A Massively Scalable Architecture for Learning Representations from Heterogeneous Graphs

NVIDIA GPU Technology Conference 2019 - San Jose, CA

C. Bayan Bruss
Anish Khazane
1. Overview & Background
2. Our Approach
3. Results

TODAY’S TALK
How to handle heterogeneity in training large graph embedding models
Who we are

Bayan Bruss

Anish Khazane
SECTION ONE: OVERVIEW

A quick background on graph embeddings & some of the issues related to scaling them
People are can be disproportionately attracted to content that is sensational or provocative.
Machine learning systems that learn how to serve content are prone to optimizing towards these types of content.
Some common problems and solutions

1. If this is a problem with content (spam, violent, racist, homophobic, etc.)
   - Flag & demote content that is deemed objectionable

2. If this is a problem with users (fake accounts, malicious actors)
   - Eliminate fraudulent accounts
What’s missing?
Basic mechanics of a neural network recommender
Basic mechanics of a neural network recommender
Basic mechanics of a neural network recommender
Basic mechanics of a neural network recommender
Basic mechanics of a neural network recommender
Basic mechanics of a neural network recommender
Basic mechanics of a neural network recommender
How can we add more fidelity to these models?

1. Treat heterogeneous graphs as containing distinct element types

2. Model interactions depending what type of entity is involved
A brief history of graph embeddings

Most Common Objective:
- Learn a continuous vector for each node in a graph that preserves some local or global topological features about the neighborhood of that node

Early Efforts Focused on Explicit Matrix Factorization
- Not very scalable
- Highly tuned to specific topological attributes
Meanwhile over in the language modeling world

Word2Vec world blows things open


Quickly ported to graph embeddings

Walks on a graph can be likened to sentences in a document
Quickly ported to graph embeddings

Walks on a graph can be likened to sentences in a document
Quickly ported to graph embeddings

Walks on a graph can be likened to sentences in a document
Quickly ported to graph embeddings

Walks on a graph can be likened to sentences in a document
Quickly ported to graph embeddings

Walks on a graph can be likened to sentences in a document
Quickly ported to graph embeddings

Walks on a graph can be likened to sentences in a document
Quickly ported to graph embeddings

Walks on a graph can be likened to sentences in a document
Quickly ported to graph embeddings

Walks on a graph can be likened to sentences in a document

[“D”, “B”, “A”, “F”]

[“F”, “C”, “F”, “E”]
Walks on graphs can be treated as sentences

[“D”, “B”, “A”, “F”]

[“F”, “C”, “F”, “E”]
Graphs are different from language

Graph's can be much bigger

Google News Corpus Vocabulary

Size

3000000
2000000
1000000
0
Graphs are different from language

Graph's can be much bigger

- Google News Corpus Vocabulary
- Products on Amazon
Graphs are different from language

Graph's can be much bigger

- Google News Corpus Vocabulary
- Products on Amazon
- Facebook Accounts

Size
Graphs can be heterogeneous
All this makes scale an even bigger challenge
Homogeneous graphs are difficult

**Dimensionality:** Millions or even billions of nodes

**Sparsity:** Each node only interacts with a small subset of other nodes
Quickly hit limits on all resources

1) An embedding space is a N X D dimensional matrix where each row corresponds to a row.
2) D is typically 100 - 200 (an arbitrary hyperparameter)
3) A 500M node graph would be 200 - 400 GB
4) Cannot hold in GPU memory
5) Quickly exceeds limits of a single worker
6) Lots of little vector multiplication ideal for GPUs
7) Sharding because of connectedness - sharding the matrix is challenging
Heterogeneous graphs are even harder

Have to keep K possible embedding spaces with N nodes for each

Have to have an architecture that routes to the right embedding space
It’s also hard from an algorithmic perspective

We’re working on this too but not the focus of today’s talk

See interesting articles

- Metapath2Vec: Scalable Representation Learning for Heterogeneous Networks
- CARL: Content-Aware representation Learning for Heterogeneous Graphs
SECTION TWO: OUR APPROACH

Applied Research: An architecture for handling heterogeneity at scale
Quick Primer on Negative Sampling

Original SkipGram Model

Need to compute softmax over entire vocabulary for each input
Quick Primer on Negative Sampling

Original SkipGram Model

VERY EXPENSIVE!
Softmax can be approximated by binary classification task

Original SkipGram Model

Binary Discriminator

- \( w(t-2) \) vs negative samples
- \( w(t-1) \) vs negative samples
- \( w(t+1) \) vs negative samples
- \( w(t+2) \) vs negative samples
Use non-edges to generate negative samples

Negatives for B

[“F”, “C”, “F”]

[“D”, “B”, “A”, “F”]

Context for B

[“F”, “C”, “F”, “E”]
Walking on heterogeneous graph
How to distribute (parallelize) training

1. Split the training set across a number of workers that execute in parallel asynchronously and unaware of the existence of each other.

2. Create some form of centralized parameter repository that allows learning to be shared across all the workers.
Parameter server partitioning

- A parameter server can hold the embeddings table which contains the vectors corresponding to each node in the graph.

- The embeddings table is a N x M table, where N is the number of nodes in the graph and M is a hyperparameter that denotes the number of embedding dimensions.
Variable Tensorflow Computational Graphs

- Memory (Global graph)
  - Batch subgraphs retrieved from global graph
  - 0 -> 1, 2, 3
  - 1 -> 4, 5, 6
- Worker 0, Worker 1, Worker N - 1, Worker N
- Dynamically compute Tensorflow graph for every batch of data
- Instantiate nodes as tensorflow variables by using their embedding representations from the parameter server
- Parameter Server
SECTION THREE: RESULTS
Capital One Heterogeneous Data

Node Type A: 18,856,021
Node Type B: 32,107,404
Total Nodes: 50,963,425
Edges: 280,422,628

Train Time: 3 Days on 28 workers
Friendster Graph

Publicly available dataset
68,349,466 vertices (users)
2,586,147,869 edges (friendships)
Sampled 80 positive and 5 * 80 negative edges per node as training data.
The data was shuffled, split into chunks and distributed across workers
Friendster Graph
Friendster Graph
Implications

Scalability:
- More nodes per entity type
- More entity types

Convergence:
- Faster as number of workers increases
Limitations and Future Directions

Limitations

• Python performance

• Not partitioning the embedding space

• Recomputing the computational graph for each batch could be optimized

Future Directions

• Evaluate c++ variant of architecture

• Intelligent partitioning of graph so that each worker gets a component of the graph and only has to go to the server for small subset of nodes in other components
THANK YOU