Embedding Based Models – Lecture 17.2

- Embedding-based models build embeddings for entities and relations to predict tuples.

- Embedding-based models build embeddings for entities and relations to predict tuples.
- Simplest case: relations between two entities.
  E.g., predicting the rating for a person on a movie (collaborative filtering).

Embedding Based Models – Lecture 17.2

- Embedding-based models build embeddings for entities and relations to predict tuples.
- Simplest case: relations between two entities.
  E.g., predicting the rating for a person on a movie (collaborative filtering).
- Knowledge graphs: predict subject-verb-object triples.

Learning a relation between two entities

A relation between users and items (movies). From Movielens:

User	Item	Rating	Timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
...	...	...	

- Netflix: 500K users, 17k movies, 100M ratings (withdrawn).
- Movielens: multiple datasets from 100K to 25M ratings, with links to IMDB, plus some user properties

Learning a relation between two entities

A relation between users and items (movies). From Movielens:

User	Item	Rating	Timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
...	...	...	

- Netflix: 500K users, 17k movies, 100M ratings (withdrawn).
- Movielens: multiple datasets from 100K to 25M ratings, with links to IMDB, plus some user properties

$\widehat{r_{ui}}$ = predicted rating of user u on item i

Es = set of (u, i, r) tuples in the training set (ignoring timestamp)

Minimize sum squares error:

$$\sum_{(u,i,r)\in Es} (\widehat{r_{ui}} - r)^2$$

Learning Relational Models with Latent Variables

- Predict same for all ratings: $\widehat{r_{ui}} = \mu$

Learning Relational Models with Latent Variables

- Predict same for all ratings: $\widehat{r_{ui}} = \mu$
- Adjust for each user and item: $\widehat{r_{ui}} = \mu + b_1[u] + b_2[i]$

Learning Relational Models with Latent Variables

- Predict same for all ratings: $\widehat{r_{ui}} = \mu$
- Adjust for each user and item: $\widehat{r_{ui}} = \mu + b_1[u] + b_2[i]$
  Question: $b_1[u]$ could be negative even though all ratings of u are higher that average. How?

Learning Relational Models with Latent Variables

- Predict same for all ratings: $\widehat{r_{ui}} = \mu$
- Adjust for each user and item: $\widehat{r_{ui}} = \mu + b_1[u] + b_2[i]$
  Question: $b_1[u]$ could be negative even though all ratings of u are higher that average. How?
- Question: How can this be used to personalize recommendations?

Learning Relational Models with Latent Variables

- Predict same for all ratings: $\widehat{r_{ui}} = \mu$
- Adjust for each user and item: $\widehat{r_{ui}} = \mu + b_1[u] + b_2[i]$
  Question: $b_1[u]$ could be negative even though all ratings of u are higher that average. How?
- Question: How can this be used to personalize recommendations?
- One latent feature: f_i for each item and g_u for each user

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + f_1[u] * f_2[i]$$

Learning Relational Models with Latent Variables

- Predict same for all ratings: $\widehat{r_{ui}} = \mu$
- Adjust for each user and item: $\widehat{r_{ui}} = \mu + b_1[u] + b_2[i]$
  Question: $b_1[u]$ could be negative even though all ratings of u are higher that average. How?
- Question: How can this be used to personalize recommendations?
- One latent feature: f_i for each item and g_u for each user

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + f_1[u] * f_2[i]$$

  - Positive $f_1[u]$ forms a soft clustering of users.

Learning Relational Models with Latent Variables

- Predict same for all ratings: $\widehat{r_{ui}} = \mu$
- Adjust for each user and item: $\widehat{r_{ui}} = \mu + b_1[u] + b_2[i]$
  Question: $b_1[u]$ could be negative even though all ratings of u are higher that average. How?
- Question: How can this be used to personalize recommendations?
- One latent feature: f_i for each item and g_u for each user

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + f_1[u] * f_2[i]$$

  - Positive $f_1[u]$ forms a soft clustering of users.
  - Positive $f_2[i]$ forms a soft clustering of items.

Learning Relational Models with Latent Variables

- Predict same for all ratings: $\widehat{r_{ui}} = \mu$
- Adjust for each user and item: $\widehat{r_{ui}} = \mu + b_1[u] + b_2[i]$
  Question: $b_1[u]$ could be negative even though all ratings of u are higher that average. How?
- Question: How can this be used to personalize recommendations?
- One latent feature: f_i for each item and g_u for each user

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + f_1[u] * f_2[i]$$

  - Positive $f_1[u]$ forms a soft clustering of users.
  - Positive $f_2[i]$ forms a soft clustering of items.
  - How do these clusterings interact?

Learning Relational Models with Latent Variables

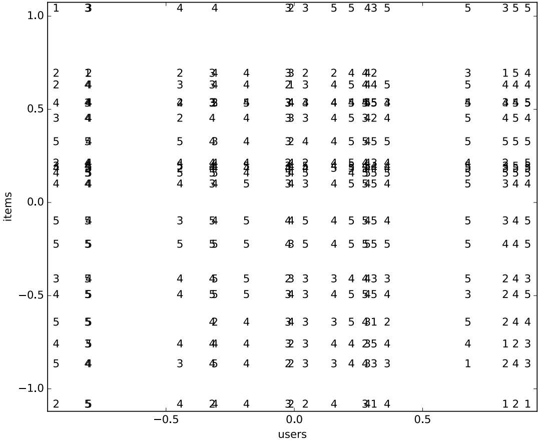
- Predict same for all ratings: $\widehat{r_{ui}} = \mu$
- Adjust for each user and item: $\widehat{r_{ui}} = \mu + b_1[u] + b_2[i]$
  Question: $b_1[u]$ could be negative even though all ratings of u are higher that average. How?
- Question: How can this be used to personalize recommendations?
- One latent feature: f_i for each item and g_u for each user

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + f_1[u] * f_2[i]$$

  - Positive $f_1[u]$ forms a soft clustering of users.
  - Positive $f_2[i]$ forms a soft clustering of items.
  - How do these clusterings interact?
  - What about negative values?

What is being learned? (Single latent feature)

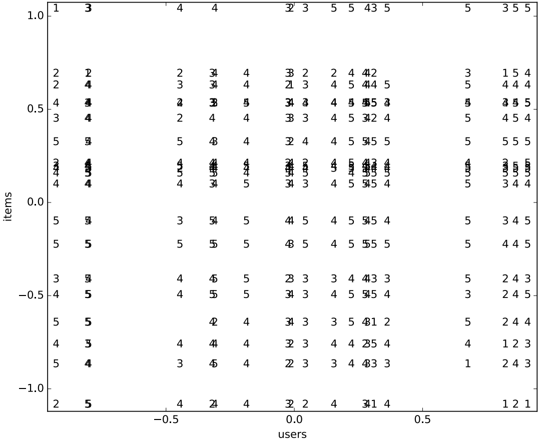
for each (u, i, r): r is plotted at point $(f_1[u], f_2[i])$.



`aipython: relnCollFilt.py`

What is being learned? (Single latent feature)

for each (u, i, r): r is plotted at point $(f_1[u], f_2[i])$.



`aipython: relnCollFilt.py`

What pattern would you expect?

Learning Relational Models with Multiple Latent Properties

- k latent features (Python notation):

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + \sum_f E_1[u][f] * E_2[i][f]$$

Learning Relational Models with Multiple Latent Properties

- k latent features (Python notation):

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + \sum_f E_1[u][f] * E_2[i][f]$$

$E_1[u]$ is the user embedding, a vector of numbers.

Learning Relational Models with Multiple Latent Properties

- k latent features (Python notation):

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + \sum_f E_1[u][f] * E_2[i][f]$$

$E_1[u]$ is the user embedding, a vector of numbers.
$E_2[i]$ is the item embedding, a vector of numbers.

Learning Relational Models with Multiple Latent Properties

- k latent features (Python notation):

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + \sum_f E_1[u][f] * E_2[i][f]$$

  $E_1[u]$ is the user embedding, a vector of numbers.
  $E_2[i]$ is the item embedding, a vector of numbers.

- Regularize parameters except

Learning Relational Models with Multiple Latent Properties

- k latent features (Python notation):

$$\widehat{r_{ui}} = \mu + b_1[u] + b_2[i] + \sum_f E_1[u][f] * E_2[i][f]$$

  $E_1[u]$ is the user embedding, a vector of numbers.
  $E_2[i]$ is the item embedding, a vector of numbers.

- Regularize parameters except μ.

Regularizing

- $L2$ regularization, minimize:

$$\left(\sum_{(u,i,r)\in Es} (\widehat{r_{ui}} - r)^2\right) + \lambda \sum_{parameter\ \ p} p^2$$

$L2$ regularization

Minimize:

$$\left(\sum_{(u,i,r)\in Es} (\mu + b_1[u] + b_2[i] + \sum_k E_1[u][f] * E_2[i][f] - r)^2 \right) + \lambda \left(\sum_i (b_1[u]^2 + \sum_f E_1[u][f]^2) + \sum_u (b_2[i]^2 + \sum_f E_2[i][f]^2) \right)$$

where λ is a regularization parameter.

$L2$ regularization

Minimize:

$$\left(\sum_{(u,i,r)\in Es} (\mu + b_1[u] + b_2[i] + \sum_k E_1[u][f] * E_2[i][f] - r)^2\right)$$
$$+ \lambda \left(\sum_i (b_1[u]^2 + \sum_f E_1[u][f]^2) + \sum_u (b_2[i]^2 + \sum_f E_2[i][f]^2)\right)$$

where λ is a regularization parameter.

To find minimizing parameters:

- Gradient descent
- Iterative least squares: fix one of E_1 and E_2; the problem is ridge regression in the other.

$\mu :=$ average rating
assign $E_1[u][f]$, $E_2[i][f]$ randomly and assign $b_1[i], b_2[u]$ arbitrarily
repeat:
    for each $(u, i, r) \in Es$:
        $e := \mu + b_1[i] + b_2[u] + \sum_k E_1[u][f] * E_2[i][f] - r$
        $b_1[i] := b_1[i] - \eta * e$
        $b_2[u] := b_2[u] - \eta * e$
        for each feature f:
            $E_1[u][f] := E_1[u][f] - \eta * e * E_2[i][f]$
            $E_2[i][f] := E_2[i][f] - \eta * e * E_1[u][f]$

$\mu :=$ average rating
assign $E_1[u][f]$, $E_2[i][f]$ randomly and assign $b_1[i], b_2[u]$ arbitrarily
repeat:

- **for each** $(u, i, r) \in Es$:
  - $e := \mu + b_1[i] + b_2[u] + \sum_k E_1[u][f] * E_2[i][f] - r$
  - $b_1[i] := b_1[i] - \eta * e$
  - $b_2[u] := b_2[u] - \eta * e$
  - **for each** feature f:
    - $E_1[u][f] := E_1[u][f] - \eta * e * E_2[i][f]$
    - $E_2[i][f] := E_2[i][f] - \eta * e * E_1[u][f]$
- **for each** item i:
  - $b_1[i] := b_1[i] - \eta * \lambda * b_1[i]$
  - **for each** feature f:
    - $E_1[u][f] := E_1[u][f] - \eta * \lambda * E_1[u][f]$
- **for each** user u:
  - $b_2[u] := b_2[u] - \eta * \lambda * b_2[u]$
  - **for each** feature k:
    - $E_2[i][f] := E_2[i][f] - \eta * \lambda * E_2[i][f]$

Variations

- What is you want to predict Boolean $rating > 3$?

Variations

- What is you want to predict Boolean *rating* > 3?
  - Use sigmoid.

Variations

- What is you want to predict Boolean *rating* > 3?
  - Use sigmoid.
  - What should we minimize?

Variations

- What is you want to predict Boolean *rating* > 3?
  - Use sigmoid.
  - What should we minimize?
  - How does the algorithm change?

Variations

- What is you want to predict Boolean $rating > 3$?
  - Use sigmoid.
  - What should we minimize?
  - How does the algorithm change?
- What if we want to predict *rated*, where $rated(u, i)$ is true if $(u, i, r) \in Es$ for some r?

Variations

- What is you want to predict Boolean $rating > 3$?
  - Use sigmoid.
  - What should we minimize?
  - How does the algorithm change?
- What if we want to predict *rated*, where $rated(u, i)$ is true if $(u, i, r) \in Es$ for some r?
  - There are no negative examples!

Variations

- What is you want to predict Boolean *rating* > 3?
  - Use sigmoid.
  - What should we minimize?
  - How does the algorithm change?
- What if we want to predict *rated*, where $rated(u, i)$ is true if $(u, i, r) \in Es$ for some r?
  - There are no negative examples!
  - Use k random examples for each positve example!
    but

Variations

- What is you want to predict Boolean $rating > 3$?
  - Use sigmoid.
  - What should we minimize?
  - How does the algorithm change?
- What if we want to predict *rated*, where $rated(u, i)$ is true if $(u, i, r) \in Es$ for some r?
  - There are no negative examples!
  - Use k random examples for each positve example!
    but the average probability is

Variations

- What is you want to predict Boolean $rating > 3$?
  - Use sigmoid.
  - What should we minimize?
  - How does the algorithm change?
- What if we want to predict *rated*, where $rated(u, i)$ is true if $(u, i, r) \in Es$ for some r?
  - There are no negative examples!
  - Use k random examples for each positve example!
    but the average probability is $1/k$, which is not derived from the data.

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s, r, o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o] \sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s, r, o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o] \\ \sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

  - a global bias μ

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s,r,o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o] \\ \sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

  - a global bias μ
  - two biases for each entity e: $b_1[e]$ used when e is in the first position and $b_3[e]$, for when e in third position

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s,r,o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o]$$
$$\sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

  - a global bias μ
  - two biases for each entity e: $b_1[e]$ used when e is in the first position and $b_3[e]$, for when e in third position
  - a bias for each relation r, namely $b_2[r]$

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s,r,o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o] \\ \sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

  - a global bias μ
  - two biases for each entity e: $b_1[e]$ used when e is in the first position and $b_3[e]$, for when e in third position
  - a bias for each relation r, namely $b_2[r]$
  - matrixes E_1 and E_3
    $E_1[e]$ is subject embedding for entity e

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s,r,o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o]$$
$$\sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

  - a global bias μ
  - two biases for each entity e: $b_1[e]$ used when e is in the first position and $b_3[e]$, for when e in third position
  - a bias for each relation r, namely $b_2[r]$
  - matrixes E_1 and E_3
    $E_1[e]$ is subject embedding for entity $e \longrightarrow$ latent properties

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s,r,o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o] \\ \sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

  - a global bias μ
  - two biases for each entity e: $b_1[e]$ used when e is in the first position and $b_3[e]$, for when e in third position
  - a bias for each relation r, namely $b_2[r]$
  - matrixes E_1 and E_3
    $E_1[e]$ is subject embedding for entity $e \longrightarrow$ latent properties
    $E_3[e]$, is object embedding entity e

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s,r,o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o] \\ \sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

  - a global bias μ
  - two biases for each entity e: $b_1[e]$ used when e is in the first position and $b_3[e]$, for when e in third position
  - a bias for each relation r, namely $b_2[r]$
  - matrixes E_1 and E_3
    $E_1[e]$ is subject embedding for entity $e \longrightarrow$ latent properties
    $E_3[e]$, is object embedding entity e
  - matrix E_2, where $E_2[r]$ is relation embedding for relation r.

Knowledge Graphs and Triples

- A knowledge graph is defined in term of triples of the form (s, r, o), with subject s, relation (verb) r, and object o
- The extension of matrix factorization to triples is polyadic decomposition:

$$\widehat{p}((s, r, o)) = sigmoid(\mu + b_1[s] + b_2[r] + b_3[o] \\ \sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])$$

  - a global bias μ
  - two biases for each entity e: $b_1[e]$ used when e is in the first position and $b_3[e]$, for when e in third position
  - a bias for each relation r, namely $b_2[r]$
  - matrixes E_1 and E_3
    $E_1[e]$ is subject embedding for entity $e \longrightarrow$ latent properties
    $E_3[e]$, is object embedding entity e
  - matrix E_2, where $E_2[r]$ is relation embedding for relation r.

All embeddings $E_1[e]$, $E_2[r]$ and $E_3[e]$ are the same length.

Learning Polyadic Decomposition

- To optimize log loss with $L2$ regularization:

Learning Polyadic Decomposition

- To optimize log loss with $L2$ regularization:
  same as previous algorithm with a different predictor and more parameters to tune and regularize.

Learning Polyadic Decomposition

- To optimize log loss with $L2$ regularization:
  same as previous algorithm with a different predictor and more parameters to tune and regularize.
- Requires negative examples, but

Learning Polyadic Decomposition

- To optimize log loss with $L2$ regularization:
  same as previous algorithm with a different predictor and more parameters to tune and regularize.
- Requires negative examples, but knowledge graphs don't have negative examples.

Learning Polyadic Decomposition

- To optimize log loss with $L2$ regularization:
  same as previous algorithm with a different predictor and more parameters to tune and regularize.
- Requires negative examples, but knowledge graphs don't have negative examples.
- Regularize μ or provide made-up negative examples.

Learning Polyadic Decomposition

- To optimize log loss with $L2$ regularization:
  same as previous algorithm with a different predictor and more parameters to tune and regularize.
- Requires negative examples, but knowledge graphs don't have negative examples.
- Regularize μ or provide made-up negative examples.
- But, not all relations have same number of negative examples, eg. "*married-to*" vs "*has-streamed*".

Improving Polyadic Decomposition

- Suppose triples are of the form $(u, likes, m)$ and $(m, directed\_by, d)$.

Improving Polyadic Decomposition

- Suppose triples are of the form $(u, likes, m)$ and $(m, directed\_by, d)$.
  How can we represent "Sam likes movies directed by Bong Joon-ho"?

Improving Polyadic Decomposition

- Suppose triples are of the form $(u, likes, m)$ and $(m, directed\_by, d)$.
  How can we represent "Sam likes movies directed by Bong Joon-ho"?
  The subject and object embeddings for movies are independent of each other, so this cannot be represented or learned.

Improving Polyadic Decomposition

- Suppose triples are of the form $(u, likes, m)$ and $(m, directed\_by, d)$.
  How can we represent "Sam likes movies directed by Bong Joon-ho"?
  The subject and object embeddings for movies are independent of each other, so this cannot be represented or learned.
- Solution: also represent $(m, likes^{-1}, u)$ and $(d, directed\_by^{-1}, m)$.

Improving Polyadic Decomposition

- Suppose triples are of the form $(u, likes, m)$ and $(m, directed\_by, d)$.
  How can we represent "Sam likes movies directed by Bong Joon-ho"?
  The subject and object embeddings for movies are independent of each other, so this cannot be represented or learned.
- Solution: also represent $(m, likes^{-1}, u)$ and $(d, directed\_by^{-1}, m)$.
- polyadic decomposition with inverses:

$$\widehat{p}(h, r, t) = \frac{1}{2}(\widehat{pd}(h, r, t) + \widehat{pd}(t, r^{-1}, h))$$

where $\widehat{pd}$ is the prediction from the polyadic decomposition.

What does polyadic decomposition learn?

- The polyadic decomposition is fully expressive: it can represent (with error less than any ϵ), any relation.

What does polyadic decomposition learn?

- The polyadic decomposition is fully expressive: it can represent (with error less than any ϵ), any relation.
- Initially assume subject and object embeddings are non-negative and bounded.
- Each embedding position in the subject/object embedding forms a soft clustering of entities.

What does polyadic decomposition learn?

- The polyadic decomposition is fully expressive: it can represent (with error less than any ϵ), any relation.
- Initially assume subject and object embeddings are non-negative and bounded.
- Each embedding position in the subject/object embedding forms a soft clustering of entities.
- For each embedding position, a relation with a high value ($\gg 0$) in that position, the entities in the subject soft cluster are related to the entities in the object soft cluster. (The product is only high when all three are high).

What does polyadic decomposition learn?

- The polyadic decomposition is fully expressive: it can represent (with error less than any ϵ), any relation.
- Initially assume subject and object embeddings are non-negative and bounded.
- Each embedding position in the subject/object embedding forms a soft clustering of entities.
- For each embedding position, a relation with a high value ($\gg 0$) in that position, the entities in the subject soft cluster are related to the entities in the object soft cluster. (The product is only high when all three are high).
- A relation with a value $\ll 0$ in a position forms exceptions.

What does polyadic decomposition learn?

- The polyadic decomposition is fully expressive: it can represent (with error less than any ϵ), any relation.
- Initially assume subject and object embeddings are non-negative and bounded.
- Each embedding position in the subject/object embedding forms a soft clustering of entities.
- For each embedding position, a relation with a high value ($\gg 0$) in that position, the entities in the subject soft cluster are related to the entities in the object soft cluster. (The product is only high when all three are high).
- A relation with a value $\ll 0$ in a position forms exceptions.
- For possibly negative subject and object embeddings:
  product of even number of negative values is positive
  product of odd number of negative values is negative

What does polyadic decomposition learn?

- The polyadic decomposition is fully expressive: it can represent (with error less than any ϵ), any relation.
- Initially assume subject and object embeddings are non-negative and bounded.
- Each embedding position in the subject/object embedding forms a soft clustering of entities.
- For each embedding position, a relation with a high value ($\gg 0$) in that position, the entities in the subject soft cluster are related to the entities in the object soft cluster. (The product is only high when all three are high).
- A relation with a value $\ll 0$ in a position forms exceptions.
- For possibly negative subject and object embeddings:
  product of even number of negative values is positive
  product of odd number of negative values is negative
- The addition lets these values be combined.

What does polyadic decomposition learn?

- The polyadic decomposition makes predictions by clustering entities in different ways.

What does polyadic decomposition learn?

- The polyadic decomposition makes predictions by clustering entities in different ways.
- It learns about each entity; embeddings are used to predict interactions.
- It does not learn general knowledge that can be applied to other populations.