CS 267
Lecture 17: Parallel Machine Learning Part 2

Unsupervised and Semi-Supervised Learning

Aydin Buluc

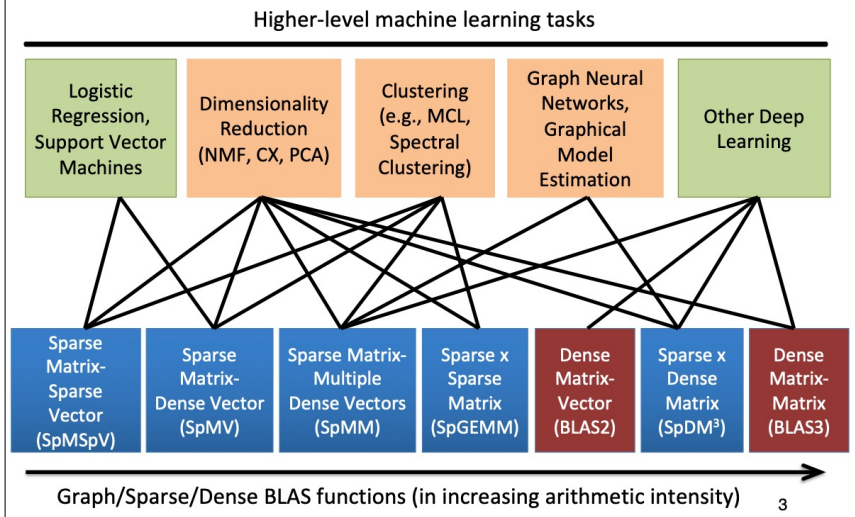
https://sites.google.com/lbl.gov/cs267-spr2021/

Outline of the parallel ML lectures

Today: Part 1, Intro and Supervised Learning
- Machine Learning & Parallelism Intro
- Neural Network Basics
- Deep Neural Network Training
- Support Vector Machines

Tuesday: Part 2, Unsupervised and Semi-Supervised Learning
- *Clustering*: Spectral Clustering and Markov Clustering
- *Dimensionality Reduction*: Non-Negative Matrix Factorization
- *Embeddings*: Node (of a graph) Embedding
- *Semi-supervised learning*: Graph Neural Networks

Machine Learning relies heavily on Linear Algebra



Clustering

Many families of methods
- Centroid based (k-means, k-medians, and variations)
- Flow based (Markov clustering)
- Spectral methods
- Density based (DBSCAN, OPTICS)
- Agglomerative methods (single linkage clustering)
- …

- Often the right method depends on the input characteristics and require some domain knowledge.
- We will talk about parallel algorithms for two: **Spectral Clustering** and **Markov Clustering (MCL).**

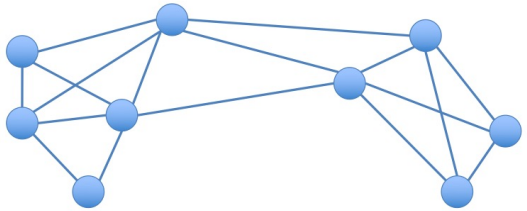
Spectral Clustering

° **Input:** Similarities between data points

° Many ways to compute similarity, some are domain specific: cosine, Jaccard index, Pearson correlation, Spearman's rho, Bhattacharyya distance, LOD score, …

° We can represent the relationships between data points in a graph.

° Weight the edges by the similarity between points



Graph definitions

- ε-neighborhood graph
  - Identify a threshold value, ε, and include edges if the affinity between two points is greater than ε.
- k-nearest neighbors
  - Insert edges between a node and its k-nearest neighbors.
  - Each node will be connected to (at least) k nodes.
- Fully connected
  - Insert an edge between every pair of nodes.

Spectral Clustering Intuition

- The minimum cut of a graph identifies an optimal partitioning of the data.
- Spectral Clustering
  - Recursively partition the data set
    - Identify the minimum cut
    - Remove edges
    - Repeat until k clusters are identified
- **Problem**: Identifying a minimum cut is NP-hard.
- There are efficient approximations using linear algebra, based on the Laplacian Matrix, or **graph Laplacian**

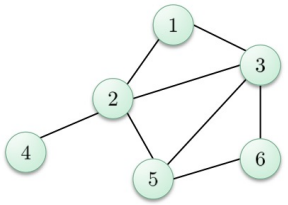
The Graph Laplacian

- Graph Laplacian
  - unnormalized graph Laplacian : $L = D - W$
  - normalized graph Laplacian

$$L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$$

$$L_{rw} = D^{-1} L = I - D^{-1} W \quad \longleftarrow \text{related to random walk}$$

  - Example



$$L = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 & 0 \\ -1 & -1 & 4 & 0 & -1 & -1 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 3 & -1 \\ 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}$$

Assume the weights of edges are 1.

One Spectral Clustering Algorithm

- I recommend the **normalized symmetric Laplacian**, as the numerical eigenvalue problem there is easier to solve.
- Normalized Spectral Clustering [Ng2002]

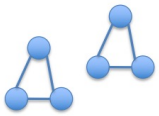
1. Construct a similarity graph and compute the normalized graph Laplacian L_{sym}.
2. Compute the k smallest eigenvectors $u_1, u_2, \cdots, u_k$ of L_{sym}.
3. Let $U = [\, u_1\ u_2\ \cdots\ u_k\,] \in \mathbb{R}^{n\times k}$.
4. Normalized the rows of U to norm 1.

$$U_{ij} \leftarrow \frac{U_{ij}}{(\sum_k U_{ik}^2)^{1/2}}$$

5. Let $y_i \in \mathbb{R}^k$ be the vector corresponding to the ith row of U.
6. Thinking of y_i's as points in $\mathbb{R}^k$, cluster them with k-means algorithms.

Why does it work? One intuitive explanation

- Ideal Case



$L = D\text{-}W$

2	-1	-1	0	0	0
-1	2	-1	0	0	0
-1	-1	2	0	0	0
0	0	0	2	-1	-1
0	0	0	-1	2	-1
0	0	0	-1	-1	2

$Lv = \lambda v$

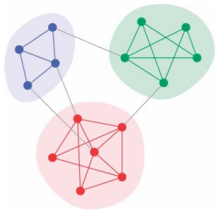
1	0
1	0
1	0
0	1
0	1
0	1

- The multiplicity of the eigenvalue 0 gives the number of clusters (in this ideal case: the number of connected components).
- The real case is assumed to be an approximation to this situation.

How to compute those smallest Eigenvectors?

- Implementation via the **Lanczos Algorithm**
  - Workhorse is **sparse-matrix-vector (SpMV) multiply**
  - SpMV has no/minimal data reuse, bound by communication
  - To optimize sparse-matrix-vector multiply and minimize its communication, we graph partition (next lecture)
- Alternative algorithms are possible
  - Power iteration is cheaper but numerically unstable
  - **LOBPCG** (Locally-Optimized Block Preconditioned Conjugate Gradient) uses **sparse-matrix times multiple vectors**, thus has more favorable performance profile due to possible data reuse.
- In the end, you probably just want to call something existing.
  - ARPACK implements reverse communication eigensolvers: You implement the SpMV, its implements the numerical outer logic
  - PARPACK is its parallel version, The following code uses it: https://github.com/openbigdatagroup/pspectralclustering

Philosophy of the Markov Cluster Algorithm (MCL)



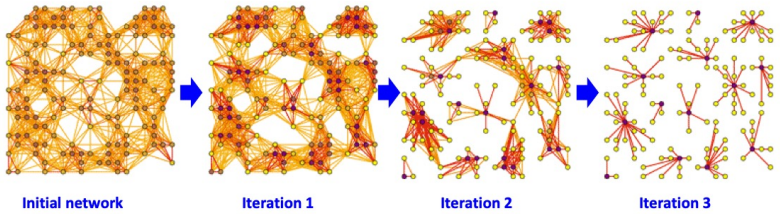
The number of **edges or higher-length paths** between two arbitrary nodes in a cluster is greater than the number of paths between nodes from different clusters

Random walks on the graph will frequently remains within a cluster

The algorithm **computes the probability** of random walks through the graph and **removes lower probability terms** to form clusters

The MCL Algorithm

Input: Adjacency matrix A (sparse & column stochastic)



At each iteration:

Step 1 (Expansion): Squaring the matrix

[corresponds to computing random walks of higher length]

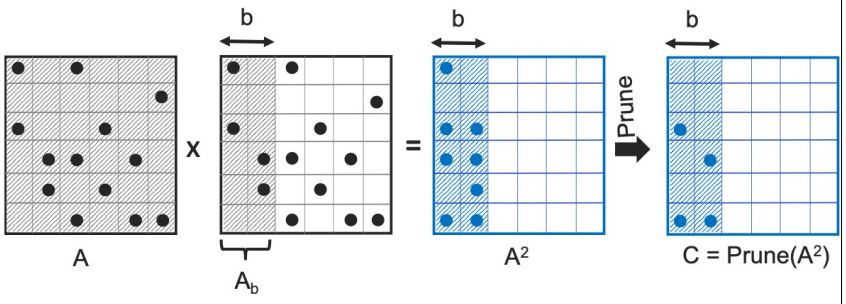
Step 2 (Inflation) : Hadamard power of a matrix (taking powers entrywise)

[boost the probabilities of intra-cluster walks and demote inter-cluster walks]

The expansion step of the MCL algorithm

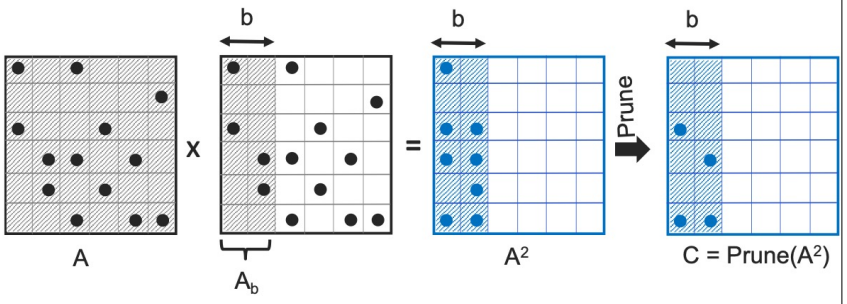
- Goal: Compute random walks of higher length
- Input: A column stochastic matrix (A)
- Algorithm
  1. Sparse matrix-sparse matrix multiplication (**SpGEMM**): A^2
  2. **Sparsify** A^2 by removing low probability terms
     - **Prune** entries in A^2 that are smaller than a threshold
     - **Recover** (if overdone pruning): Keep at least R entries (column-wise top-K selection)
     - **Selection** (if underdone pruning): Sparsify denser columns by keeping at most S entries (column-wise top-K selection)
- After sparsification at most max(R,S) (default to 1400) entries remains in each column of A^2

A combined expansion and pruning step



- b: number of columns in the output constructed at once
  - Smaller b: less parallelism, memory efficient (b=1 is equivalent to sparse matrix-sparse vector multiplication used in MCL)
  - Larger b: more parallelism, memory intensive

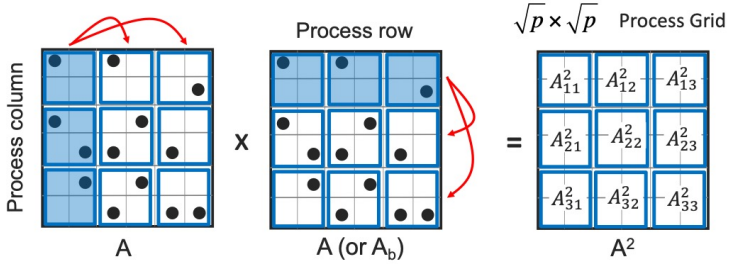
A combined expansion and pruning step



- b: number of columns in the output constructed at once
  - HipMCL selects b dynamically as permitted by the available memory
  - The algorithm works in h=N/b phases where N is the number of columns (vertices in the network) in the matrix

Current sparse matrix-matrix multiply algorithm in HipMCL

- Sparse SUMMA algorithm.
- Do this for each phase.
- Issue: repeated broadcast of A.
- Using communication-avoiding algorithms is faster [1]



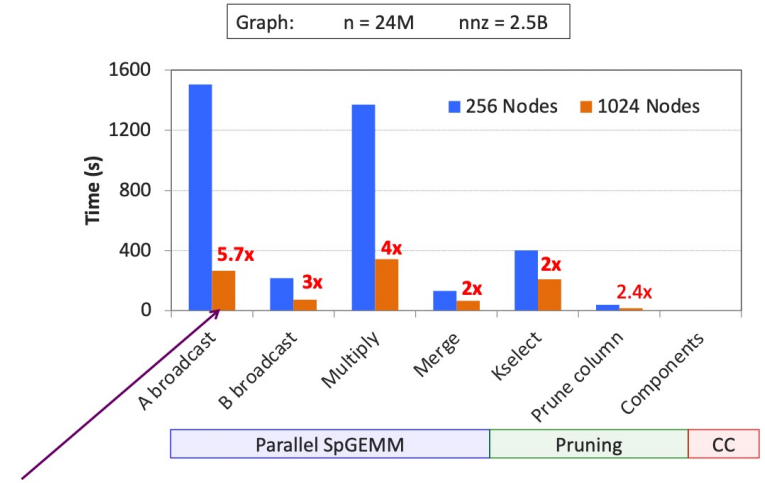
[1] Md Taufique Hussain, Oguz Selvitopi, Aydin Buluç, Ariful Azad. Communication-Avoiding and Memory-Constrained Sparse Matrix-Matrix Multiplication at Extreme Scale. Proceedings of the IPDPS, 2021

Other algorithmic steps of HipMCL

- There is more than sparse matrix multiply here.
  - Parallel k-selection algorithm for each column of the matrix (for Recovery and Selection). We will cover some algorithms in the **sorting and searching** lecture
  - Parallel pruning algorithm
  - Parallel connected component algorithm (to identify clusters after MCL is converged). Very fundamental **graph algorithm**, though we won't cover it this year.
  - Parallel file I/O. Reading terabytes of text efficiently is a challenge.

Azad, A., Pavlopoulos, G.A., Ouzounis, C.A., Kyrpides, N.C. and Buluç, A., 2018. HipMCL: a high-performance parallel implementation of the Markov clustering algorithm for large-scale networks. Nucleic acids research.

Scaling of HipMCL



On 1024 nodes, we need fewer phases because of more aggregated memory

Outline of the parallel ML lectures

Today: Part 1, Intro and Supervised Learning

- Machine Learning & Parallelism Intro
- Neural Network Basics
- Deep Neural Network Training
- Support Vector Machines

Tuesday: Part 2, Unsupervised and Semi-Supervised Learning

- *Clustering*: Spectral Clustering and Markov Clustering
- *Dimensionality Reduction*: Non-Negative Matrix Factorization
- *Embeddings*: Node (of a graph) Embedding
- *Semi-supervised learning*: Graph Neural Networks

Dimensionality Reduction

- Mapping "high-dimensional" data onto a lower dimensional "manifold"
- The high dimensionality of the data might be obvious (e.g., millions of features explicitly collected) or perceived (e.g., images, videos, text)

The mappings can be:

- Linear
  - PCA, NMF, ….
- Non-linear
  - Laplacian eigenmaps*
  - IsoMap
  - ….

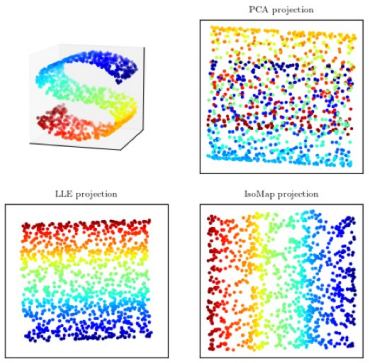


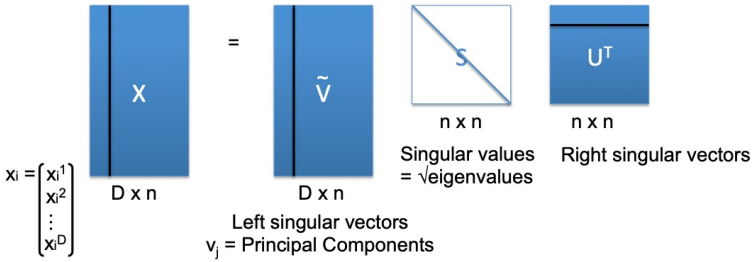
Figure source: "Statistics, Data Mining, and Machine Learning in Astronomy"

Principal Component Analysis (PCA)

Find k unit vectors $v_i \in R^d$ called *Principal Components*, such that:

1. the variance of the dataset projected onto the direction determined by v_i is maximized
2. The vector v_i is chosen to be orthogonal to v_j for $j < i$

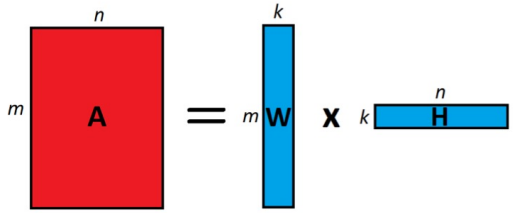
- PCA can be computed with singular value decomposition



$x_i = \begin{bmatrix} x_i^1 \\ x_i^2 \\ \vdots \\ x_i^D \end{bmatrix}$

Non-negative matrix factorization (NMF)

$$\min_{W \geq 0, H \geq 0} f(W,H) = \frac{1}{2}\|A - WH\|_F^2$$



- **Dimensionality reduction** with non-negativity constraints
- The name "factorization" is a misnomer; NMF is just a low-rank approximation as exact factorization is NP-hard
- NMF is a family of methods, not just one algorithm

The Alternating Updates Framework

Initialize H

Repeat until convergence:

1. For fixed H, solve $\min_{W \geq 0,} \|H^T W^T - A^T\|_F^2$
2. For fixed W, solve $\min_{H \geq 0,} \|WH - A\|_F^2$

Lots of algorithms fall into this framework.

- Multiplicative update (MU)
- Alternating least squares (ALS)
- Alternating non-negative least squares (ANLS)

J. Kim and H. Park. "Fast nonnegative matrix factorization: An active-set-like method and comparisons." SIAM Journal on Scientific Computing, 2011

Caveat emptor: This is not the only method for finding an NMF

Gemulla, Rainer, et al. "Large-scale matrix factorization with distributed stochastic gradient descent." KDD, 2011

The Alternating Updates Framework

Main computation is large-scale matrix multiplications:

1. $\mathbf{HH}^T$ and $\mathbf{AH}^T$ for updating **W**, given a fixed **H**
2. $\mathbf{W}^T\mathbf{W}$ and $\mathbf{W}^T\mathbf{A}$ for updating **H**, given a fixed **W**

- In general **W** and **H** are dense, but short-fat or tall-skinny
- **A** is often sparse but can be dense depending on application
- For increased interpretability, **H** or **W** can also be sparse

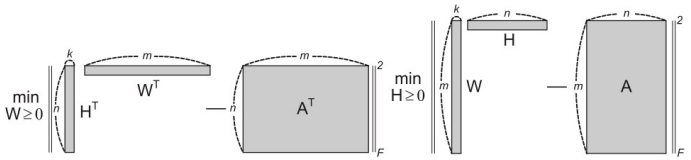


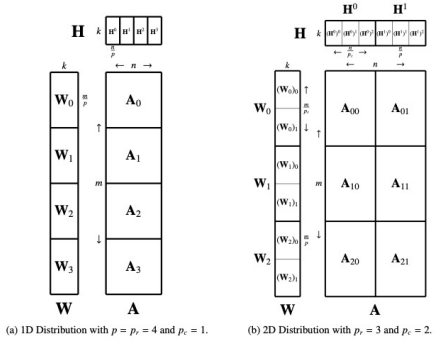
Figure: Kim and Park

The Alternating Updates Framework

Main computation is large-scale matrix multiplications

Choose the best distribution and algorithm depending on:
1- the relative sizes of the dimensions of the matrices
2- the number of processors

This work: Never communicate A, because it is asymptotically larger than H and W



(a) 1D Distribution with $p = p_r = 4$ and $p_c = 1$. (b) 2D Distribution with $p_r = 3$ and $p_c = 2$.

Kannan, Ballard, Park. "MPI-FAUN: An MPI-Based Framework for Alternating-Updating Nonnegative Matrix Factorization". 2016.

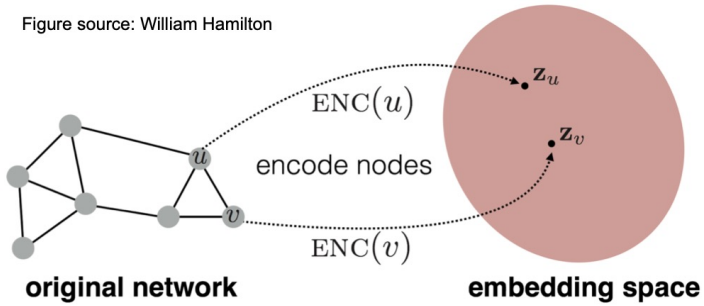
Outline of the parallel ML lectures

Today: Part 1, Intro and Supervised Learning
- Machine Learning & Parallelism Intro
- Neural Network Basics
- Deep Neural Network Training
- Support Vector Machines

Tuesday: Part 2, Unsupervised and Semi-Supervised Learning
- *Clustering*: Spectral Clustering and Markov Clustering
- *Dimensionality Reduction*: Non-Negative Matrix Factorization
- *Embeddings*: Node (of a graph) Embedding
- *Semi-supervised learning*: Graph Neural Networks

What is an embedding?

Figure source: William Hamilton

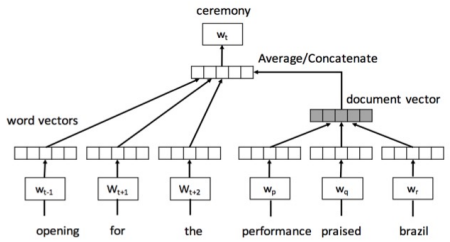


Goal: similarity(u,v) ≈ $\mathbf{z}_u^T \mathbf{z}_v$

Spectral clustering's conversion of the similarity graph into a matrix of k-eigenvectors was an embedding

What is an embedding?

- Arguably (if you consider objects like graphs or text as *high dimensional*), this is also dimensionality reduction
- In theory, you can embed almost any object:
  - node2vec
  - word2vec
  - seq2vec
  - graph2vec
  - ....



Doc2VecC by Minmin Chen

- The actual name does not have to follow this recipe

Factorization-based embeddings

Also known as "shallow" embeddings

- Laplacian eigenmaps et al. [You already know this!]
  - Spectral clustering = Laplacian eigenmaps + k-means clustering
- Graph Factorization et al. [You already know this!]
  - $\min\ (\|A - ZZ^T\|_N + \|reg\|)$
  - N: some norm, reg: regularization (if any), add constraint (if any)
  - Parallelization techniques from NMF apply
  - You can also use Stochastic Gradient Descent (SGD), and use various gradient synchronization techniques from ML part 1 lecture
- All are inherently *transductive* (as opposed to *inductive*), i.e. they can not be used to do inference on new samples
- *Side story*: you can use embeddings or factor matrices as input to many different clustering algorithms

Random walk-based embeddings

If two vertices co-occur on a random walk, they are similar

The loss function:

$$L = -\sum_{u \in V} \sum_{v \in N_R(u)} \log\left(\frac{\exp(z_u{}^T z_v)}{\sum_{n \in V} \exp(z_u{}^T z_v)}\right)$$

Predicted probability of u and v co-occurring on random walks

Doubly-nested loop over all vertices makes this algorithm **O(|V|²)**

Negative Sampling

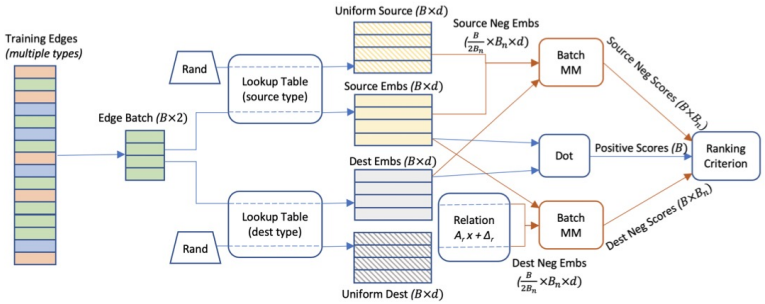
- Negative sampling is used when the data has orders of magnitude more negative samples (e.g., missing edges) than positive ones (present edges)
- Replace the normalization sum: Instead of normalizing w.r.t. all nodes, just normalize against random "negative samples".

$$\frac{\exp(z_u{}^T z_v)}{\sum_{n \in V} \exp(z_u{}^T z_v)} \approx P_{G,T}(v|u)$$

$P_{G,T}(v|u)$ is the probability of visiting v on a length-T random walk starting at u, with T usually defined to be in the range [2-10]

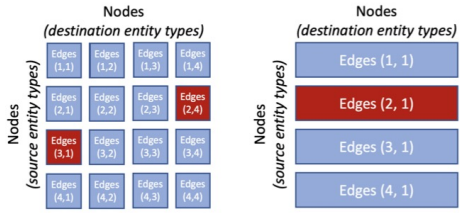
You know how to do random walks on graphs (hint: HipMCL)!

PyTorch-BigGraph



Memory-efficient **batched negative sampling**. Embeddings are fetched for the B source and destination entities in a batch of edges, as well as B uniformly-sampled source and destination entities. Each chunk of $B_n/2$ edges is corrupted with all source or destination entities in its chunk, as well as the corresponding chunk of the uniform embeddings, resulting in B_n negative examples per positive edge. The negative scores are computed via a batch matrix multiply

PyTorch-BigGraph



The PBG partitioning scheme for large graphs.

Left: nodes are divided into P partitions that are sized to fit in memory. Edges are divided into buckets based on the partition of their source and destination nodes. In distributed mode, multiple buckets with non-overlapping partitions can be executed in parallel (red squares).

Right: Entity types with small cardinality do not have to be partitioned; if all entity types used for tail nodes are unpartitioned, then edges can be divided into P buckets based only on source node partitions.

Limits of embeddings

- The graph embeddings described, either based on factorization or random walks, are **task independent**.
- **Upside:** Embed once, use for any tasks.
  - Run any clustering algorithm on R^d embeddings
  - Treat embeddings as data and run traditional deep learning algorithms
- **Downside:** We can not automatically learn to improve the way we embed
  - That requires **backpropagation**
  - Enter graph neural networks*

Outline of the parallel ML lectures

Today: Part 1, Intro and Supervised Learning

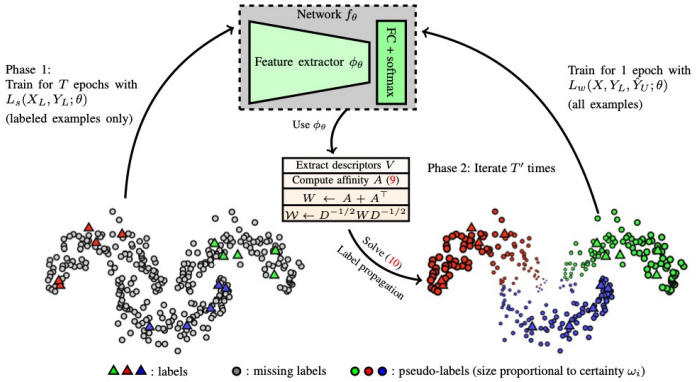
- Machine Learning & Parallelism Intro
- Neural Network Basics
- Deep Neural Network Training
- Support Vector Machines

Tuesday: Part 2, Unsupervised and Semi-Supervised Learning

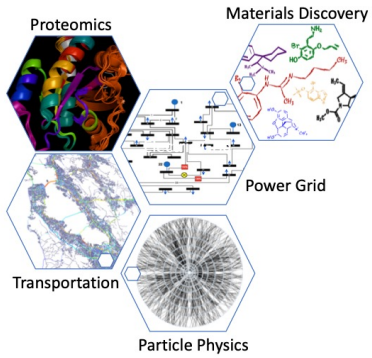
- *Clustering*: Spectral Clustering and Markov Clustering
- *Dimensionality Reduction*: Non-Negative Matrix Factorization
- *Embeddings*: Node (of a graph) Embedding
- *Semi-supervised learning*: Graph Neural Networks

Semi-supervised Learning

- A (small) subset of data has training labels, but most of the data is unlabeled
- *Label propagation* is the canonical graph semi-supervised learning algorithm



Graph Neural Networks (GNNs)



GNNs are finding success in many challenging scientific problems that involve interconnected data.

- Graph classification
- Edge classification
- **Node classification**

GNNs are computationally intensive to train. Distributed training need to scale to large GPU/node counts despite challenging sparsity.

How to use GNNs

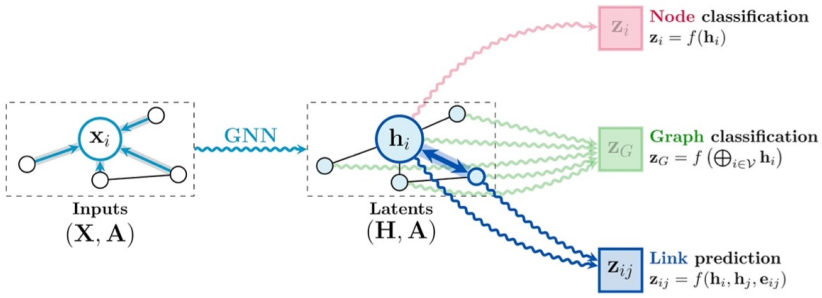
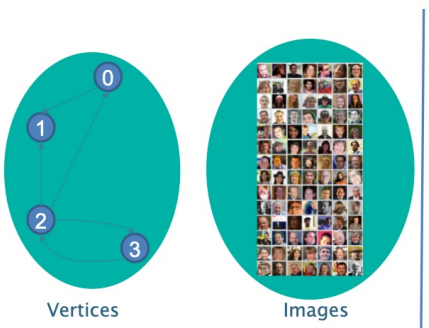


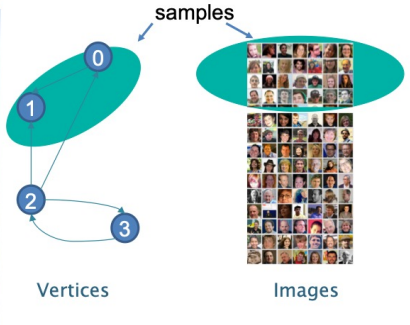
Figure source: Petar Veličković

Full-graph vs. mini-batch SGD



Full-graph training:

- Train on **entire** training set
- Slower convergence per epoch
- Faster training per epoch
- **Focus of this work**



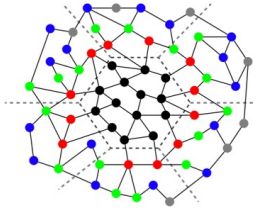
Mini-batch SGD:

- Train on multiple **samples** from training set
- Faster convergence per epoch
- Slower training per epoch
- Requires graph sampling, which effects accuracy and performance

Full-graph vs. mini-batch SGD

No dependencies

sample

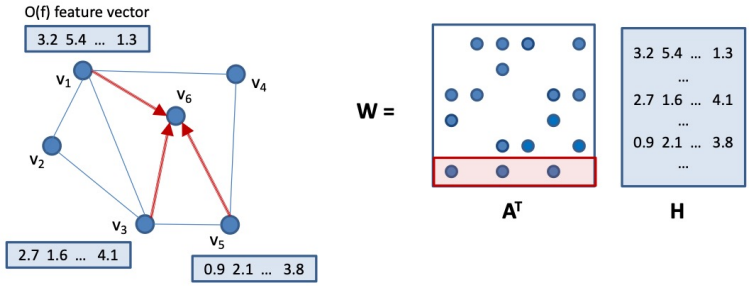


Layered dependencies

- Vertices (unlike images) are dependent on each other
- L-layer GNN uses L-hop neighbors for vertices in batch
- Must store almost the whole graph for any minibatch for power-law graphs
- How to subsample from L-hop neighborhood and keep accuracy?
- CAGNET (Communication-Avoiding Graph Neural nETworks) full gradient descent to avoid such issues: https://github.com/PASSIONLab/CAGNET/

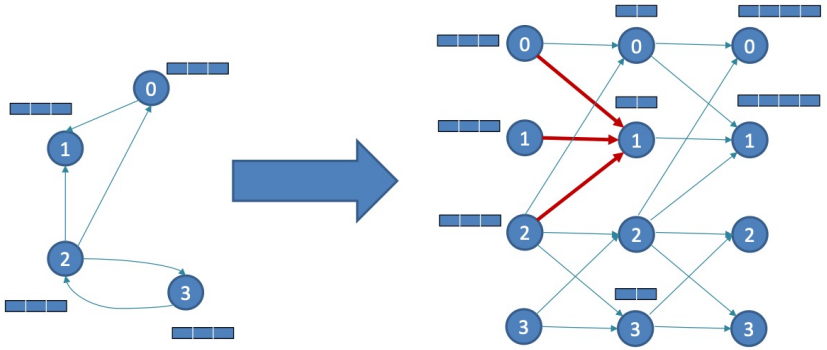
Graph convolutions

Graph convolution: Feature aggregation from neighbors



- GNN is an umbrella term for any neural network that performs graph representation learning.
- CAGNET focuses on Graph Convolutional Networks (GCNs)
- We are working on adding graph attention layers

Graph convolutions



Input Graph

GNN of Input Graph

- Recall that a CNN can have different *channel* dimension at each layer.
- GNNs also have different embedding dimension at each layer

GCN training in matrix notation

Forward Propagation:

$$\mathbf{Z}^l \leftarrow \mathbf{A}^\top \mathbf{H}^{l-1} \mathbf{W}^l$$

$$\mathbf{H}^l \leftarrow \sigma(\mathbf{Z}^l)$$

Backward Propagation:

$$\mathbf{G}^{l-1} \leftarrow \mathbf{A}\mathbf{G}^l (\mathbf{W}^l)^\top \odot \sigma'(\mathbf{Z}^{l-1})$$

$$\mathbf{Y}^{l-1} \leftarrow (\mathbf{H}^{l-1})^\top \mathbf{A}\mathbf{G}^l$$

Symbols and Notations

Symbol	Description
$\mathbf{A}$	Modified adjacency matrix of graph $(n \times n)$
$\mathbf{H}^l$	Embedding matrix in layer l $(n \times f)$
$\mathbf{W}^l$	Weight matrix in layer l $(f \times f)$
$\mathbf{Y}^l$	Matrix form of $\frac{\partial \mathcal{L}}{\partial W^l_{ij}}$ $(f \times f)$
$\mathbf{Z}^l$	Input matrix to activation function $(n \times f)$
$\mathbf{G}^l$	Matrix form of $\frac{\partial \mathcal{L}}{\partial Z^l_{ij}}$ $(n \times f)$
σ	Activation function
f	Length of feature vector per vertex
f_u	Feature vector for vertex u
L	Total layers in GNN
P	Total number of processes
α	Latency
β	Reciprocal bandwidth

GCN training

- Each node is initialized with a feature vector
  - H^0 has initial feature vector per node $(n \times f)$
- Each node aggregates vectors of its neighbors, applies a weight
- Each layer computes gradients

```
for i = 1 … E
    for l = 1 … L
        Z^l = A^T * H^(l-1) * W^l
        H^l = σ(Z^l)
    ...
    for l = L-1 … 1
        G^l = A * G^(l+1) * (W^(l+1))^T ⊙ σ'(Z^l)
       dH/dW = (H^(l-1))^T * A * G^l
```

$A \in n \times n$

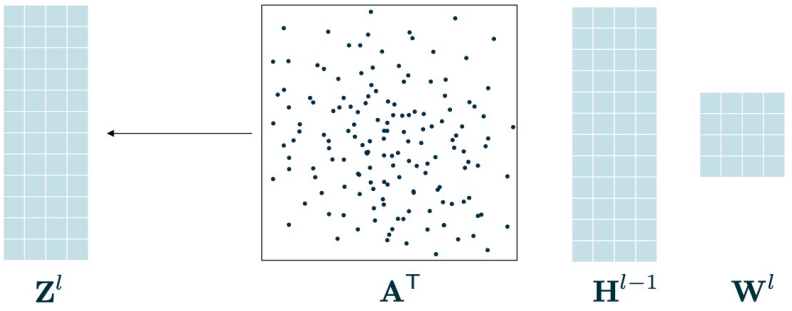
$H^l \in n \times f^l$

$G^l \in n \times f^l$

$W^l \in f^{l-1} \times f^l$

- A is sparse and f << n, so the main workhorse is SpMM (sparse matrix times tall-skinny dense matrix)

Bottleneck of GCN training

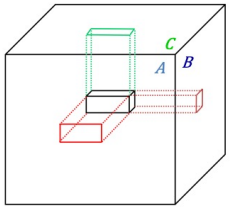


Cost(SpMM) >>> Cost(DGEMM)

(mostly because W is so small)

The computation cube of matrix-matrix multiplication

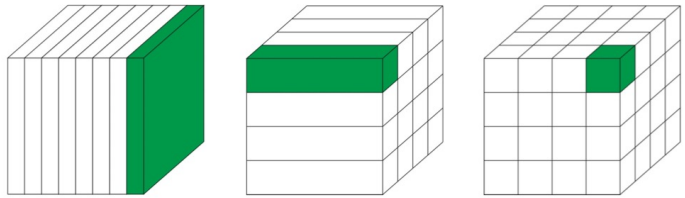
Matrix multiplication: $\forall(i,j) \in n \times n, \quad C(i,j) = \Sigma_k A(i,k)B(k,j),$



The *computation (discrete) cube*:

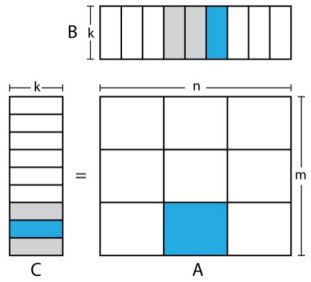
- A face for each (input/output) matrix
- A grid point for each multiplication

.5D algorithms interpolate between two



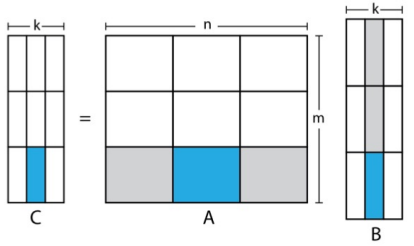
1D algorithms 2D algorithms 3D algorithms

Distributed SpMM algorithms



- Stationary A, 1.5D algorithm
- **A** is split on a p/c-by-c grid

A is sparse, **B** and **C** are dense



- Stationary C, 2D algorithm
- Memory optimal

- 1D algorithm not shown, degeneration of sA-1.5D for the c=1 case
- Right before reduction, sA-1.5D uses c times more dense-matrix memory

Distributed SpMM algorithms

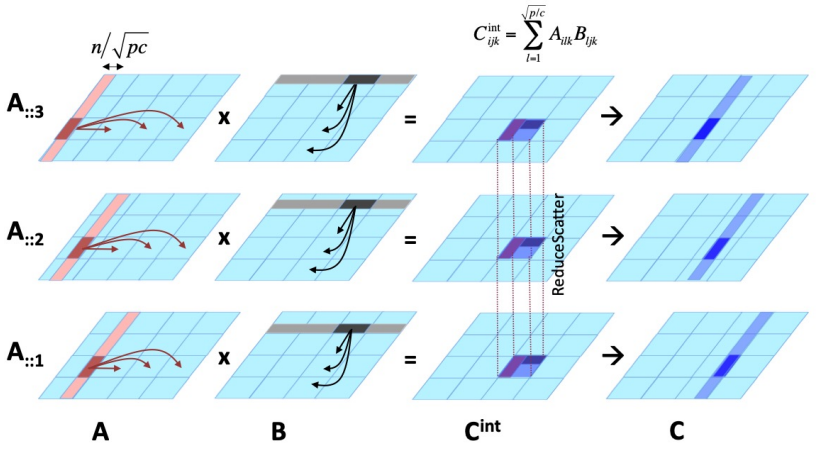


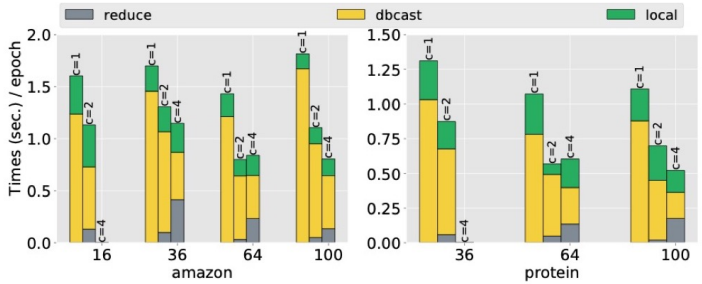
Illustration of the 3D algorithm on a $\sqrt{p/c} \times \sqrt{p/c} \times c$ process grid

Communication analysis

CAGNET Cost Analyses (per process)			
Algorithm	Latency	Bandwidth	Memory
1D	$\lg P + 2P$	$2nf + f^2$	$\frac{nnz(\mathbf{A})+nfL}{P}$
1.5D	$2\frac{P}{c^2}\lg\frac{P}{c^2}$	$\frac{2nf}{c} + \frac{2nfc}{P}$	$\frac{nnz(\mathbf{A})+nfL}{P} + \frac{nfc}{P}$
2D	$5\sqrt{P} + 3\lg P$	$\frac{8nf}{\sqrt{P}} + \frac{2nnz(\mathbf{A})}{\sqrt{P}}$	$\frac{nnz(\mathbf{A})+nfL}{P}$
3D	$4P^{1/3}$	$\frac{2nnz(\mathbf{A})}{P^{2/3}} + \frac{12nf}{P^{2/3}}$	$\frac{nnz(\mathbf{A})+nfL}{P} + \frac{nfc}{P}$

Symbols and Notations	
Symbol	Description
$\mathbf{A}$	Modified adjacency matrix of graph $(n \times n)$
$\mathbf{H}^l$	Embedding matrix in layer l $(n \times f)$
$\mathbf{W}^l$	Weight matrix in layer l $(f \times f)$
$\mathbf{Y}^l$	Matrix form of $\frac{\partial \mathcal{L}}{\partial W^l_{ij}}$ $(f \times f)$
$\mathbf{Z}^l$	Input matrix to activation function $(n \times f)$
$\mathbf{G}^l$	Matrix form of $\frac{\partial \mathcal{L}}{\partial Z^l_{ij}}$ $(n \times f)$
σ	Activation function
f	Length of feature vector per vertex
f_u	Feature vector for vertex u
L	Total layers in GNN
P	Total number of processes
α	Latency
β	Reciprocal bandwidth

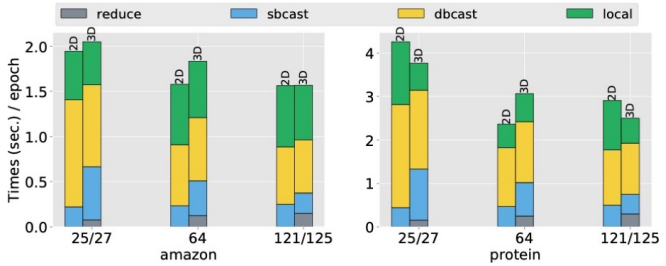
Communication avoidance (CA) in GNN Training



- Scales with both P (GPUs – x axis) and c (replication layers in CA algorithms)
- This is 1 GPU/node on Summit (all GPUs per node results in paper)
- Expect to scale with all GPUs / node with future architectures (e.g. Perlmutter)

Alok Tripathy, Katherine Yelick, Aydın Buluç. Reducing Communication in Graph Neural Network Training. SC'20

2D vs. 3D performance



- 64 hidden-layer activations
- Communication scales with P, consistent with analysis
- Computation scales less well → explained in paper

Alok Tripathy, Katherine Yelick, Aydın Buluç. Reducing Communication in Graph Neural Network Training. SC'20

Parallel GNN training conclusions

- Graph representation learning is transforming science
  - » Lots of deep learning problems on graphs
- Can solve DL on graphs with GNNs
  - » But must distribute training
- Alok's work
  - » Can formulate GCN training as SpMM
  - » Distribute GCN training with distributed SpMM
  - » Code: https://github.com/PASSIONLab/CAGNET
- Future work
  - » Distributed sampling for mini-batch training