Machine Learning with Graphs
Homework 2

Andreas Casparsen, Josefine Holm, Tobias Kallehauge, Victor Croisfelt

Aalborg University, January 10, 2022

1 GCN (10 points)

1.1 Question 1.1 (1 point)

This question is answered using two different methodologies.

1.1.1 Finding a mapping using the graph

To initially show the simple solution we intend to make a one-to-one mapping between the 2 graphs which involves associating nodes in one graph to nodes in the other. If they are isomorphic it should be possible to draw G2 using the nodes in G1, such that they maintain their neighbors.

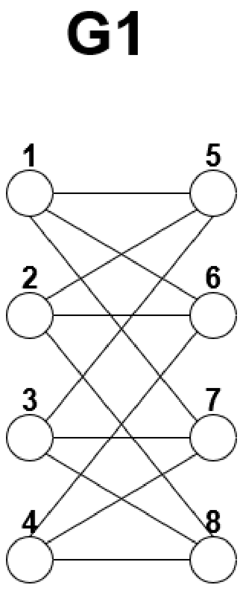

Figure 1: G1 graph

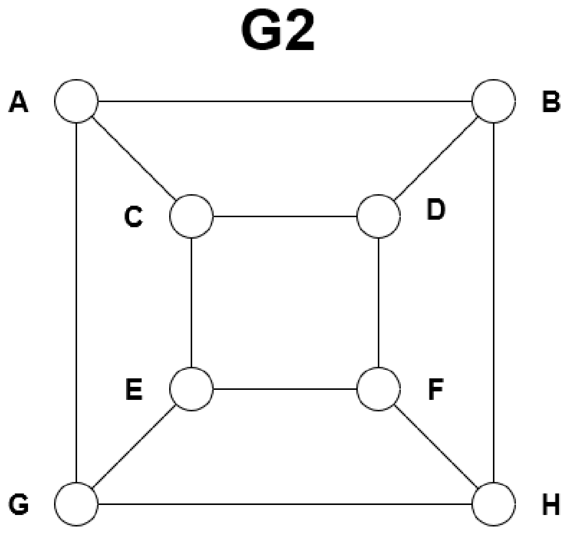

Figure 2: G2 graph

Starting in G2 we set 1=A, and run through the neighbors of G1 and relate these in G2 (5=B, 6=C etc.), which creates the list that defines a relation between nodes in G1 and G2.

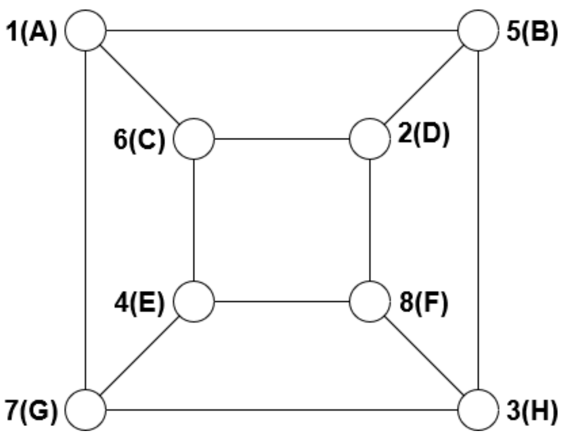

Figure 3: Final graph with insertion of G1 nodes in G2's graph

This can alternatively, more compactly be illustrated as

$$y = \begin{bmatrix} G_1(1) = G_2(A) \\ G_1(2) = G_2(D) \\ G_1(3) = G_2(H) \\ G_1(4) = G_2(E) \\ G_1(5) = G_2(B) \\ G_1(6) = G_2(C) \\ G_1(7) = G_2(G) \\ G_1(8) = G_2(F) \end{bmatrix}. \tag{1}$$

Thus a direct example of how the nodes of G1 could be drawn in G2, which indicates they are isomorphic. The above mapping can also be seen as a the permutation matrix:

$$\mathbf{R} = [\mathbf{e}_1, \mathbf{e}_5, \mathbf{e}_6, \mathbf{e}_2, \mathbf{e}_4, \mathbf{e}_8, \mathbf{e}_7, \mathbf{e}_3] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \tag{2}$$

where $\mathbf{e}_i \in \mathbb{R}^8$ is the i-th canonical vector. Let $\mathbf{A}$ and $\mathbf{B}$ denote the adjacency matrix of the 1st and 2nd graphs, respectively. Then

$$\mathbf{A} = \mathbf{R}\mathbf{B}\mathbf{R}^T. \tag{3}$$

This means that the adjacency matrices are equivalent $\mathbf{A} \sim \mathbf{B}$.

1.1.2 Finding a mapping using the adjacency matrix

In this method, we analyze whether the two graphs are isomorphic or not by relying on their adjacency matrices $\mathbf{A}$ and $\mathbf{B}$. For the first graph (left), its adjacency matrix is

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4}$$

For the second graph (right), we consider the alphabetical order A to G of the nodes, obtaining the following adjacency matrix:

$$\mathbf{B} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}. \tag{5}$$

From the fact that both adjacency matrices are real symmetric matrices, they are diagonalizable, meaning that:

$$\mathbf{Q}_A^{\mathrm{T}} \mathbf{A} \mathbf{Q}_A = \mathbf{\Lambda}_A \tag{6}$$

$$\mathbf{Q}_B^{\mathrm{T}} \mathbf{B} \mathbf{Q}_B = \mathbf{\Lambda}_B \tag{7}$$

where $\mathbf{Q}_i$ is an orthonormal matrix containing normalized eigenvectors and $\mathbf{\Lambda}_i$ is a diagonal matrix with the respective eigenvalues. In fact, we can interpret the diagonalization method as a process of changing basis from one in which the matrix is given by its canonical coordinates (standard basis), as $\mathbf{A}$, to other in which $\mathbf{A}$ can be written as a diagonal matrix. The basis of the last one is defined by the normalized eigenvectors $\mathbf{Q}_A$ (a very special basis). Moreover, given the fact that $\mathrm{rank}(\mathbf{A}) = \mathrm{rank}(\mathbf{B}) = 8$, we know that $\mathbf{A} \sim \mathbf{B}$ are equivalent, meaning that we can write $\mathbf{B}$ by performing elementary transformations over $\mathbf{A}$: $\mathbf{B} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}$ for a nonsingular matrix $\mathbf{P}$. We want to find $\mathbf{P}$. If we want to write $\mathbf{A}$ from the basis $\mathbf{Q}_A$ to $\mathbf{Q}_B$, we need to perform:

$$[\mathbf{\Lambda}_A]_B = \mathbf{Q}_B [\mathbf{\Lambda}_A]_A \mathbf{Q}_B^T \tag{8}$$

$$[\mathbf{\Lambda}_A]_B = \mathbf{Q}_B \mathbf{Q}_A^{\mathrm{T}} \mathbf{A} \mathbf{Q}_A \mathbf{Q}_B^T \tag{9}$$

by calling $\mathbf{P} = \mathbf{Q}_A \mathbf{Q}_B^T$, we have that

$$[\mathbf{\Lambda}_A]_B = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}, \tag{10}$$

and therefore $\mathbf{B} = [\mathbf{\Lambda}_A]_B$. In this way, we have found a way to connect the adjacency matrix $\mathbf{A}$ from graph 1 to graph $\mathbf{B}$.

1.2 Question 1.2 (3 points)

Figure 4 gives an example of two graphs so as to show what is the importance of the aggregation function for the expressiveness of a GCN. We want to evaluate the node embedding of node v given its neighborhood $\mathcal{N}(v)$ and an aggregation function from the set $\{\mathrm{mean}, \mathrm{max}, \mathrm{sum}\}$. Without loss of generality, we assume that blue nodes initialize their node embedding as $h_b^{(0)} = 0$, whereas red nodes $h_r^{(0)} = 1$, $\forall b, r \in \mathcal{N}(v)$ – this is valid for both graphs. In the following, we carry out the aggregation step for each one of the aggregation functions and considering both graphs.

- **Mean**: The mean aggregation function is: $\mathrm{mean} = \sum_{u \in \mathcal{N}(v)} |\mathcal{N}(v)|^{-1} h_u^{(k-1)}$. For $k = 1$, we have

  $$h_{v_1}^{(1)} = \frac{1}{2}(0+1) = \frac{1}{2}$$

  for Graph 1 and,

  $$h_{v_2}^{(1)} = \frac{1}{4}(0+0+1+1) = \frac{1}{2}$$

  for Graph 2, where we added a subscript to v to differentiate the graphs. This examples uncovers the main disadvantage of using the mean aggregation function: *the mean aggregation function fails to propagate relevant information when the proportion of features (embeddings) is the same over the neighborhood nodes.* Hence, Graphs 1 and 2 will end up with the same node embedding for node v.

- **Maximum:** The max aggregation function is: $\max = \max\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\}$. For $k = 1$, we have

  $$h_{v_1}^{(1)} = \max\{0, 1\} = 1$$

for Graph 1 and,

$$h_{v_2}^{(1)} = \max\{0, 0, 1, 1\} = 1$$

for Graph 2. The main problem with the maximum aggregation function is that: *the maximum aggregation function cannot differentiate distinct multi-sets (neighborhoods) which have the same composition of features (embeddings), as is the case of the two graphs.* Again, Graphs 1 and 2 will end up with the same node embedding for node v.

- **Sum:** The sum aggregation function is: $\text{sum} = \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}$. For $k = 1$, we have

  $$h_{v_1}^{(1)} = 0 + 1 = 1$$

  for Graph 1 and,

  $$h_{v_2}^{(1)} = 0 + 0 + 1 + 1 = 2$$

  for Graph 2. The advantage of the sum function is that: *the sum function is capable of differentiating distinct multi-sets (neighborhoods).* Considering the given aggregation functions, only in this case, node v will end up with a different node embedding for each one of the graphs.

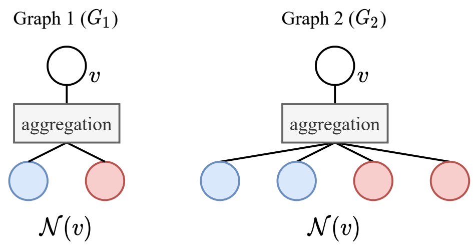

Figure 4: Two graphs to exemplify the importance of the aggregation function for the expressiveness of a GCN. The color of the nodes represents different features. Without loss of generality, we consider that blue nodes have $h_b^{(0)} = 0$ and red nodes have $h_r^{(0)} = 1$ for $b, r \in \mathcal{N}(v)$ (valid for both graphs).

1.3 Question 1.3 (6 points)

To prove that our isomorphism test based on GCN is at best as good as WL we intend to illustrate this through a proof by contradiction, which will lead to the conclusion that our isomorphism tests performance is upper bounded by the WL algorithm.
First we note the WL algorithm is defined in the following fashion.

$$l_v^k = HASH(l_v^{k-1}, [l_u^{k-1} \forall u \in N(v)])$$

Which boils down to a hash function that by definition is injective, which takes the input of the previous iteration for the node in question and "combines/aggregates" with the neighboring nodes of the same iteration.

We then consider our GCN model.

$$h_v^k = combine(h_v^{k-1}, aggregate[h_u^{k-1} \forall u \in N(v)]])$$

A thing to consider is that a very similar structure is found in both. This is the combine and hash function which similarly outputs a result based on the previous iteration embedding along with some "aggregation" of neighbors in the WL test and in GCN. Furthermore the GCN aggregation can in principle have collision and should not be thought of as fully injective.

From this we define the test that is, at the K'th iteration then the GCN isomorphism test and the WL test are to determine if 2 graphs are isomorphic. The 2 graphs are not isomorphic which means the following

$$\text{readout}(h_{v1}^k, \forall u \in v_1) \neq \text{readout}(h_{v2}^k, \forall u \in v_2)$$

If we then consider the contradiction in that the WL test cannot determine yet if they are isomorphic at the K'th iteration i.e.

$$l_{v1}^k = HASH(l_v^{k-1}, [l_u^{k-1} \forall u \in N(v_1)]) = l_{v2}^k = HASH(l_v^{k-1}, [l_u^{k-1} \forall u \in N(v_2)])$$

then that would mean the GCN similarly cannot determine if 2 graphs are isomorphic, then the GCN test reach such a conclusion that:

$$h_{v1}^k = comb(h_v^{k-1}, aggre[h_u^{k-1} \forall u \in N(v_1)]]) = h_{v2}^k = comb(h_v^{k-1}, aggre[h_u^{k-1} \forall u \in N(v_2)]])$$

Which essentially means that neither can determine if the the graphs are isomorphic, but ground truth is that they are not.

$$\text{readout}(h_{v1}^k, \forall u \in v_1) = \text{readout}(h_{v2}^k, \forall u \in v_2)$$

This thus means, if WL cannot determine, then we similarly cannot determine with GCN, and we are upper bounded by WL.

$$WL \geq GCN$$

For this reasons the GCN will be at best as good as the WL test, and never better.

2 Node Embeddings with TransE (25 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 in Homework 1 preserved structure in the sense that nodes connected by an edge were usually close together in the (onedimensional) 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**. 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 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 $\mathbf{l} \in R * k$, such that the difference between the embeddings of entities linked via the relationship l is approximately $\mathbf{l}$. That is, if $(h, l, t) \in S$, TransE tries to ensure that $\mathbf{h}+\mathbf{l} \approx \mathbf{t}$. Simultanesouly, TransE tries to make sure that $\mathbf{h}+\mathbf{l} \approx \mathbf{t}$ 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 relationships in the graph, and bold letters $\mathbf{e}$, $\mathbf{l}$, etc., to denote their corresponding embeddings. TransE accomplishes this by minimizing the following loss:

$$\mathcal{L} = \sum_{(h,l,t)\in S} \left(\sum_{(h',l,t')\in S'_{(h,l,t)}} [\gamma + d(\mathbf{h}+\mathbf{l}, \mathbf{t}) - d(\mathbf{h'}+\mathbf{l}, \mathbf{t'})]_+ \right) \tag{11}$$

Here (h', l, t') are ”corrupted” triplets, chosen from the set $S'_{(h,l,t)}$ of corruptions of (h, l, t), which are all triples where either h or t (but not both) is replaced by a random entity.

$$S'_{(h,l,t)} = \{(h', l, t)|h' \in E\} \cup \{(h, l, t')|t' \in E\} \tag{12}$$

Additionally, $\pounds\gamma > 0$ 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: $\|e\|^2 = 1$ for every $e \in E$. For reference, here is the **TransE** algorithm, as described in the original paper on page 3:

2.1 Question 2.1 (3 points)

Say we were intent on using a simpler loss function. Our objective function includes a term maximizing the distance between $h' + l$ and t'. If we instead simplified the objective, and just tried to minimize

$$\mathcal{L}_{simple} = \sum_{(h,l,t)\in S} d(\mathbf{h}+\mathbf{l}, \mathbf{t}) \tag{13}$$

we would obtain a useless embedding. Give an example of a simple graph and corresponding embeddings that will minimize the new objective function all the way to zero, but still give a completely useless embedding.

2.1.1 Solution

For any graph, if all nodes are embedded in the same point and all edges are embedded in zero, i.e $\mathbf{h} = \mathbf{t} \forall h, t$ and $\mathbf{l} = \mathbf{0} \forall l$. $\mathcal{L}_{simple}$ would be zero, but the embedding would be useless.

2.2 Question 2.2 (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

$$\mathcal{L} = \sum_{(h,l,t)\in S} \left(\sum_{(h',l,t')\in S'_{(h,l,t)}} [d(\mathbf{h}+\mathbf{l},\mathbf{t}) - d(\mathbf{h'}+\mathbf{l},\mathbf{t'})]_+ \right) \tag{14}$$

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 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.)

2.2.1 Solution

If we choose the same embedding as before; $\mathbf{h} = \mathbf{t} \forall h, t$ and $\mathbf{l} = \mathbf{0} \forall l$. the distance bewteen any pair of nodes are zero. Because of the positive function $[\cdot]_+$ all cases where the distance of negative edges is bigger that positive edges $\mathcal{L}$ is zero, but it is also zero if the distances are the same.

2.3 Question 2.3 (7 points)

Recall that TransE normalizes every entity embedding to have unit length. 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?

2.3.1 Solution

according to the paper ”Translating Embeddings for Modeling Multi-relational Data” by Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston and Oksana Yakhnenko states: *The optimization is carried out by stochastic gradient descent (in minibatch mode), over the possible **h**, **l** and **t**, with the additional constraints that the L2-norm of the embeddings of the entities is 1 (no regularization or norm constraints are given to the label embeddings **l**). This constraint is important for our model, as it is for previous embedding-based methods [3, 6, 2], because it prevents the training process to trivially minimize $\mathcal{L}$ by artificially increasing entity embeddings norms.*

2.4 Question 2.4 (10 points)

Give an example of a simple graph for which no perfect embedding exists, i.e., no embedding perfectly satisfies $\mathbf{u} + \mathbf{l} = \mathbf{v} (u, l, v) \in S$ and$\mathbf{u} + \mathbf{l} \neq \mathbf{v} (u, l, v) \notin S$, for any choice of entity embeddings ($\mathbf{e}$ for $e \in E$) and relationship embeddings ($\mathbf{l}$ for $l \in L$). Explain why this graph has no perfect embedding in this system, and what that means about the expressiveness of TransE embeddings.

2.4.1 Solution

TransE can't moddle symetry. A simple graph that cannot be moddled is a graph with two nodes af different types with two edges of the same type with oppesite directions, see 9

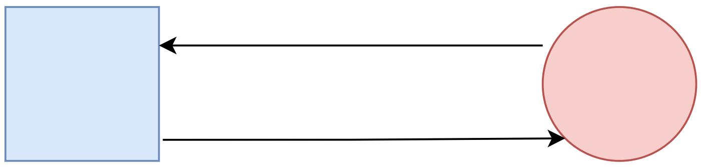

Figure 5: Simple graph that can't be moddeled by TransE.

3 Expressive Power of Knowledge Graph Embeddings (10 points)

Throughout this question, we denote a Knowledge graph using the triplet (h, ℓ, t), where h is the head, ℓ is the relation, and t is the tail; head and tail are nodes of a graph, while the relation is an edge. Our goal here is to make the embedding (h, ℓ) as close as possible to the embedding of t using a shallow learning approach. This means that the entities h, t and the relation ℓ are defined as vectors $(\mathbf{h}, \boldsymbol{\ell}, \mathbf{t})$ in the embedding space $\mathbb{R}^d$, which in its turn is defined by a look-up table.

3.1 Question 3.1 (3 points)

For the given forms of relations, we have for the TransE model:

- **Symmetry** means that $\ell(h,t) = \ell(t,h)$. Considering the TransE model, we have that $\mathbf{h}+\boldsymbol{\ell} \approx \mathbf{t}$. If we write the symmetric relationship, we end up with $\mathbf{h} + \boldsymbol{\ell} \approx \mathbf{t} + \boldsymbol{\ell}, \therefore \boldsymbol{\ell} = \mathbf{0}$. This is undesirable since this means that there is no actual relationship between the entities; they are the same. Therefore, TransE is not able to capture symmetric relationships.

- **Inverse** means that $\ell_1(h,t) \Longrightarrow \ell_2(t,h)$. Given the first relation ℓ_1, we have $\mathbf{h} + \boldsymbol{\ell}_1 \approx \mathbf{t}$. Since the inverse element of vector addition is $-\boldsymbol{\ell}_1$, such that $\boldsymbol{\ell}_1 + (-\boldsymbol{\ell}_1) = \mathbf{0}$, we can go from t back to h by subtracting $\boldsymbol{\ell}_1$. Hence, $\mathbf{h} + \boldsymbol{\ell}_2 \approx \mathbf{t} = \mathbf{h} - \boldsymbol{\ell}_1 \approx \mathbf{t}$. Therefore, TransE is able to capture inverse relationships.

- **Composition** means that $\ell_1(a,b) \wedge \ell_2(b,c) \Longrightarrow \ell_3(c,a)$, where $\wedge$ is the logical symbol for conjugation. Using the TransE model, we have $\mathbf{a} + \boldsymbol{\ell}_1 \approx \mathbf{b}$, and $\mathbf{b} + \boldsymbol{\ell}_2 \approx \mathbf{c}$. Then, using the second into the first, $\mathbf{a} + \boldsymbol{\ell}_1 \approx \mathbf{c} - \boldsymbol{\ell}_2$, meaning $\mathbf{a} + \boldsymbol{\ell}_3 \approx \mathbf{c}$ with $\boldsymbol{\ell}_3 = (\boldsymbol{\ell}_1 + \boldsymbol{\ell}_2)$. The last holds due to the closure property of vector addition and the parallelogram rule. Therefore, TransE is able to capture transitive relationships (compositions).

3.2 Question 3.2 (3 points)

For this new model we transform the relational representation of a vector addition in TransE into a shift in the angle in RotatE.
That is to say TransE uses (h, r, t) to present the relation between a head and a tail, meanwhile RotatE uses (h, θ, t) for this representation, where θ is an angle that the embedding h is rotated by. This can also be interpreted as the embeddings of TransE are written with polar coordinates instead, and the relation vector is instead the angular difference between head and tail.

3.2.1 Symmetry

The pattern of symmetry can be implemented through RotatE by considering figure 6 where we have (h, θ, t)

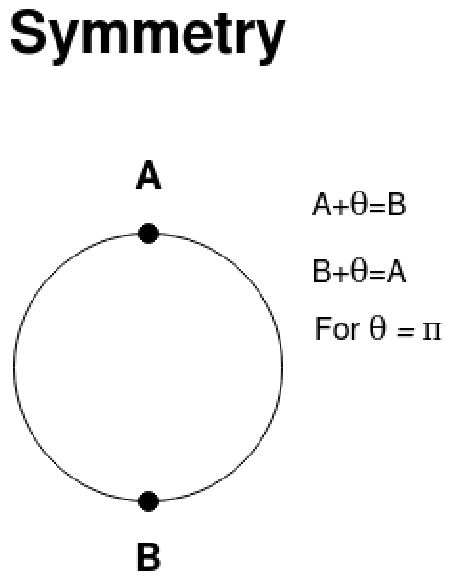

Figure 6: Symmetric relation with RotatE

We can see that by rotating from an embedding 180 degrees or π we end up at some tail where if we apply a similar rotation then (t, θ, h). We rotate with the same relation, and end up at our starting point the head again. Thus RotatE support symmetric relations.

3.2.2 Inverse

The inverse relation is also possible with RotatE, by considering figure 7

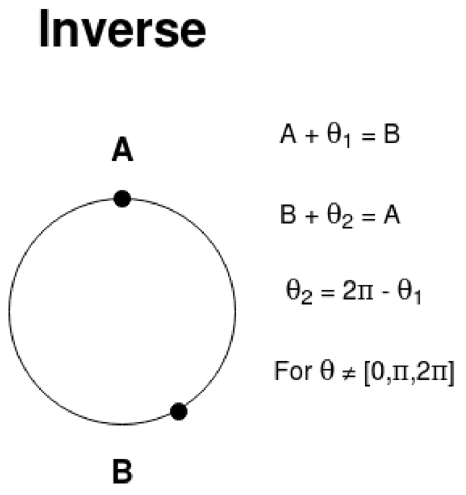

Figure 7: Inverse relation with RotatE

We see that by rotating from A to B we move 160 degrees. By rotating from B to A we will rotate 360-θ degrees hence the relation is $\theta_2 = 360 - \theta_1$. Therefore inverse relations are also possible as $\theta_1 \neq \theta_2$

3.2.3 Composition

For composition, we have that $\mathbf{a} \circ \boldsymbol{\theta}_1 \approx \mathbf{b}$, and $\mathbf{b} \circ \boldsymbol{\theta}_2 \approx \mathbf{c}$. So, $\mathbf{a} \circ \boldsymbol{\theta}_1 \approx \mathbf{c} \circ -\boldsymbol{\theta}_2 \rightarrow \mathbf{a} \circ (\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2) \approx \mathbf{c}$, using the inverse of rotation as $\boldsymbol{\theta} \circ (-\boldsymbol{\theta}) = \mathbf{0}$. In order to the result of $\mathbf{a} \circ (\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2)$ be closed under the rotation operator $\circ$, we need to certify that $(\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2)$ is within the range $[0, 2\pi)$. To so so, we add the following operation $(\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2) \mod 2\pi$. Therefore, $\mathbf{a} \circ ((\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2) \mod 2\pi) \approx \mathbf{c} \rightarrow \mathbf{a} \circ \boldsymbol{\theta}_3 \approx \mathbf{c}$. Therefore, RotatE is able to capture composite relations.

3.3 Question 3.3 (4 points)

- Given the performance seen in 3.2 for the different patterns then graphs involving symmetry, inverse, composition and antisymmetry (see figure 8) can all be supported. However, the pattern of 1-to-n cannot be realized similarly to TransE due to the rotation of an embedding leads to a single point, hence multiple tails cannot be realized.

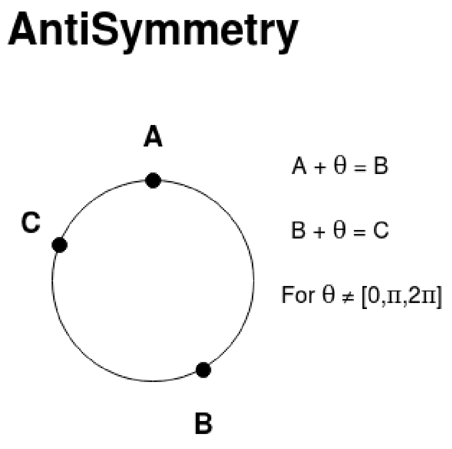

Figure 8: Caption

Antisymmetry as mentioned can also be realized with RotatE, given the constraint that the angles are not all $\theta \neq \pi$ as it would be symmetrical if that happened. Otherwise any angle would lead to a symmetric relationship
A thing to note is that the antisymmetry is somehwat limited. Eventually by applying the rotational angle we would end up back at the head again depending how the angles line up with each other, where a periodicity will occur. that is to say $(h, \theta, t_1), (t_1, \theta_2, t_2)..., (t_n, \theta, h)$. When this occurs is fully dependent on the rotational angles of θ.

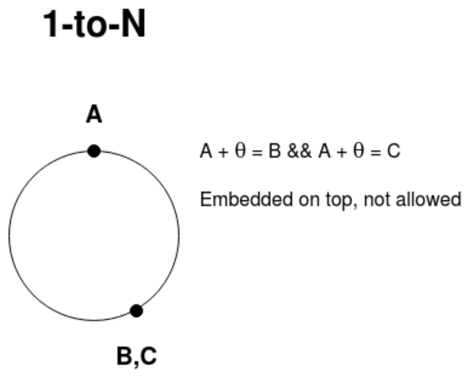

Figure 9: Caption

From this we overall found that rotatE is capable of the same as TransE with the addition of symmetric relations, at the cost of more complexity in how these are defined.

4 Honor Code (0 points)

Andreas Casparsen, Josefine Holm, Tobias Kallehauge and Victor Croisfelt collaborated on this homework.