Large Scale Graph ML

21.07.2021

Group 0 - Outliers

Magdalena Baumgärtl, Moritz Koch,
Patrick Tamunjoh, Pia Hammer, Yiwei Li

Overview

1. Task description
2. Introduction to the dataset
3. Baseline models
4. Trained models
5. Results
6. Conclusion & Lessons learned

Why Large Scale Graph Machine Learning?

- Many graph-structured real-world applications
  - Social Networks
  - Recommender Systems
  - Linked Web documents

- Real world data forms very large graphs
  - Billions of edges or millions of graphs

- Promising domain of active research

What was the Task?

OGB-LSC — Large - Scale Challenge — @ KDD Cup 2021

- Construct GNN for large scale graph data
- Keep as much information as possible

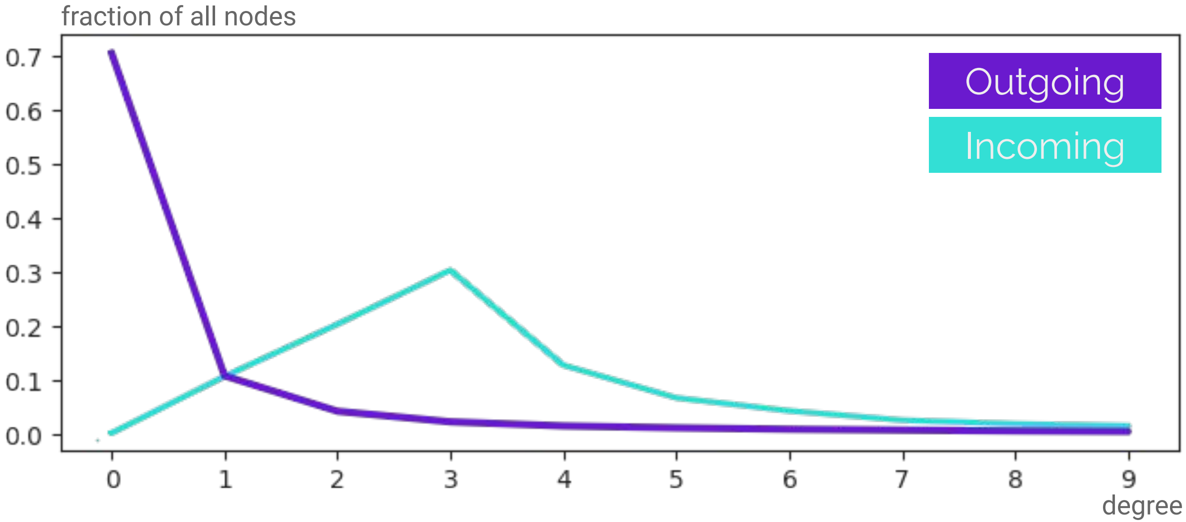
- OGB-LSC @ KDD Cup 2021: Link prediction on large scale graphs
    - Multi-relational graph
    - Graph consists of head-relation-tail triples
    - Predict the correct tail for a given head-relation pair
    - Provide the sorted top 10 tails for a given sample of 1001

The Dataset • WikiKG90M

Number of entities	87,143,637
Number of relations	1,315
Relation occurrence max	174,439,560
Relation occurrence mean	381,110.63

Number of feature dimensions	768
Number of training samples	501,160,482

The Dataset • WikiKG90M



Degree	Value
Outgoing mean	5.75
Outgoing max	8,320
Incoming mean	5.75
Incoming max	36,424,411

Baseline Models

Entity Co-Occurrence

- Untrained baseline model
- Relations are ignored completely

- Scoring is based on head and tail occurrences
    - Given a head, the most common tail is scored 1
    - All other tails are scored 0

Pseudo Typing

- Untrained baseline model
- Heads are ignored completely

- Scoring is based on relation and tail occurrences
  - Given a relation, the most common tail is scored 1
  - All other tails are scored 0

Entity Co-Occurrence & Pseudo Typing

- Trained baseline model
- Uses the trained Entity Co-Occurrence & Pseudo Typing baselines as input

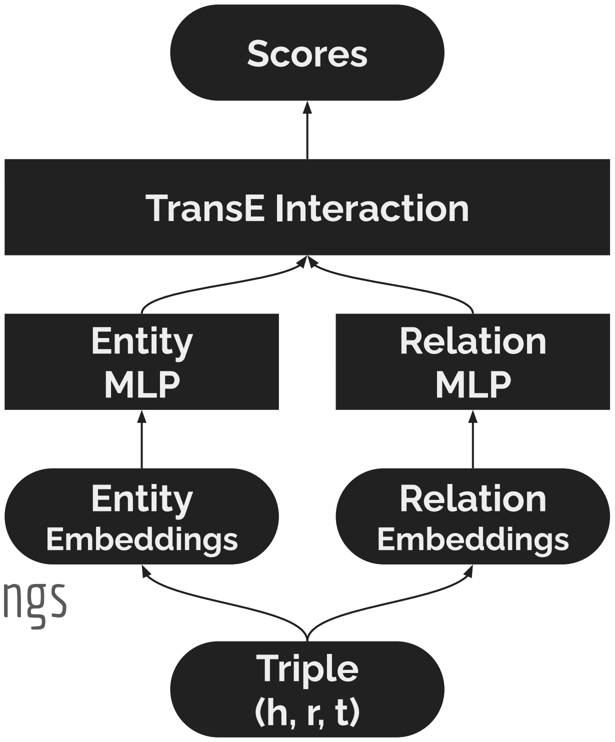
- Scores are derived from the input baselines
  - Weighted sum of the individual baseline scores

Trained Models

MLP Embeddings Model

- Uses triples as input, no features
- Entities and relations are represented in vector space
  - Initiate embeddings with random values
  - Embeddings are passed through MLP
  - An interaction function is applied to the final embeddings

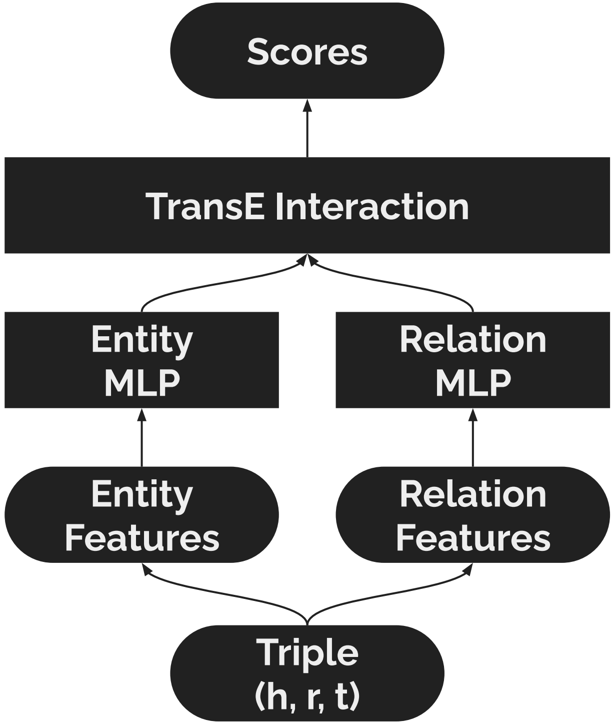
- High number of entities forces low dimensional embeddings
- MLP to increase complexity (16 ➡ 64 ➡ 32)
- Negative samples are generated by corruption



Entity & Relation Feature Model

- Uses entity and relation features
  - Features are loaded on demand
  - Features are passed through MLP for “enhancement”
  - An interaction function is applied to the enhanced features

- Feature loading makes up most of training duration
- Negative samples are generated by corruption

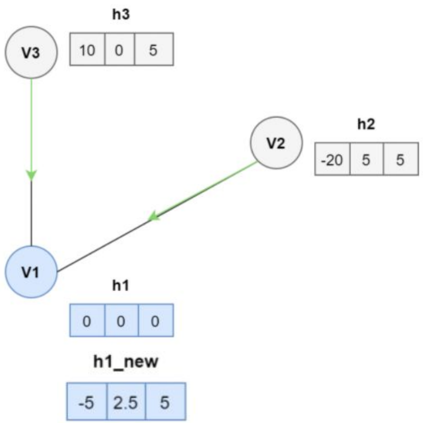


ComplEx with PyTorch BigGraph

- Distributed system for learning graph embeddings
- Designed for very large graphs
- Used to train the ComplEx Model:
  - Semantic matching model
  - Calculates the matching latent semantics of entities and relations embodied in their vector space representations
  - Based on complex Embeddings

Graph Convolutional Networks (GCNs)

1. Nodes are represented by a vector
2. Vectors get aggregated for each node ("message")
3. The vector of the current node gets update using the messages
4. Process can be repeated by multiple layers

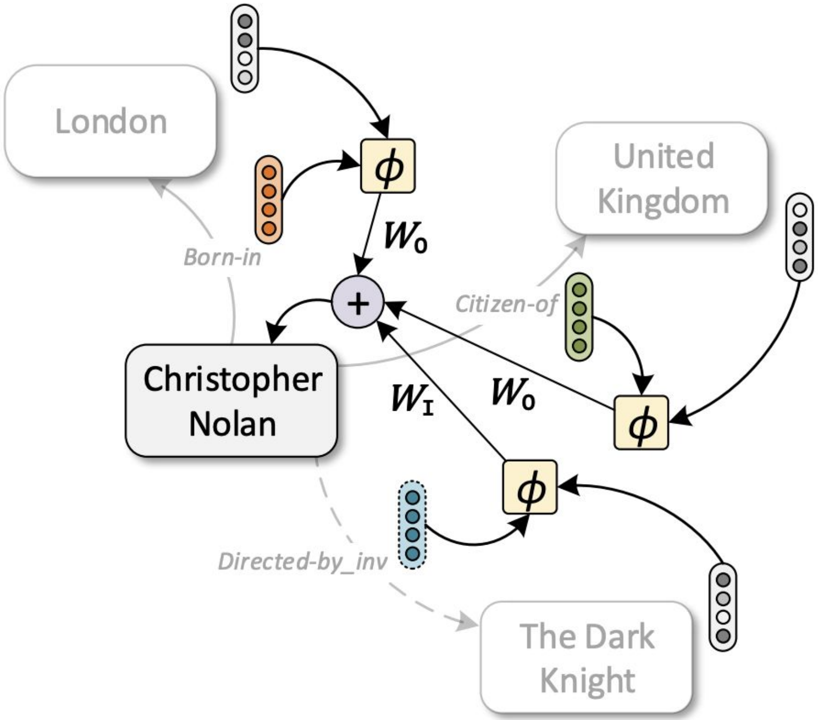


SGCN

- Simplifying Graph Convolutional Networks
- Majority of the benefit arises from the local averaging
- Power of GCNs originates primarily from the repeated graph propagation
- Reduces complexity through removing the nonlinearities
- Does not negatively impact accuracy

CompGCN

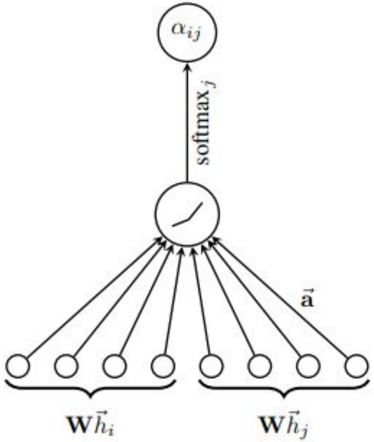
- Composition-based Multi-Relation GCN
- Uses inverse Edges
- Uses Embeddings for Relations and Entities
  - Using seperate weight matrix for relations
  - Access direct relation representations
  - Facing over-parameterization
- Special: Using subgraphs as batches

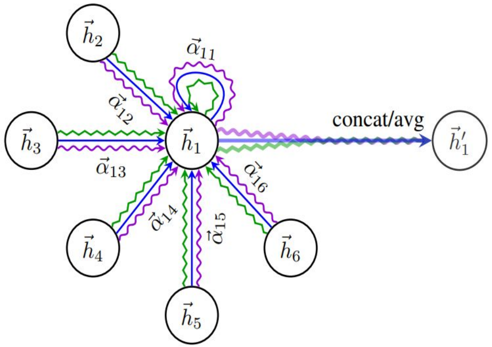


CompGCN Update

CompGCN with Graph Attention Layer Model

- Take the idea of "Attention", to give each neighbour node unique weights
- Take the strategy of mask graph attention.
  - Calculate attention coefficient with neighbour
  - Multi-head Aggregation
- Benefits comparing with basic GCN
  - Can do inductive job
  - Give different weights to neighbourhoods





Results

Custom Split & Ranking

- OGB dataset split
  - Temporal split (September/October/November)
  - No access to testing data solution

- Custom dataset split
  - Random split from OGB training data
  - Same split percentages as the OGB split

- Evaluation metric: Mean Reciprocal Rank (MRR)

Model Name	OGB Validation MRR	Custom Validation MRR	Custom Training MRR
Entity Co-Occurrence	0.0030	0.0123	0.0122
Pseudo Typing	0.2280	0.1569	0.1569
Entity Co-Occurrence + Pseudo Typing	0.2281	**0.1663**	**0.1662**
Entity & Relation Features	**0.4649**	-	-
MLP Embeddings	-	0.1485*	-
ComplEx with PyTorch BigGraph	-	-	0.0340*
CompGCN	-	-	-
CompGCN with Graph Attention	-	-	-
SGCN	-	-	-
Random Model	0.0029 (50 runs)	0.0029 (50 runs)	0.0029 (50 runs)

* intermediary results on subset

Conclusion & Lessons Learned

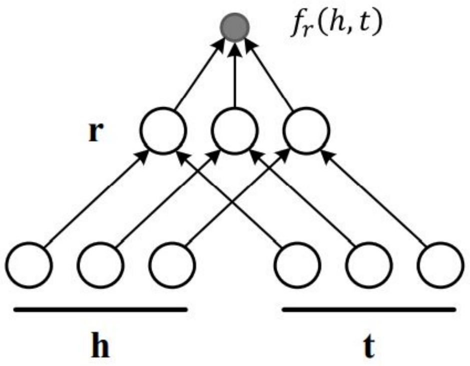
- Working with large datasets is hard
  - Matrices, embeddings etc. get very large very quickly
  - Working memory must be managed efficiently
  - Frequent movement between CUDA and working memory
  - Project management and better training strategy are important

- Best performing model uses features
  - However, most models are still being trained and are improving
  - Difficult to draw conclusions currently.

We are happy to take your questions now!

Appendix

Score-Function for ComplEx Model: $f_r(h,t) = \mathrm{Re}\left(\mathbf{h}^\top \mathrm{diag}(\mathbf{r})\bar{\mathbf{t}}\right) = \mathrm{Re}\left(\sum_{i=0}^{d-1} [\mathbf{r}]_i \cdot [\mathbf{h}]_i \cdot [\bar{\mathbf{t}}]_i\right)$



DistMult

Appendix

Mean Reciprocal Rank:

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\mathrm{rank}_i}$$