# PyTorch-BigGraph: A Large-Scale Graph Embedding System

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, Alex Peysakhovich [1]
Facebook AI Research, SysML 2019

Presenter: Derrick Blakely

May 13, 2019

University of Virginia

https://qdata.github.io/deep2Read/

# Table of contents

# Introduction

# Relationship Graphs

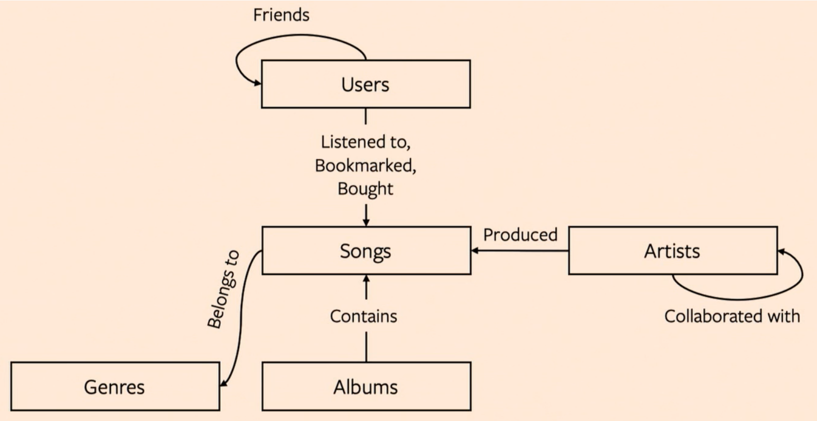



**Figure 1:** *A graph with entities (users, songs, etc) connected by relations (listened to, is friends with, etc.).*

# Graph Embedding

- Embedding: learned map from entities to vectors that encode similarity between entities
- Word embeddings: word $\rightarrow$ vector
- Graph embedding: node $\rightarrow$ vector
- Intuition: connected nodes should be more similar than unconnected nodes
- Not the same GNNs

# Why Graph Embeddings?

- Unsupervised
- Task-agnostic node representations
- Features can be used for downstream tasks
- Nearest neighbors are semantically meaningful

# Graph Embedding Model

# Graph Embedding Formalism

Graph $G = (V, R, E)$

- $V$: nodes aka entities
- $R$: relations
- $E$: edges; $e \in E = (s, r, d) =$ (source, relation, entity)

Score function for each edge:

- Vectors of parameters for each entity and relation type: $\theta_s, \theta_r, \theta_d$
- Score function $f(\theta_s, \theta_r, \theta_d)$

## Score Functions

$$f(\theta_s, \theta_r, \theta_d) = sim\left(g_{(s)}(\theta_s, \theta_r), g_{(d)}(\theta_d, \theta_r)\right)$$

Composed of two parts:

1. Relation operator $g$: linear transformation, translation, complex multiplication
2. Similarity function *sim*: dot product or cosine similarity

# Several Different Score Functions

| Model | Scoring Function | Relation parameters | $\mathcal{O}_{time}$ | $\mathcal{O}_{space}$ |
|---|---|---|---|---|
| RESCAL (Nickel et al., 2011) | $e_s^T W_r e_o$ | $W_r \in \mathbb{R}^{K^2}$ | $\mathcal{O}(K^2)$ | $\mathcal{O}(K^2)$ |
| TransE (Bordes et al., 2013b) | $\|\|(e_s + w_r) - e_o\|\|_p$ | $w_r \in \mathbb{R}^K$ | $\mathcal{O}(K)$ | $\mathcal{O}(K)$ |
| NTN (Socher et al., 2013) | $u_r^T f(e_s W_r^{[1..D]} e_o + V_r \begin{bmatrix} e_s \\ e_o \end{bmatrix} + b_r)$ | $W_r \in \mathbb{R}^{K^2 D}, b_r \in \mathbb{R}^K$ $V_r \in \mathbb{R}^{2KD}, u_r \in \mathbb{R}^K$ | $\mathcal{O}(K^2 D)$ | $\mathcal{O}(K^2 D)$ |
| DistMult (Yang et al., 2015) | $< w_r, e_s, e_o >$ | $w_r \in \mathbb{R}^K$ | $\mathcal{O}(K)$ | $\mathcal{O}(K)$ |
| HolE (Nickel et al., 2016b) | $w_r^T(\mathcal{F}^{-1}[\overline{\mathcal{F}[e_s]} \odot \mathcal{F}[e_o]]))$ | $w_r \in \mathbb{R}^K$ | $\mathcal{O}(K \log K)$ | $\mathcal{O}(K)$ |
| ComplEx | $\mathrm{Re}(< w_r, e_s, \bar{e}_o >)$ | $w_r \in \mathbb{C}^K$ | $\mathcal{O}(K)$ | $\mathcal{O}(K)$ |

**Figure 2:** Score functions shown in "Complex Embeddings for Simple Link Prediction" [2]

| Model | $\mathbf{g(x, \theta_r)}$ | $\mathbf{sim(a, b)}$ |
|---|---|---|
| RESCAL | $A_r x$ | $< a, b >$ |
| TransE | $x + \theta_r$ | $cos(a, b)$ |
| DistMult | $x \odot \theta_r$ | $< a, b >$ |
| ComplEx | $x \odot \theta_r$ | $Re\{< a, \bar{b} >\}$ |

**Figure 3:** Score functions supported by PBG

# Graph Embedding Scoring

Intuition: maximize $f(\cdot)$ for edges that exist and minimize $f(\cdot)$ for edges that don't exist.

$\implies$ Margin/hinge objective:

$$\mathcal{L} = \sum_{e \in G} \sum_{e' \in S'_e} \max\left(f(e) - f(e') + \lambda, 0\right)$$

- $f(e) = cos(\theta_s, \theta_r + \theta_d)$ – score for an actual edge in the graph
- $f(e') = cos(\theta_s, \theta_r + \theta_d)$ – score for an edge that isn't in the graph (aka, a negative sample)
- $\lambda$: margin hyperparameter
- Minimum value is 0

# Their Hinge Loss

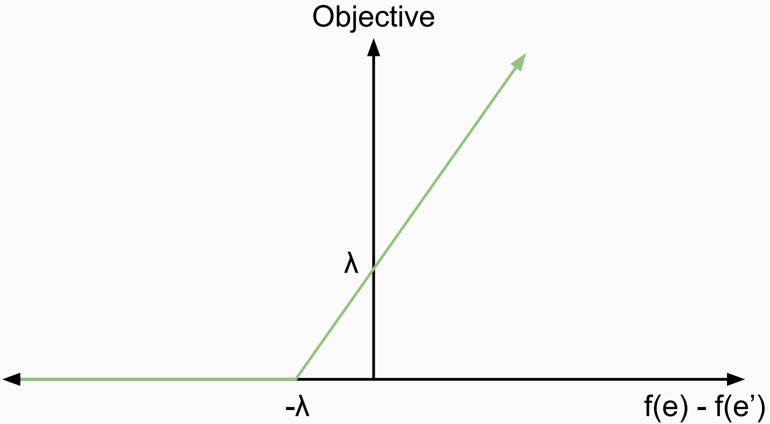



**Figure 4:** *If* $f(e) - f(e') \leq -\lambda, \mathcal{L} = 0$, *if* $f(e) - f(e') > -\lambda, \mathcal{L} = f(e) - f(e')$

# Constructing Negative Samples

Take a real edge and replace the source or destination with a random node.

$$S'_e = \{(s', r, d) | s' \in V\} \cup \{(s, r, d') | d' \in V\}$$

$$e' \leftarrow S'_e$$

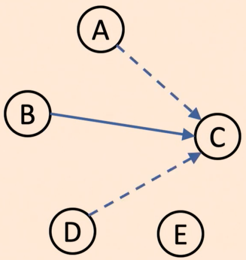
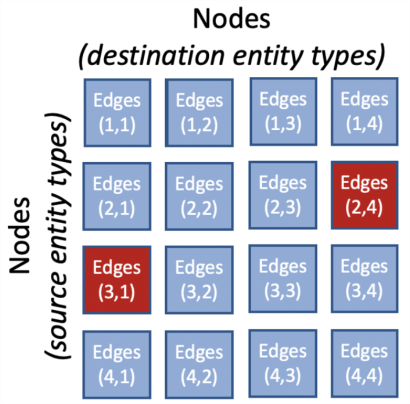



**Figure 5:** *Creating a negative sample by replacing source B with either A or D, while leaving the relation unchanged.*

# PBG System

# Partitioning

- Nodes divided in $N$ shards; edges divided into $N^2$ buckets
- Single machine: 2 partitions used at a time; others swapped to disk
- Distributed training: buckets with disjoint partitions trained simultaneously



# Embedding Quality Loss

Two problems:

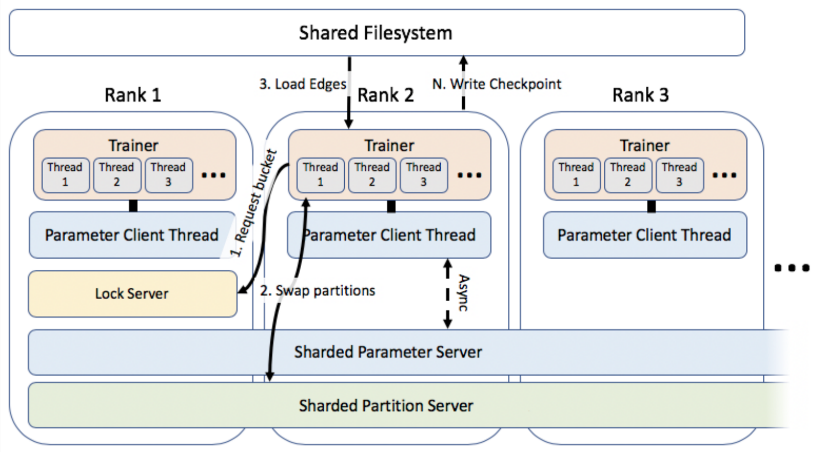
1. If you don't sample edges i.i.d, convergence is slower
2. Partitioning changes distribution of negative samples

# Distributed Training

Three types of communication need to happen:

1. Synchronizing bucket accesses
2. Exchanging partitions
3. Sharing common parameters

# PBG System Overview



# Batched Negative Sampling

- 10-100 negative samples per real edge
- Training time dominated by negative samples
- Solution: corrupt a batch of 100 edges with the same set of random nodes

# Batched Negative Sampling Advantages

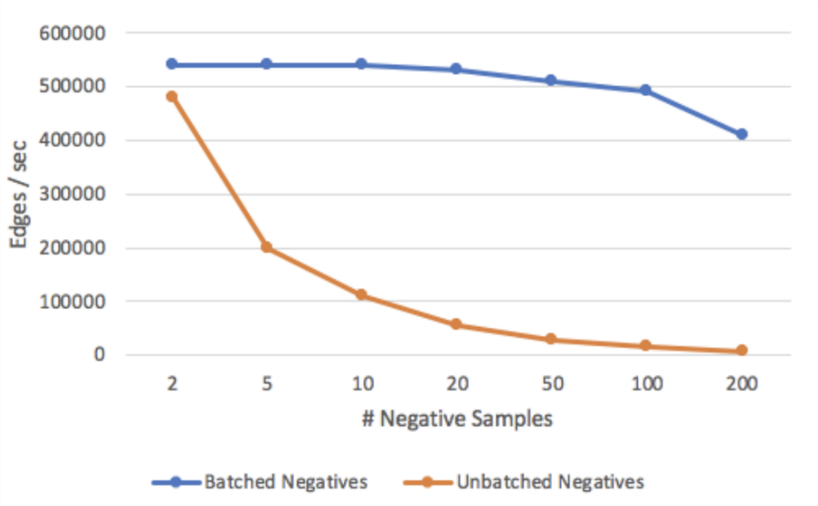
1. Reduce random-access memory bandwidth by a factor of 100
2. Use matrix multiplications to compute $f(\cdot)$

# Experiments

# Datasets

- LiveJournal user-user interaction graph
- Twitter user-user interaction graph
- Youtube user-user interaction graph
- FreeBase Wikipedia knowledge graph

# Negative Batching

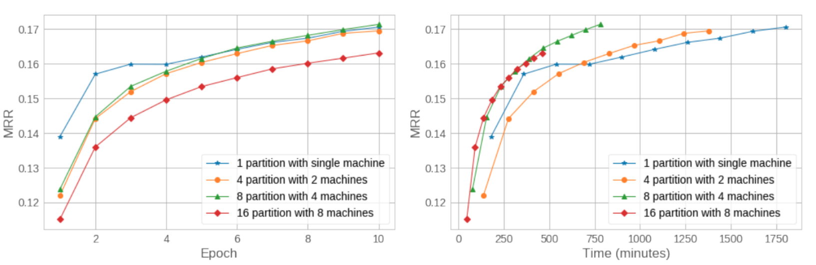


# FreeBase

| # Parts | MRR | Hits@10 | Time (h) | Mem (GB) |
|---|---|---|---|---|
| 1 | 0.170 | 0.285 | 30 | 59.6 |
| 4 | 0.174 | 0.286 | 31 | 30.4 |
| 8 | 0.172 | 0.288 | 33 | 15.5 |
| 16 | 0.174 | 0.290 | 40 | 6.8 |

| # Machines | # Parts | MRR | Hits@10 | Time (h) | Mem (GB) |
|---|---|---|---|---|---|
| 1 | 1 | 0.170 | 0.285 | 30 | 59.6 |
| 2 | 4 | 0.170 | 0.280 | 23 | 64.4 |
| 4 | 8 | 0.171 | 0.285 | 13 | 30.5 |
| 8 | 16 | 0.163 | 0.276 | 7.7 | 15.0 |

# FreeBase



0.17
0.16
0.15
0.14
0.13
0.12
MRR
2
4
6
8
10
Epoch
1 partition with single machine
4 partition with 2 machines
8 partition with 4 machines
16 partition with 8 machines
0
250
500
750
1000
1250
1500
1750
Time (minutes)

# Conclusion

# Strengths

- Can handle very large graphs
- Good partitioning strategy
- Convincing experiments
- Probably actually works pretty well (FB uses it, lots of GitHub starts)

# Weaknesses

- Their partitioning causes embedding quality degradation
- Only intended for unsupervised graph embedding
- Doesn't benefit from GPUs (intended for CPU only)
-

# References i


A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich.
**Pytorch-biggraph: A large-scale graph embedding system.**
*arXiv preprint arXiv:1903.12287*, 2019.

T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard.
**Complex embeddings for simple link prediction.**
In *International Conference on Machine Learning*, pages 2071–2080, 2016.