding $\ell = (-1, 0, 1, 0)$ is non-zero and valid. TransE freely redistributes magnitude across dimensions.

Conclusion

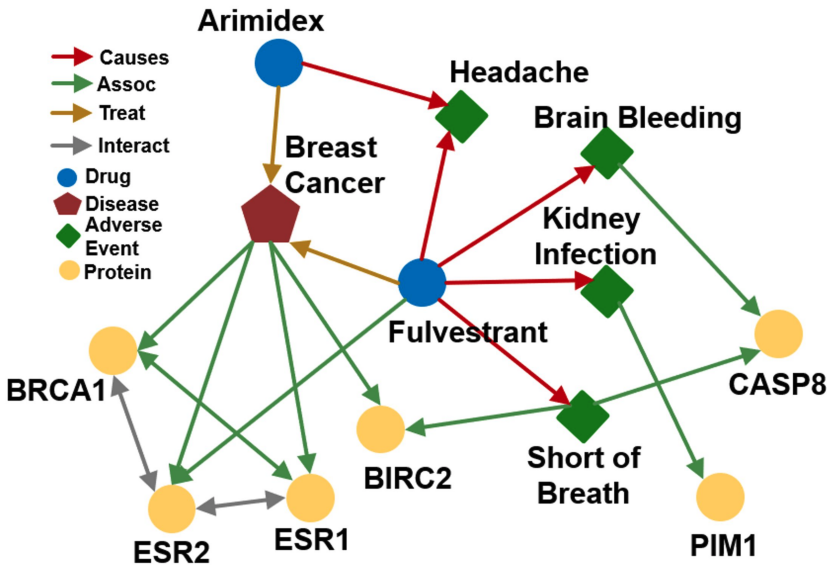
- RotatE fails because rotations are confined to individual 2D planes.
- TransE succeeds because translations operate globally across dimensions.

3 Queries on Knowledge Graphs [14 points]

Knowledge graphs (KGs) can encode a wealth of information about the world. Beyond representing the information using knowledge graphs, we can often derive previously unknown insights about entities and relations in the graphs. In this question, we will explore different approaches for reasoning over knowledge graphs. Recall from that lecture that we are interested in predicting `tail` nodes given (`head`, `relation`). We will use the same formulation throughout this question.

3.1 Path Queries on Complete KGs [3 points]

Consider the biomedicine knowledge graph from lecture. Assume the question of interest is: "What proteins are associated with diseases treated by Arimidex?" Write the question in query form (eg. (e:AnchorEntity, (r:Relation))) and find the answer(s) to the query. Partial credit will be rewarded to correct intermediate steps.



★ Solution ★

$(e : \text{Arimidex}, (r : \text{Treat}, r : \text{Assoc}))$
ESR1

3.2 Conjunctive Queries on Complete KGs [1 point]

Consider the same biomedicine knowledge graph from before. Write a conjunctive query to which BIRC2 is the only answer using drugs as anchor entities. If such a query doesn't exist, provide a one-sentence explanation.

★ Solution ★

What proteins are associated with Shortness of Breath caused by Fluvestrant and associated with Breast Cancer treated with Arimidex?

$((e : \text{Armidex}, (r : \text{Treat}, r : \text{Assoc})), (e : \text{Fulvestrant}, (r : \text{Cause}, r : \text{Assoc})))$

3.3 Incomplete KGs [2 points]

A major issue with direct traversals on knowledge graphs is that they are usually incomplete in reality. One solution is to encode entities, relations, and queries in an embedding space that meaningfully organizes information. We would then be able to impute missing relation links by considering all nearby points of the query embedding as answers to the query. From lecture, we learned that TransE embeddings can be used for this. Can you come up with a way to adopt DistMult embeddings, which uses bilinear modeling, for answering path queries? If yes, describe in one or two sentences what can be modified from the TransE application. If no, provide a one-sentence explanation.

★ Solution ★

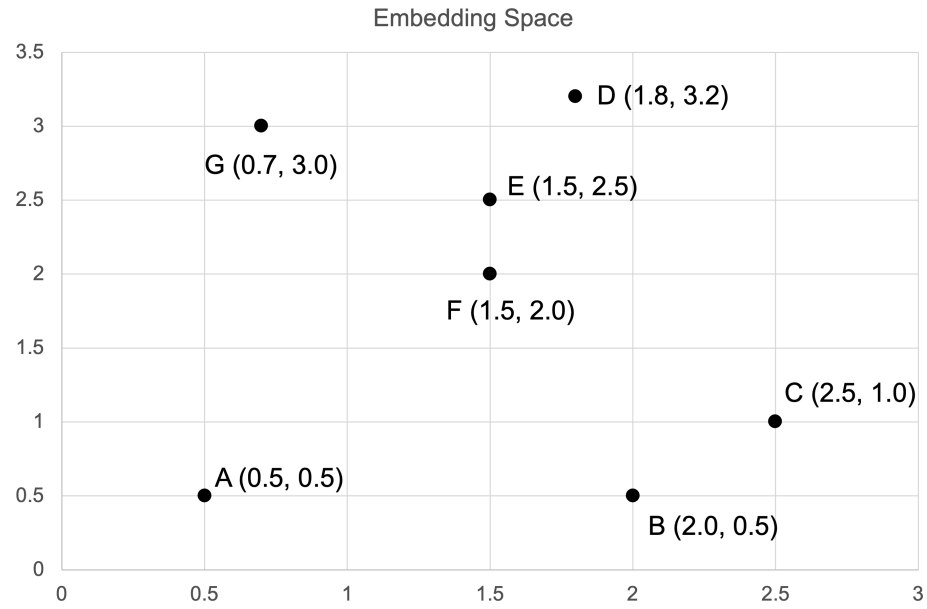
Yes, DistMult can be adapted for path queries by replacing TransE's additive composition of relations with multiplicative composition (element-wise product of relation embeddings) along the path, followed by the bilinear scoring function to compute answer entity scores.

3.4 Query2box [8 points]

Query2box is an effective approach for answering complex conjunctive queries. Consider the following 2-dimensional embedding space. Assume that there are 7 entities $A, B, C, D, E, F, G \in V$, whose embeddings are shown below. There are 3 relations: R_1, R_2, R_3. $R_1 \in R$ shifts the center of a box by $(0.25, 2)$ and increases the width and height of a box by $(0.5, 2)$. R_2 shifts the center of a box by $(1, 0)$ and increases the width and height of a box by $(1, 0)$. R_3 shifts the center of a box by $(-0.75, 1)$ and increases the width and height of a box by $(1.5, 3)$.

Use the Query2box projection operator to find the answers to the conjunctive query: ((e:A, (r:R_1, r:R_2), (e:C, (r:R_3)). Show your work. Partial credit will be rewarded to correct intermediate steps.

Note: Shifting by a negative value means moving towards the left or bottom. Increasing the width and height by an amount means adding that amount in absolute value, not multiplying that amount as a factor. Assume that each path query starts with a box centered at the anchor entity with zero width and height.



★ **Solution** ★

Step 1: Compute boxes for each path query

Path 1: $(e : A, (r : R_1, r : R_2))$

$$\begin{aligned}
&\text{Start: Center } A = (0.5, 0.5),\ \text{Width} = 0,\ \text{Height} = 0 \\
&\text{Apply } R_1 \text{ :Center } = (0.5 + 0.25, 0.5 + 2) = (0.75, 2.5),\ \text{Width} = 0.5,\ \text{Height} = 2 \\
&\text{Apply } R_2 \text{ :Center } = (0.75 + 1, 2.5 + 0) = (1.75, 2.5),\ \text{Width} = 1.5,\ \text{Height} = 2 \\
&\text{Final Box: } X \in [1.75 - 0.75, 1.75 + 0.75] = [1.0, 2.5],\ Y \in [2.5 - 1.0, 2.5 + 1.0] = [1.5, 3.5]
\end{aligned}$$

Path 2: $(e : C, (r : R_3))$

$$\begin{aligned}
&\text{Start: Center } C = (2.5, 1.0),\ \text{Width} = 0,\ \text{Height} = 0 \\
&\text{Apply } R_3 \text{ :Center } = (2.5 - 0.75, 1.0 + 1) = (1.75, 2.0),\ \text{Width} = 1.5,\ \text{Height} = 3 \\
&\text{Final Box: } X \in [1.75 - 0.75, 1.75 + 0.75] = [1.0, 2.5],\ Y \in [2.0 - 1.5, 2.0 + 1.5] = [0.5, 3.5]
\end{aligned}$$

Step 2: Compute intersection of boxes

$$\begin{aligned}
&X\text{-overlap: } [\max(1.0, 1.0), \min(2.5, 2.5)] = [1.0, 2.5] \\
&Y\text{-overlap: } [\max(1.5, 0.5), \min(3.5, 3.5)] = [1.5, 3.5]
\end{aligned}$$

Intersection Box: $X \in [1.0, 2.5],\ Y \in [1.5, 3.5]$

Step 3: Check entities within the intersection box

Entity	Coordinates	X in [1.0, 2.5]	Y in [1.5, 3.5]	In Intersection?
A	(0.5, 0.5)	×	×	×
B	(2.0, 0.5)	✓	×	×
C	(2.5, 1.0)	×	×	×
D	(1.8, 3.2)	✓	✓	✓
E	(1.5, 2.5)	✓	✓	✓
F	(1.5, 2.0)	✓	✓	✓
G	(0.7, 3.0)	×	✓	×

Final Answer

$$\boxed{D,\ E,\ F}$$

4 Subgraph and Order Embeddings [20 points]

In lecture, we demonstrated that subgraph matching can be effectively learned by embedding subgraphs into the order embedding space. The reason is that many properties associated with subgraphs are naturally reflected in the order embedding space.

For this question, we say "graph A is a subgraph of graph B" when there exists a subgraph of B that is graph-isomorphic to graph A. We additionally only consider the induced subgraph setting introduced in lecture, and all the order embeddings are non-negative.

Recall that the order embedding constraint states that: A is a subgraph of B if and only if $z_A[i] \leq z_B[i]$ for all embedding dimension i. For simplicity, we do not consider anchor nodes in this question, and assume that the order embedding z_A is an embedding of graph A.

4.1 Transitivity [4 points]

Show that the subgraph relation is transitive: if graph A is a subgraph of graph B, and graph B is a subgraph of C, then graph A is a subgraph of C. The proof should make use of the subgraph isomorphism definition: if graph A is a subgraph of graph B, then there exists a bijective mapping f that maps all nodes in V_A to a subset of nodes in V_B, such that the subgraph of B induced by $\{f(v)|v \in V_A\}$ is graph-isomorphic to A. (NOTE: You can assume that the composition of two bijective functions is bijective)

★ Solution ★

The subgraph of B induced by $\{f_1(v)|v \in V_A\}$ is graph-isomorphic to A, which could be set to V_B^A

The subgraph of C induced by $\{f_2(v)|v \in V_B^A\}$ is graph-isomorphic to V_B^A, which could be set to V_C^A

V_C^A induced by $\{f_1(f_2(v))|v \in V_A\}$ is graph-isomorphc to A

$f_1(f_2(v))$ is a compostion of two bijective function which is also bijective.

4.2 Anti-symmetry [4 points]

Use the same definition on subgraph isomorphism to show that the subgraph relation is anti-symmetric: if graph A is a subgraph of graph B, and graph B is a subgraph of graph A, then A and B are graph-isomorphic.

Hint: What do these conditions imply about the number of nodes in A and B? How does this relate to graph isomorphism?

★ Solution ★

Step 1: Assume $A \subseteq B$ and $B \subseteq A$.
By the definition of subgraphs:

- $|V_A| \leq |V_B|$ (since $A \subseteq B$).
- $|V_B| \leq |V_A|$ (since $B \subseteq A$).

Thus, $|V_A| = |V_B|$, and similarly $|E_A| = |E_B|$.

Step 2: Construct bijections between V_A and V_B.
Since $A \subseteq B$, there exists an injective vertex map $f : V_A \hookrightarrow V_B$ preserving edges.
Similarly, $B \subseteq A$ implies an injective vertex map $g : V_B \hookrightarrow V_A$.
Because $|V_A| = |V_B|$, both f and g are bijections.

Step 3: Show that f (or g) is a graph isomorphism.

- f preserves edges: If $(u, v) \in E_A$, then $(f(u), f(v)) \in E_B$.
- Conversely, suppose $(f(u), f(v)) \in E_B$. Since $B \subseteq A$, applying g gives $(g(f(u)), g(f(v))) \in E_A$. But $g \circ f = \mathrm{Id}_{V_A}$, so $(u, v) \in E_A$.

Thus, f preserves edges in both directions, making it an isomorphism.
Similarly, g is also an isomorphism.

Conclusion:
If $A \subseteq B$ and $B \subseteq A$, then $A \cong B$. Hence, the subgraph relation is anti-symmetric.

$$\boxed{A \cong B}$$

4.3 Common Subgraphs [3 points]

Consider a 2-dimensional order embedding space. Graph A is embedded into z_A, and graph B is embedded into z_B. Suppose that the order embedding constraint is perfectly preserved in this order embedding space (this is equivalent to assuming that $z_A \preccurlyeq z_B$ holds if and only if A is a subgraph of B). Prove that graph X is a common subgraph of A and B if and only if $z_X \preccurlyeq \min\{z_A, z_B\}$. Here min denotes the element-wise minimum between two embedding vectors.

★ Solution ★

Forward Direction ($\Rightarrow$): Assume X is a common subgraph of A and B. Then:

- $X \subseteq A \implies z_X \preccurlyeq z_A$ (by the order embedding property).
- $X \subseteq B \implies z_X \preccurlyeq z_B$ (by the order embedding property).

Since $\min\{z_A, z_B\}$ is the element-wise minimum of z_A and z_B, it is the greatest lower bound (meet) of z_A and z_B under $\preccurlyeq$. Therefore, $z_X \preccurlyeq \min\{z_A, z_B\}$.

4.4 Order Embedding Constraints [3 points]

Suppose that graphs A, B, C are non-isomorphic graphs that are not subgraphs of each other. We embed them into a 2-dimensional order embedding space. Without loss of generality, suppose that we compare the values of their embeddings in the first dimension (dimension 0) and have $z_A[0] > z_B[0] > z_C[0]$. What does this imply about the relation among $z_A[1], z_B[1], z_C[1]$, assuming that the order embedding constraint is perfectly satisfied?

★ Solution ★

$z_A[1] < z_B[1] < z_C[1]$

4.5 Subgraph Relations [6 points]

In this question, we show that a 2-dimensional order embedding space is not sufficient to perfectly model subgraph relations.

Consider three non-isomorphic graphs: A, B, and C. These graphs are not subgraphs of each other. Let's assume, without loss of generality, that $z_A[0] > z_B[0] > z_C[0]$. Now, imagine we have three other graphs: X, Y, and Z. Each of these is a common subgraph of one or more of the original graphs (A, B, or C). For instance, X could be a common subgraph of both A and B and not C. The task is to construct a scenario where the embeddings of these subgraphs (X, Y, and Z) implicitly satisfy the conditions: $z_X \preccurlyeq z_Y$ and $z_X \preccurlyeq z_Z$. You don't need to provide the specific embedding coordinates. Just describe the relationships between the subgraphs (X, Y, and Z) and the original graphs (A, B, and C). Also, explain why your example meets the given conditions (i.e. $z_X \preccurlyeq z_Y$ and $z_X \preccurlyeq z_Z$).

Note that this condition implies that X is a common subgraph of Y and Z. However, one can construct actual example graphs of A, B, C, X, Y, Z such that X is not a common subgraph of Y and Z. This means that 2-dimensional order embedding space cannot perfectly model subgraph relations. Hence in practice, we use high-dimensional order embedding space. For this question, you do not have to show such example graphs.

★ Solution ★

Step-by-Step Construction

Step 1: Define relationships between original graphs A, B, C:

- A, B, C are non-isomorphic and not subgraphs of each other.
- Embeddings satisfy:

$$z_A[0] > z_B[0] > z_C[0] \quad \text{and} \quad z_A[1] < z_B[1] < z_C[1].$$

Example coordinates:

$$z_A = (5, 1), \quad z_B = (3, 2), \quad z_C = (1, 3).$$

Step 2: Define subgraphs X, Y, Z:

- Y: Common subgraph of A and B:

$$z_Y = (\min(z_A[0], z_B[0]), \min(z_A[1], z_B[1])) = (3, 1).$$

- Z: Common subgraph of B and C:

$$z_Z = (\min(z_B[0], z_C[0]), \min(z_B[1], z_C[1])) = (1, 2).$$

- X: Common subgraph of A and C:

$$z_X = (\min(z_A[0], z_C[0]), \min(z_A[1], z_C[1])) = (1, 1).$$

Step 3: Verify embedding conditions:

$$\begin{aligned} \min(z_Y, z_Z) &= (\min(3, 1), \min(1, 2)) \\ &= (1, 1) = z_X. \end{aligned}$$

Thus, $z_X \preccurlyeq z_Y$ and $z_X \preccurlyeq z_Z$.

Step 4: Show the contradiction:

- X is a subgraph of A and C, but:
- $Y \subseteq A \cap B$, $Z \subseteq B \cap C$.
- X shares no edges/nodes with B, so $X \not\subseteq Y$ and $X \not\subseteq Z$.

Conclusion

2D order embeddings fail because $z_X \preccurlyeq \min(z_Y, z_Z)$ implies $X \subseteq Y$ and $X \subseteq Z$, but this is false.

5 Honor Code [0 points]

(X) I have read and understood Stanford Honor Code before I submitted my work.

Collaboration: Write down the names & SUNetIDs of students you collaborated with on Homework 2 (None if you didn't).

Note: Read our website on our policy about collaboration!