Marius: Learning Massive Graph Embeddings on a Single Machine

Jason Mohoney and Roger Waleffe, *University of Wisconsin-Madison;*
Henry Xu, *University of Maryland, College Park;* Theodoros Rekatsinas and Shivaram Venkataraman, *University of Wisconsin-Madison*

Review by Kliment Serafimov
For 6.827
April 26th 2022

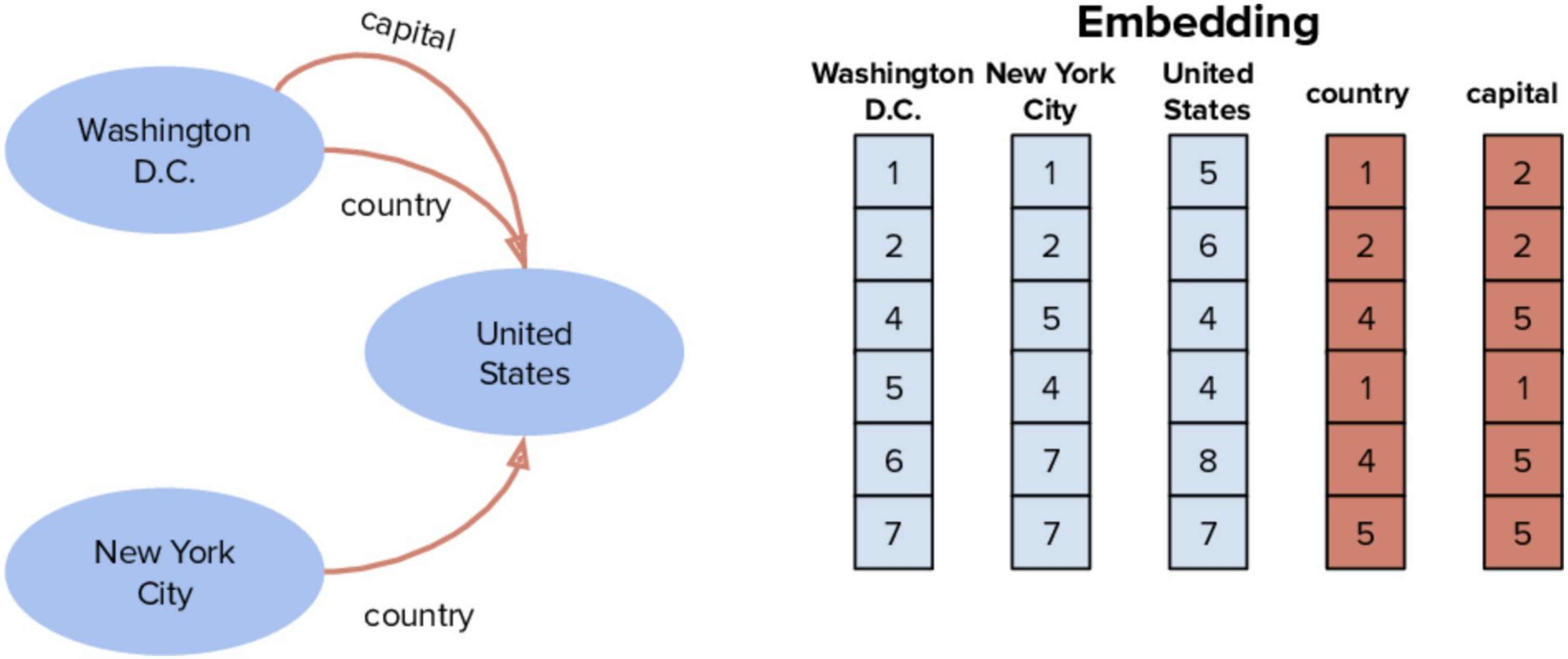


Figure 2. Starting from a knowledge graph, embedding methods generate representations of the elements of the knowledge graph that are embedded in a vector space. For example, these representations could be vectors… Expand

Published in Knowledge Graphs for eXplainable Artificial Intelligence 2020

Knowledge Graph Embeddings and Explainable AI

Federico Bianchi, Gaetano Rossiello, Luca Costabello, M. Palmonari, Pasquale Minervini

Problem and Motivation: Learning Graph Embeddings

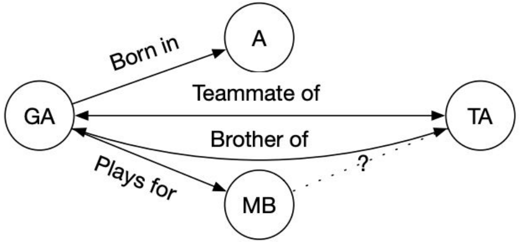


Figure 2: A sample knowledge graph.

- Learning graph embeddings is widely used in industry for a variety of downstream tasks.
- This paper focuses on **link prediction.**

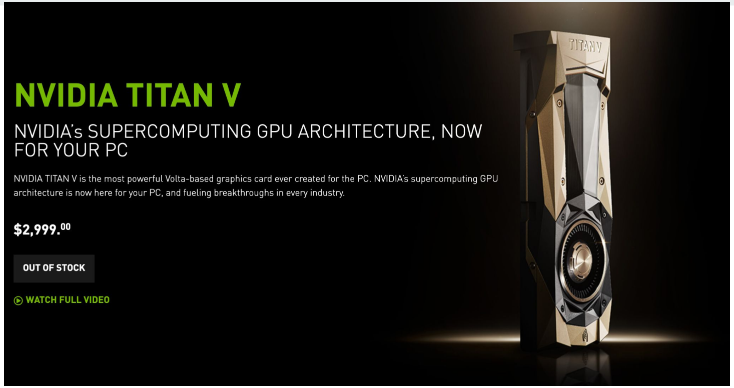
Set up:

- Parameters: 1 high-dimensional vector \in R^400 per node and per edge type.
- Training: minimize contrastive loss:
- Example f(θ_s, θ_r, θ_d) = $\theta_s$diag(θ_r)θ_d; makes f(e) ~ 1 if e \in E, ~ 0 otherwise
- Second term is approximated.

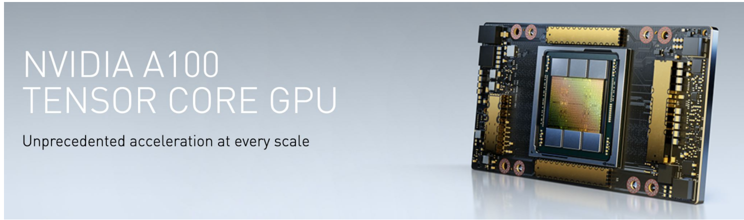
$$\mathcal{L} = -\sum_{s,r,d \in E} \left(f(\mathbf{e}_\theta) - \log\Big(\sum_{s',r',d' \notin E} e^{f(\mathbf{e}_\theta{}')}\Big)\right) \qquad (1)$$

where $\mathbf{e}_\theta = (\theta_s, \theta_r, \theta_d)$ and $\mathbf{e}_\theta{}' = (\theta'_s, \theta'_r, \theta'_d)$.

Scaling issues

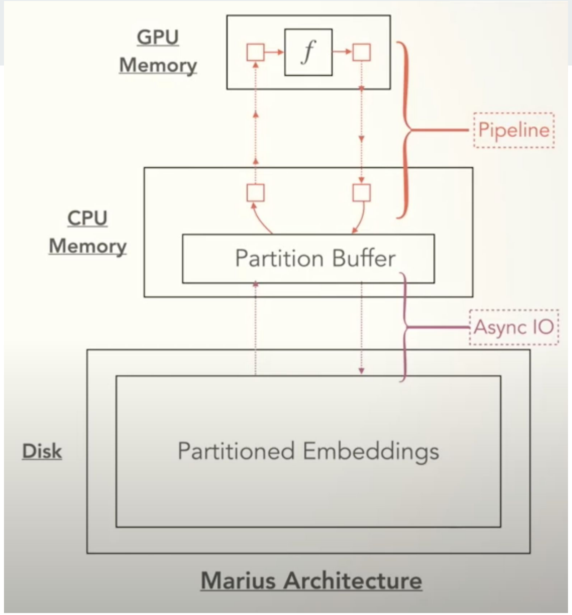


- Very memory intensive!
  - For a graph of 100M nodes, and 400 floats per embedding vector, 4 bytes per float:
  - ~160GB! Just for the node embeddings.
  - Highest GPU memory: 80 GB available (NVIDIA A100)
  - Modest GPU: 12GB (NVIDIA TITAN V)
- Key challenge:
  - How do you facilitate data movement to and from GPU?
  - CPU RAM <--> GPU? (Amazon's DGL-KE, 2020)
  - DISK <--> GPU? (PyTorch BigGraph, 2019)
  - Distributed GPU Compute? (both ^)
  - How to overcome memory bottleneck and IO bottleneck?



Solution!

- Pipelining for hiding IO overhead
- Disk <--> CPU RAM (used as cache) <--> GPU
  - Partition nodes
  - Load and store partitions with smart replacement schedule



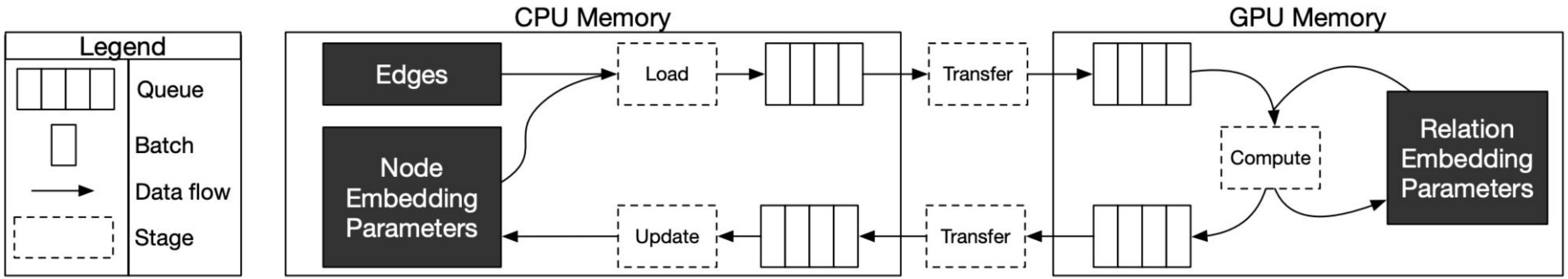


Figure 4: Marius training pipeline.

Partitions

- Nodes are partitioned into p partitions
- To calculate loss, need to cycle through p^2 pairs of partitions.

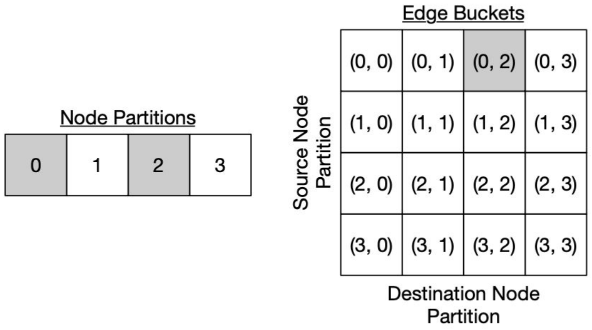


Figure 3: Partitions and edge-buckets with $p = 4$. All edges in edge-bucket (0, 2) have a source node in node-partition 0 and a destination node in node-partition 2.

Replacement Schedule Walkthrough

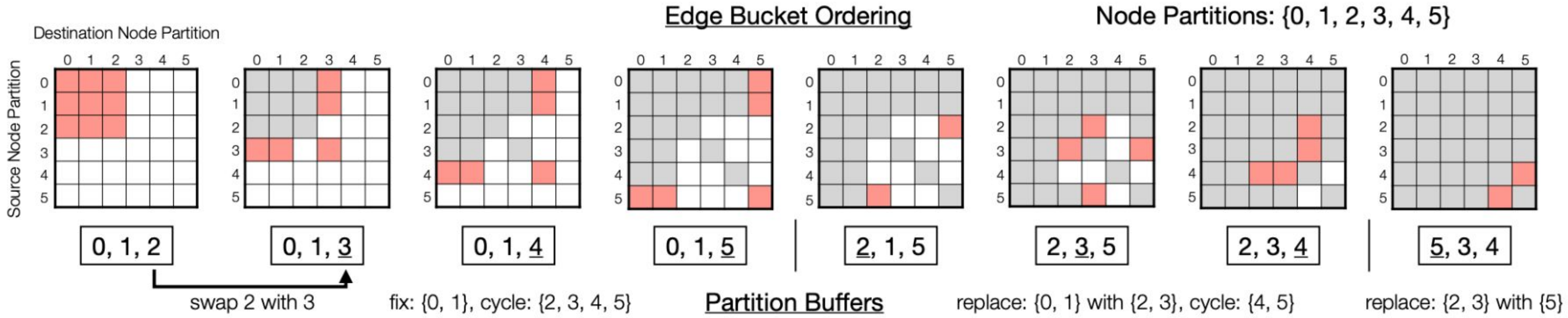


Figure 5: Example *BETA* ordering for $p = 6$ and $c = 3$. The sequence of partition buffers corresponds to first fixing {0, 1}, then replacing {0, 1} with {2, 3}, fixing {2, 3}, and finally replacing {2, 3} with {5}. Each successive buffer differs by one swap. A corresponding edge bucket ordering is shown above the buffers. For each partition buffer in the sequence, all previously unprocessed edge buckets which have their source and destination node partitions in the buffer are added to the ordering (red edge buckets). For each buffer, these edge buckets can be added in any order.

Results

See paper.

Discussion

- Exposing a general interface for implementing similar algorithms?
- Machine-aware self-configuring implementation?