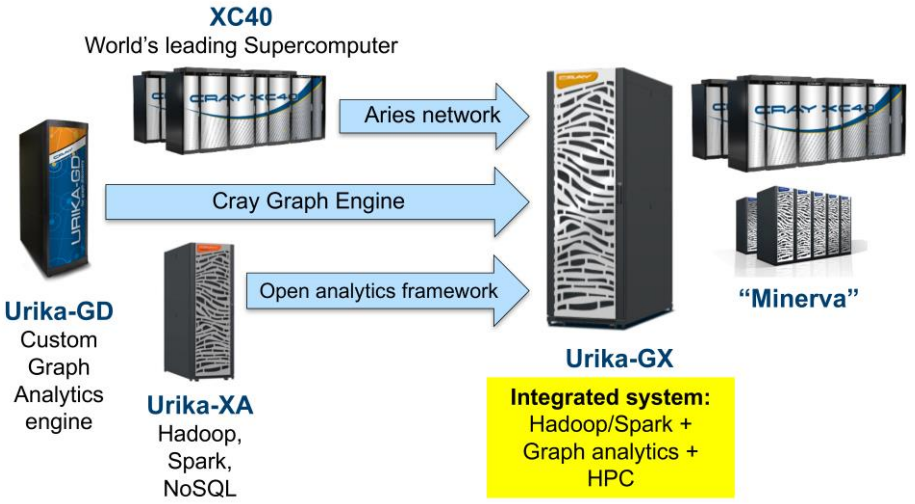
Lecture 12:
Machine Learning

Simulation + Data + Learning

- Data analytics and machine learning increasingly important in scientific discovery
  - Event identification, correlation in high-energy physics
  - Climate simulation validation using sensor data
  - Determine patterns and trends from astronomical data
  - Genetic sequencing

- **The convergence of simulation, data, and learning**
  - current hot topic: workshops, conferences, research initiatives, funding calls

- Driving changes in supercomputer architecture
  - Multiprecision hardware
  - Specialized accelerators
  - Memory at node



Outline of the lecture

Intro and Supervised Learning

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

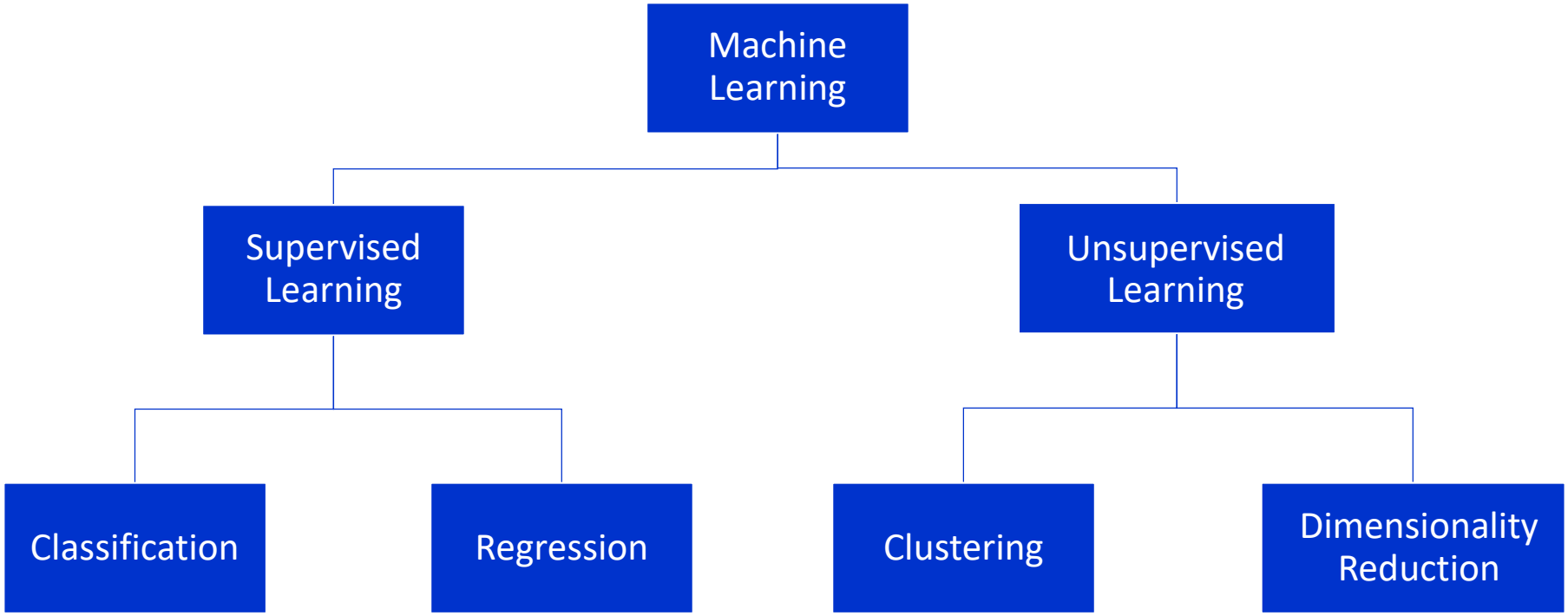
Unsupervised Learning

- Non-Negative Matrix Factorization
- Spectral and Markov Clustering
- Sparse Inverse Covariance Matrix Estimation

Machine Learning Classes and Tasks

The central question in Machine Learning:

"How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?"



Machine Learning Sources of Confusion

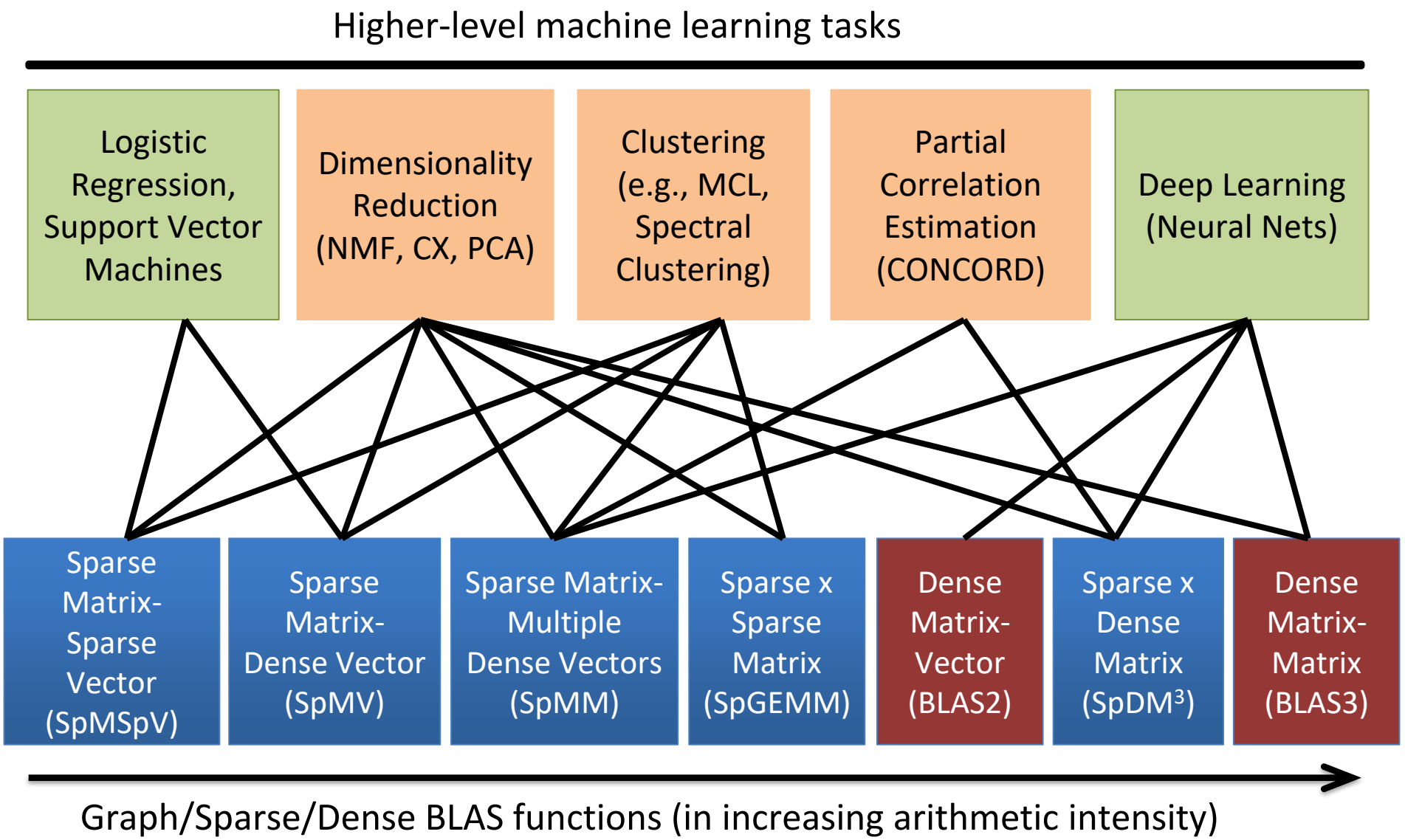
Method vs. Task: A common confusion is between specific learning methods and learning tasks.

- Example #1: Principal Component Analysis is a method for dimensionality reduction task
- Example #2: Support Vector Machines are methods used for supervised learning tasks.

Another confusion comes from ***optimization techniques*** vs. **learning methods**.

- Example #1: Sequential Minimal Optimization is an optimization technique to train Support Vector Machines
- Example #2: Stochastic Gradient Descent is a popular optimization technique to train Neural Networks.

Machine Learning relies a lot on Linear Algebra



Parallelism in Machine Learning

Implicit Parallelization: Keep the overall algorithm structure (the sequence of operations) intact and parallelize the individual operations.

Example: parallelizing the BLAS operations in previous figure

+ Often achieves exactly the same accuracy (e.g., model parallelism in DNN training)

- Scalability can be limited if the critical path of the algorithm is long

Explicit Parallelization: Modify the algorithm to extract more parallelism, such as working on individual pieces whose results can later be combined

Examples: CA-SVM and data parallelism in DNNs

+ Significantly better scalability can be achieved

- (Maybe) no longer the same algorithmic properties

Outline of the lecture

Intro and Supervised Learning

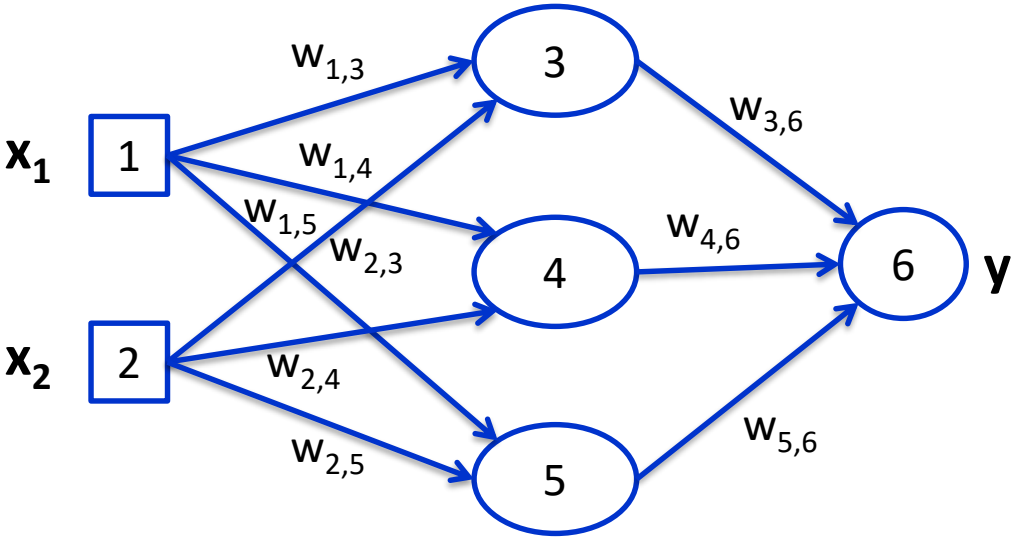
- Machine Learning & Parallelism Intro
- Neural Network Basics
- Support Vector Machines

Unsupervised Learning

- Non-Negative Matrix Factorization
- Spectral and Markov Clustering
- Sparse Inverse Covariance Matrix Estimation

Training Neural Networks

- Training is to adjust the weights (W) in the connections of the neural network, in order to change the function it represents.



Only parameters are weights for simplicity (i.e. ignore bias parameters)

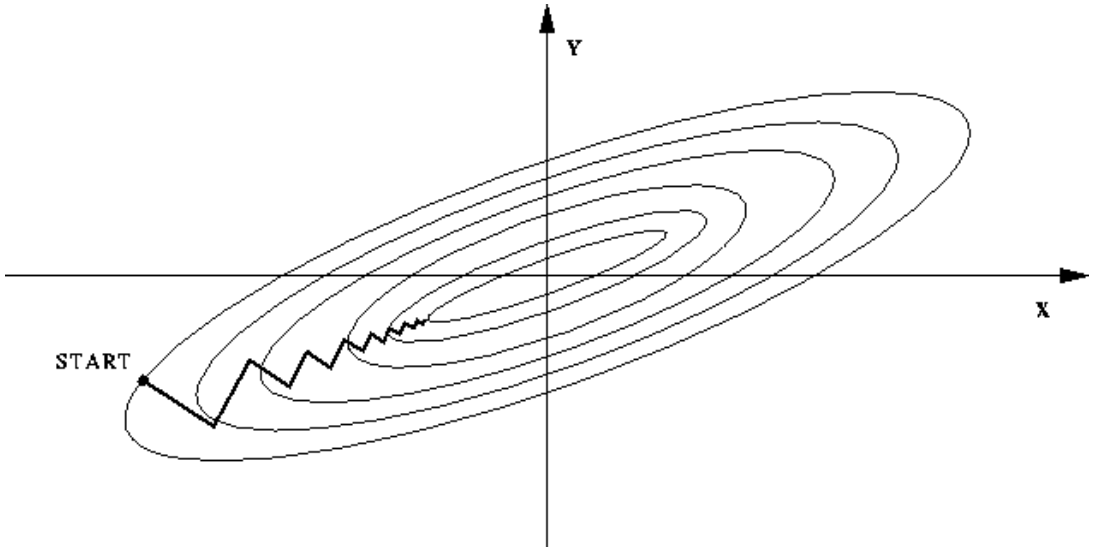
W: the matrix of weights

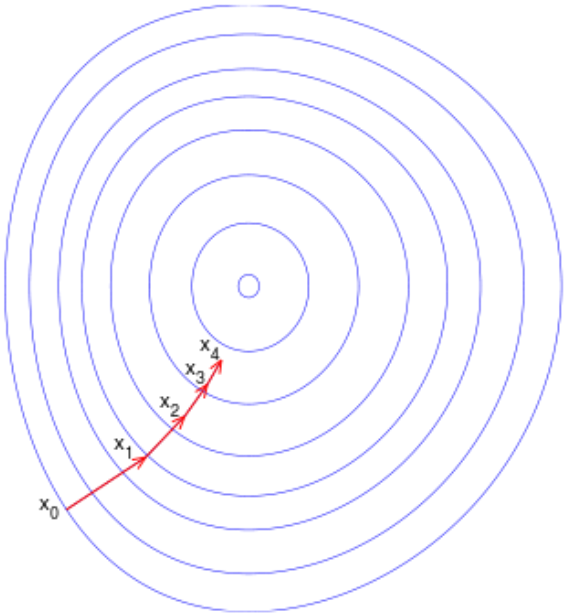
A "shallow" neural network with only one hidden layer (nodes 3,4,5), two inputs and one output.

Gradient Descent

$$W^{t+1} \leftarrow W^t - a \cdot \nabla_W f(W^t, x)$$

- Also called the steepest descent algorithm
- In order to minimize a function, move towards the opposite direction of the gradient at a rate of α.
- α is the step size (also called the learning rate)
- Used as the ***optimization backend*** of many other machine learning methods (example: NMF)





Stochastic Gradient Descent (SGD)

$$\text{Assume } f(W^t, x) = \frac{1}{n}\sum_{i=1}^{n} f_i(W^t, x)$$

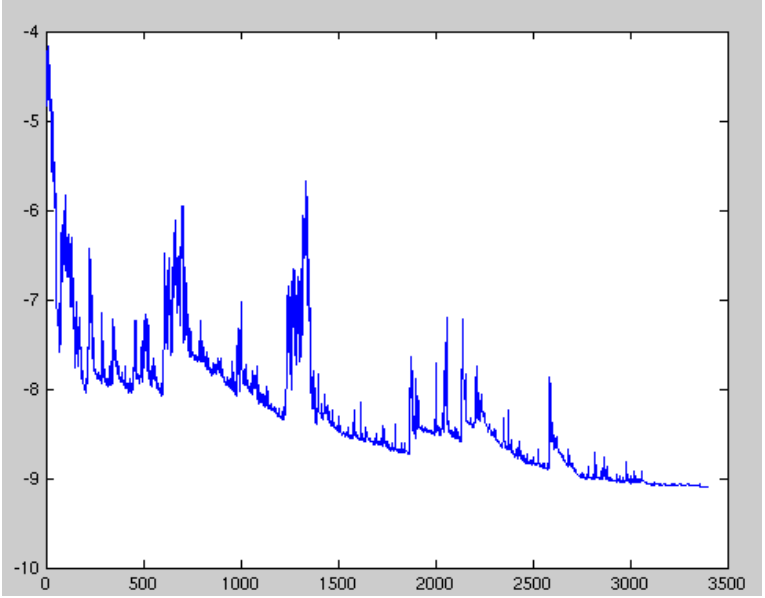
$$W^{t+1} \leftarrow W^t - a \cdot \nabla_W f_i(W^t, x)$$

Pure SGD: compute gradient using 1 sample

$$W^{t+1} \leftarrow W^t - a \cdot \frac{1}{b}\sum_{i=k+1}^{k+b} \nabla_W f_i(W^t, x)$$

Mini-batch: compute gradient using b samples

f is not going down for every iteration

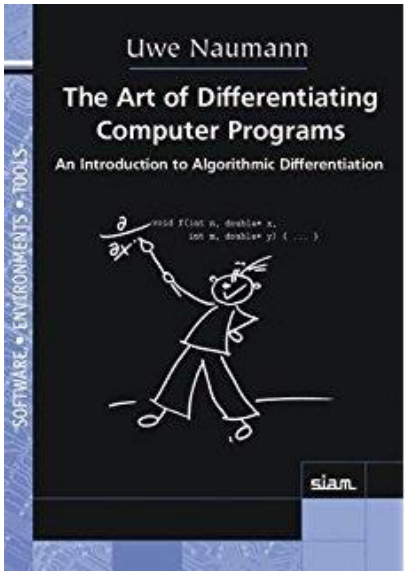


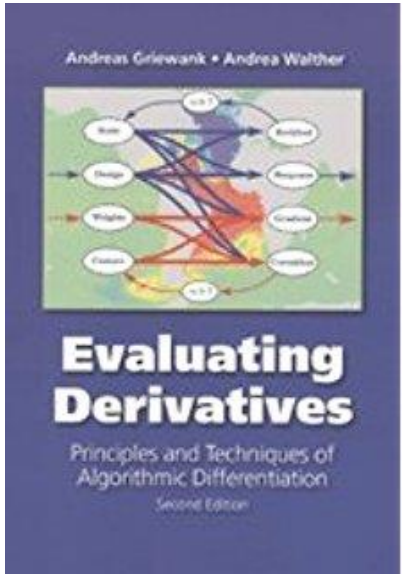
- Actually the name is a misnomer, *this is not a "descent" method*
- But we will stick to it anyway to avoid confusion.
- Performance and parallelism requires batch training
- Larger batch sizes hurt convergence as they get trapped easily
- SGD escapes sharp local minima due to its "noisy" gradients

Training Neural Networks

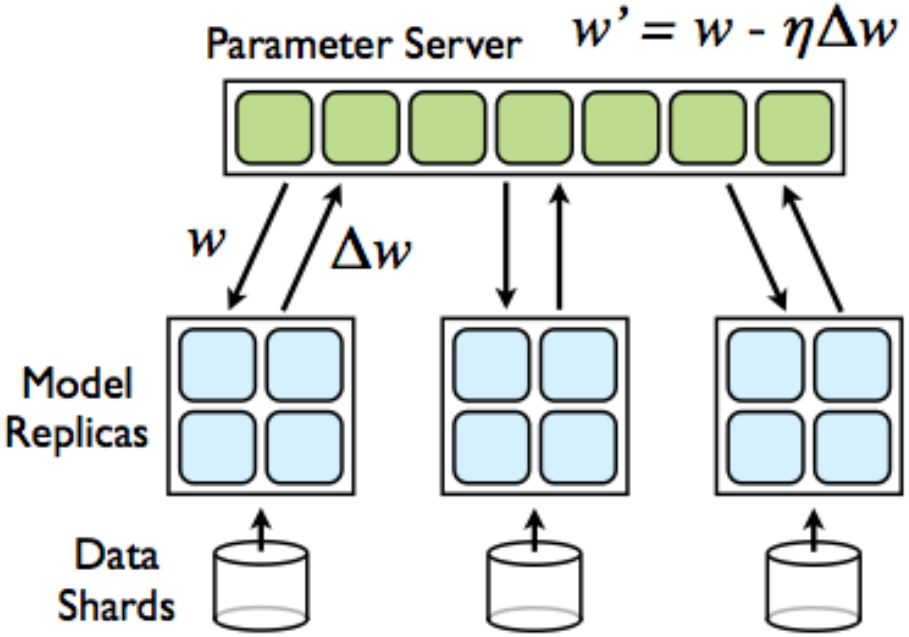
- Training is performed using an optimization algorithm like SGD
- **SGD needs derivatives.**
- The algorithm ***to compute derivatives*** on a neural network is called ***back-propagation***.
- The back-propagation algorithm is ***not a training algorithm***
- **Idea:** Repeated application of the chain rule from calculus

Back-propagation is just a special case of the ***reverse mode automatic/algorithmic differentiation***





Data Parallelism #1



Parameter server is some sort of master process

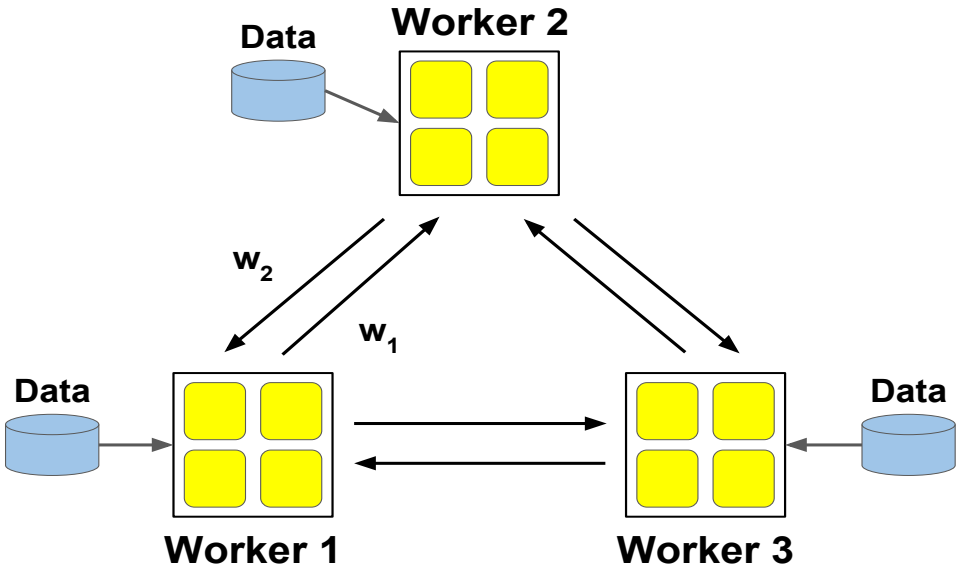
- The fetching and updating of gradients in the parameter server can be done either ***synchronously*** or ***asynchronously***.
- Both has pros and cons. Over-synchronization hurts performance where asynchrony is not-reproducible and might hurt convergence

Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

Data Parallelism #2

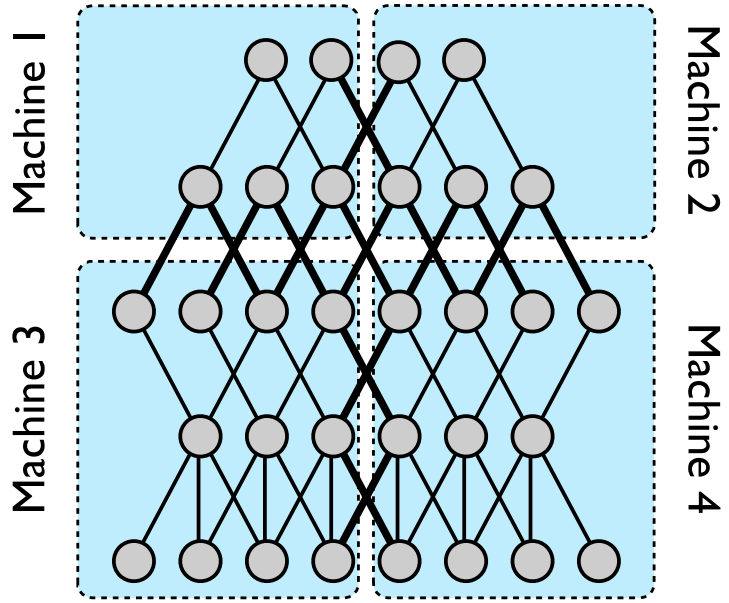
Options to avoid the parameter server bottleneck

1. **For synchronous SGD:** Perform all-reduce over the network to update gradients (good old MPI_Allreduce)
2. **For asynchronous SGD:** Peer-to-peer gossiping



Peter Jin, Forrest Iandola, Kurt Keutzer, "How to scale distributed deep learning?" NIPS ML Sys 2016

Model Parallelism

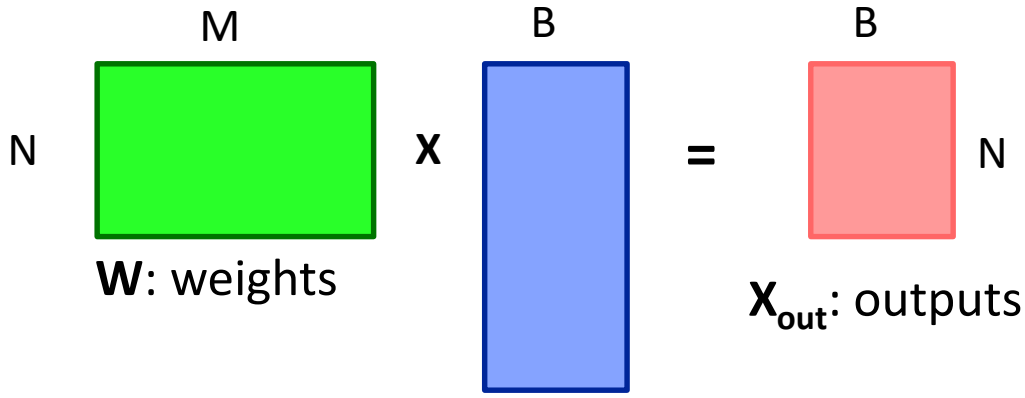


Interpretation #1: Partition your neural network into processors

Interpretation #2: Perform your matrix operations in parallel

Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

SGD training of NNs as matrix operations

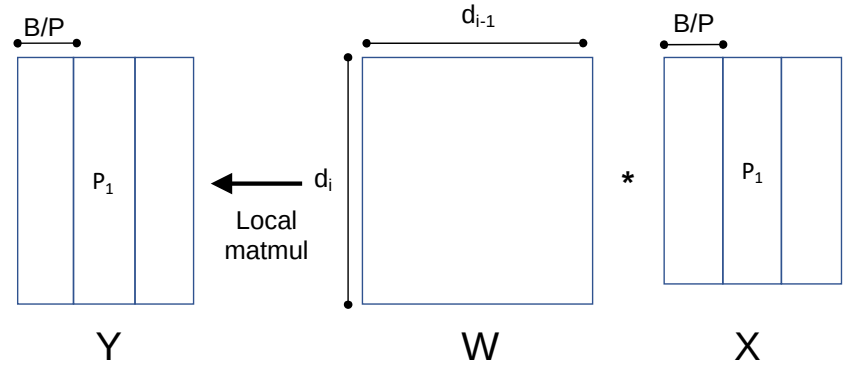


N = the number of outputs
M = the number of inputs
B = the size of the minibatch

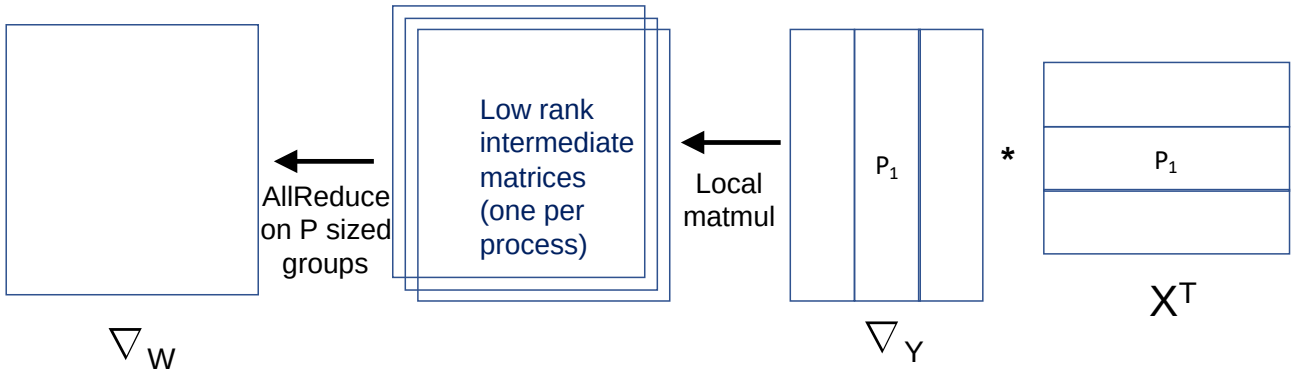
The impact to parallelism:

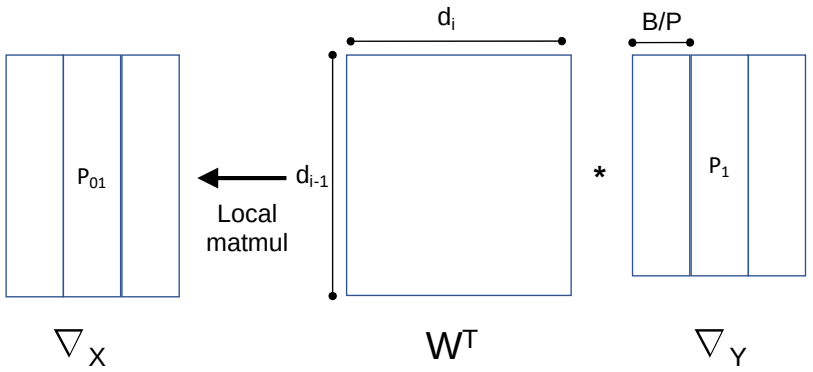
- **W** is replicated to processor, so it doesn't change
- $\mathbf{X_{in}}$ and $\mathbf{X_{out}}$ gets skinnier if we only use data parallelism, i.e. distributing **b=B/p** mini-batches per processor
- GEMM performance suffers as *matrix dimensions get smaller* and *more skewed (BLAS3 vs. 2)*
- **Result:** Data parallelism can hurt single-node performance

Data Parallel SGD training of NNs as matrix operations



1. Which matrices are replicated?
2. Where is the communication?
3. Which steps can be overlapped?



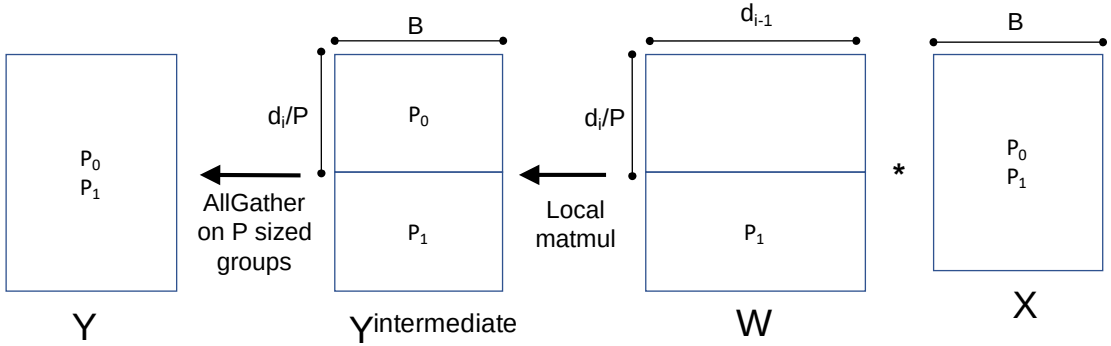


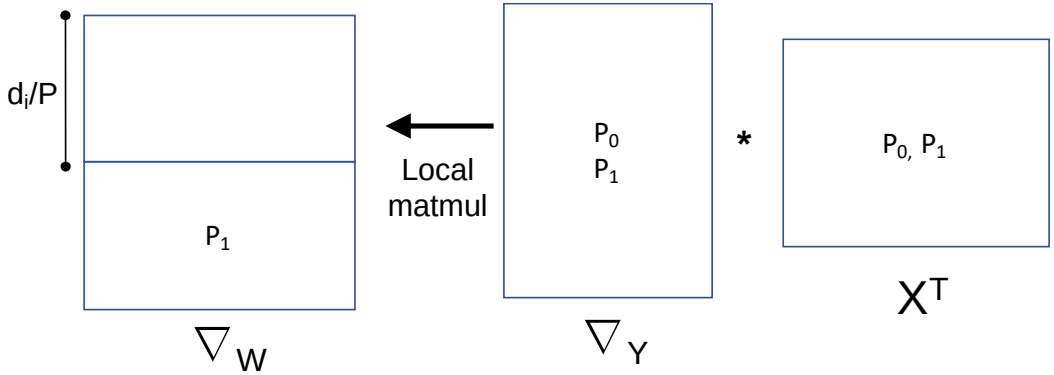
$\nabla_Y = \partial L / \partial Y$ = how did the loss function change as output activations changed?

$\nabla_X = \partial L / \partial X$

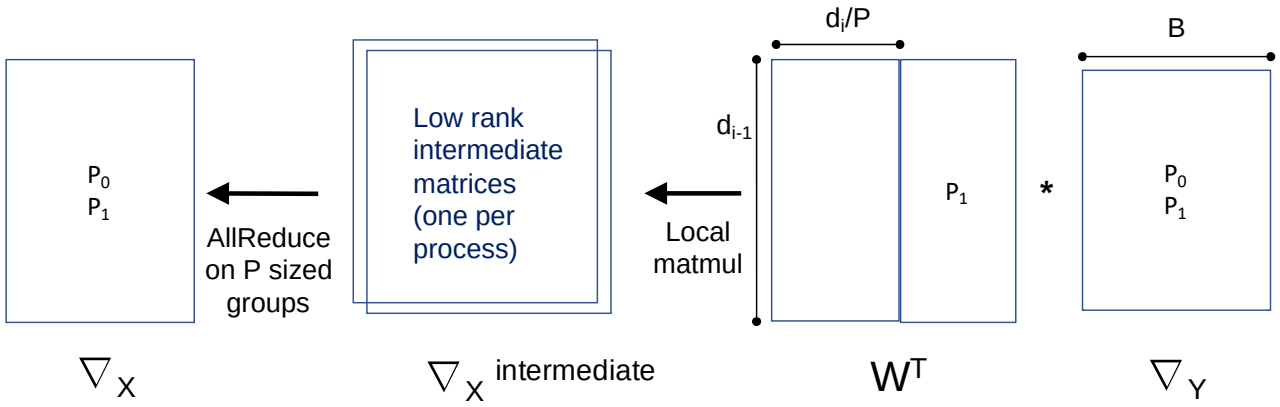
$\nabla_W = \partial L / \partial W$

Model Parallel SGD training of NNs as matrix operations

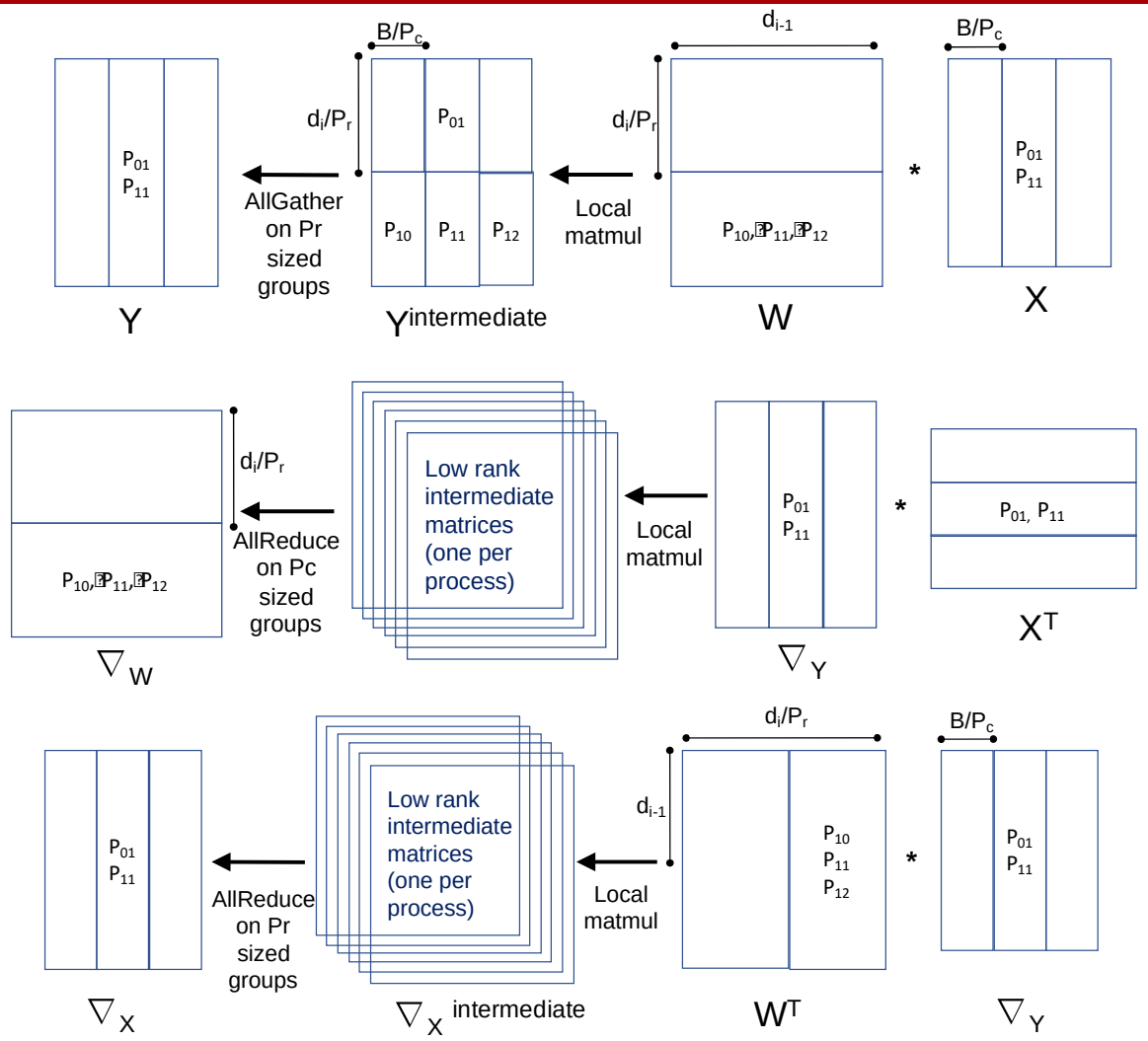




1. Which matrices are replicated?
2. Where is the communication?
3. How can matrix algebra capture both model and data parallelism?



Data & Model Parallel SGD training of NNs as matrix operations



Processes are 2D indexed:
P= P_r x P_c

Amir Gholami, Ariful Azad, Peter Jin, Kurt Keutzer, Aydın Buluç. "Integrated Model, Batch and Domain Parallelism in Training Neural Networks." https://arxiv.org/abs/1712.04432

Outline of the lecture

Intro and Supervised Learning

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

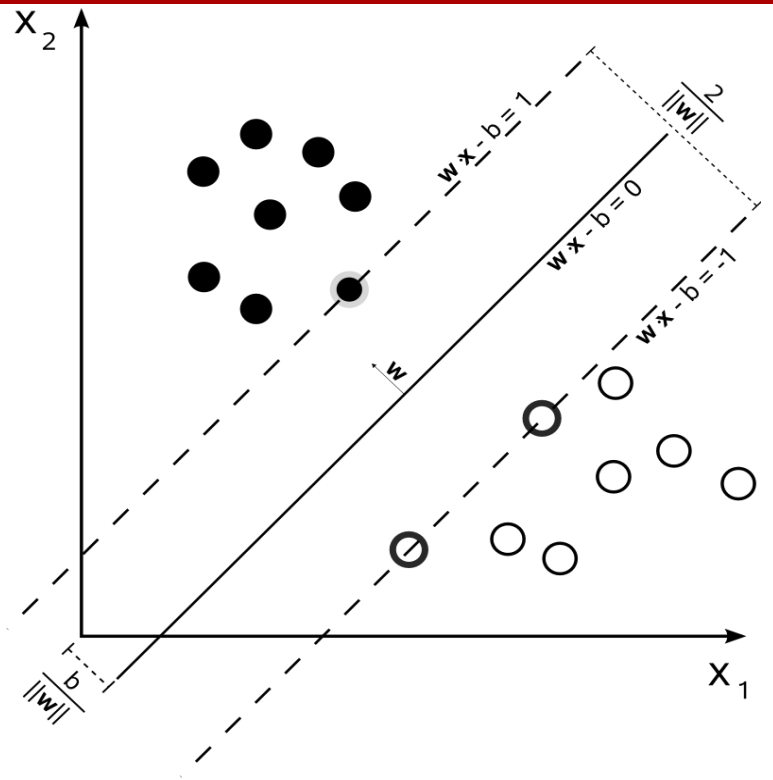
Unsupervised Learning

- Non-Negative Matrix Factorization
- Spectral and Markov Clustering
- Sparse Inverse Covariance Matrix Estimation

Support Vector Machine

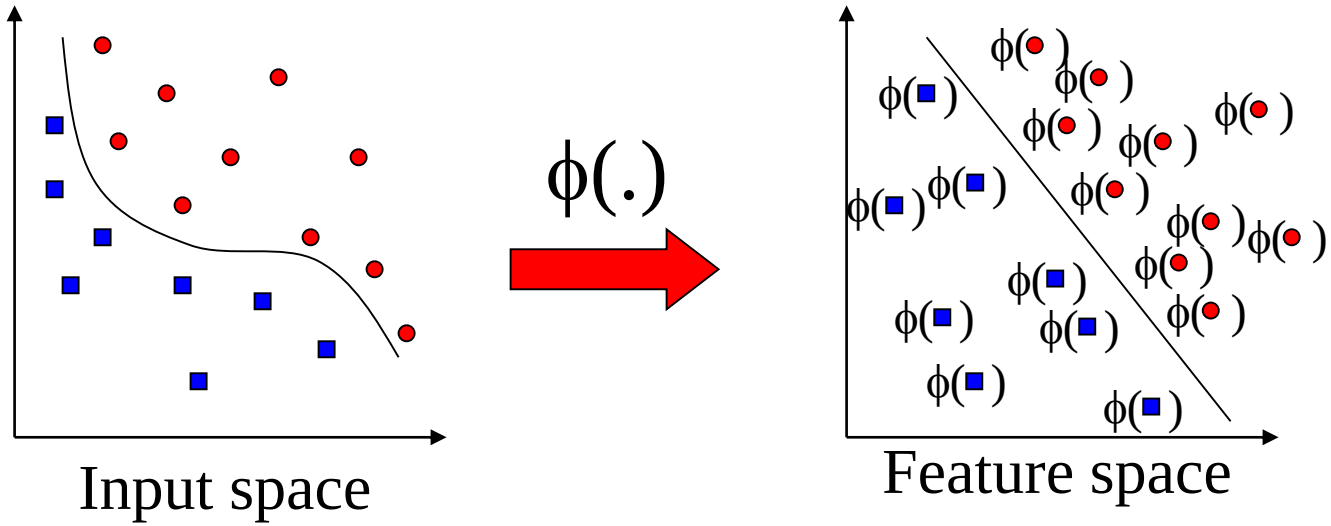
$$\max_{w,b} \quad \frac{1}{\|w\|}$$

$$\text{s.t} \quad y_i(w^T x_i + b) \geq 1, \quad \forall i$$



- Only the classification constraints on the support vectors are active
- Naively, leads to a giant quadratic constrained optimization (QP) problem
- Special algorithms, such as Sequential Minimal Optimization (SMO), decompose this giant QP to smaller (in fact ***minimal***) QP sub-problems.

Kernel Support Vector Machine



Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!

Kernel Support Vector Machine

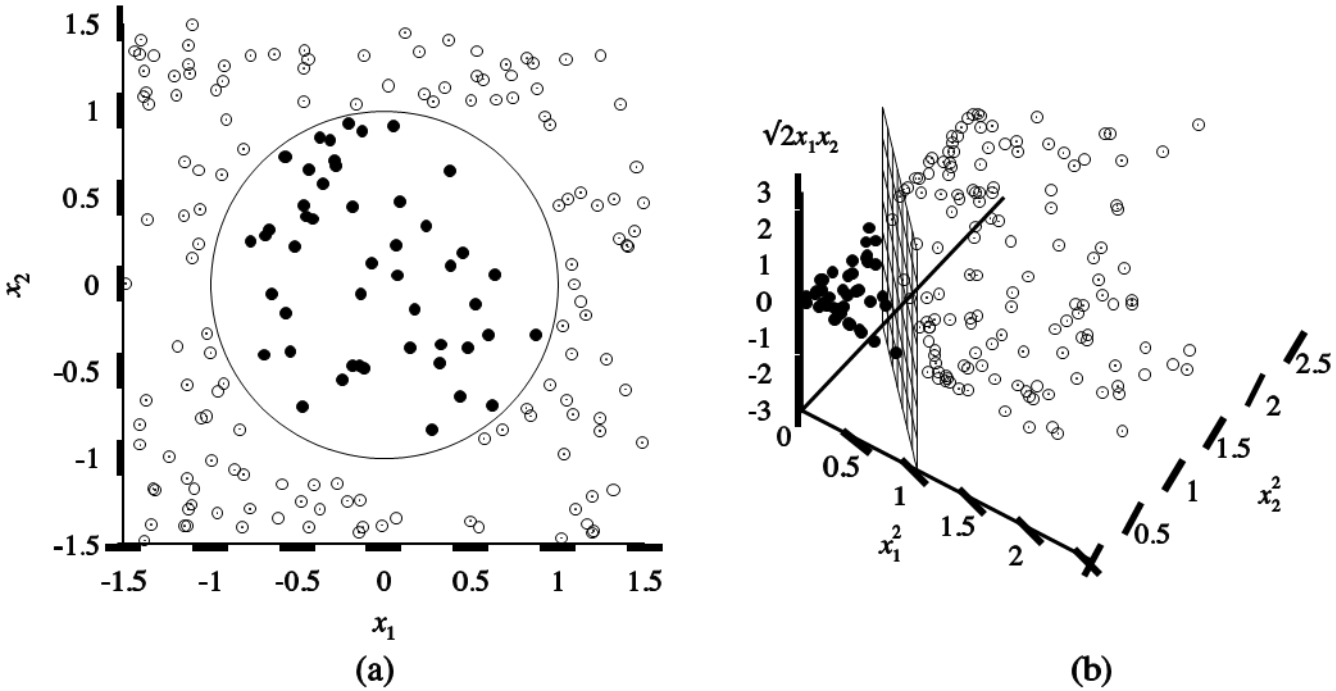


Figure source:
Russell & Norvig

The circular decision boundary in 2D (a) becomes a linear boundary in 3D (b) using the following transformation: $f(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$

If we define the kernel function as follows,

$$K(x, y) = (x^T y)^2$$

there is no need to carry out $\phi(.)$ explicitly because (kernel trick!)

$$\phi(x) \cdot \phi(y) = (x^T y)^2 = K(x, y)$$

Major Bottleneck of Kernel SVM

- Input dataset: n-by-d matrix (n >> d)
  - X1, X2, ..., Xn. Xi is a vector with d features
- Generate a n-by-n Kernel matrix at runtime
  - K[i][j] = exp(-r||Xi - Xj||^2), r is positive number
- O(n^3) operations and O(n^2) memory are huge!
  - a small input generates a large Kernel matrix
  - 357MB input (52K-by-90) = 2000GB Kernel matrix
- Solution: SMO (sequential minimal optimization)
  - using iterative method, avoiding Kernel matrix
  - key computation for sparse inputs: *sparse matrix times sparse vector*

Sequential Minimal Optimization

$$\max_{\alpha} \; W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \boxed{\kappa(\mathbf{x}_i^T \mathbf{x}_j)}$$

The kernel function

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, \ldots, m$$

$$\boxed{\sum_{i=1}^{m} \alpha_i y_i = 0.}$$

The equality constraint

- The smallest possible optimization problem involves "two" Lagrange multipliers at a time
- Because just changing on multiplier would violate the equality constraint

Platt, John. "Fast training of support vector machines using sequential minimal optimization." Advances in kernel methods. 1999.

Sequential Minimal Optimization

$$\max_{\alpha} \; W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \boxed{\kappa(\mathbf{x}_i^T \mathbf{x}_j)}$$

The kernel function

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, \ldots, m$$

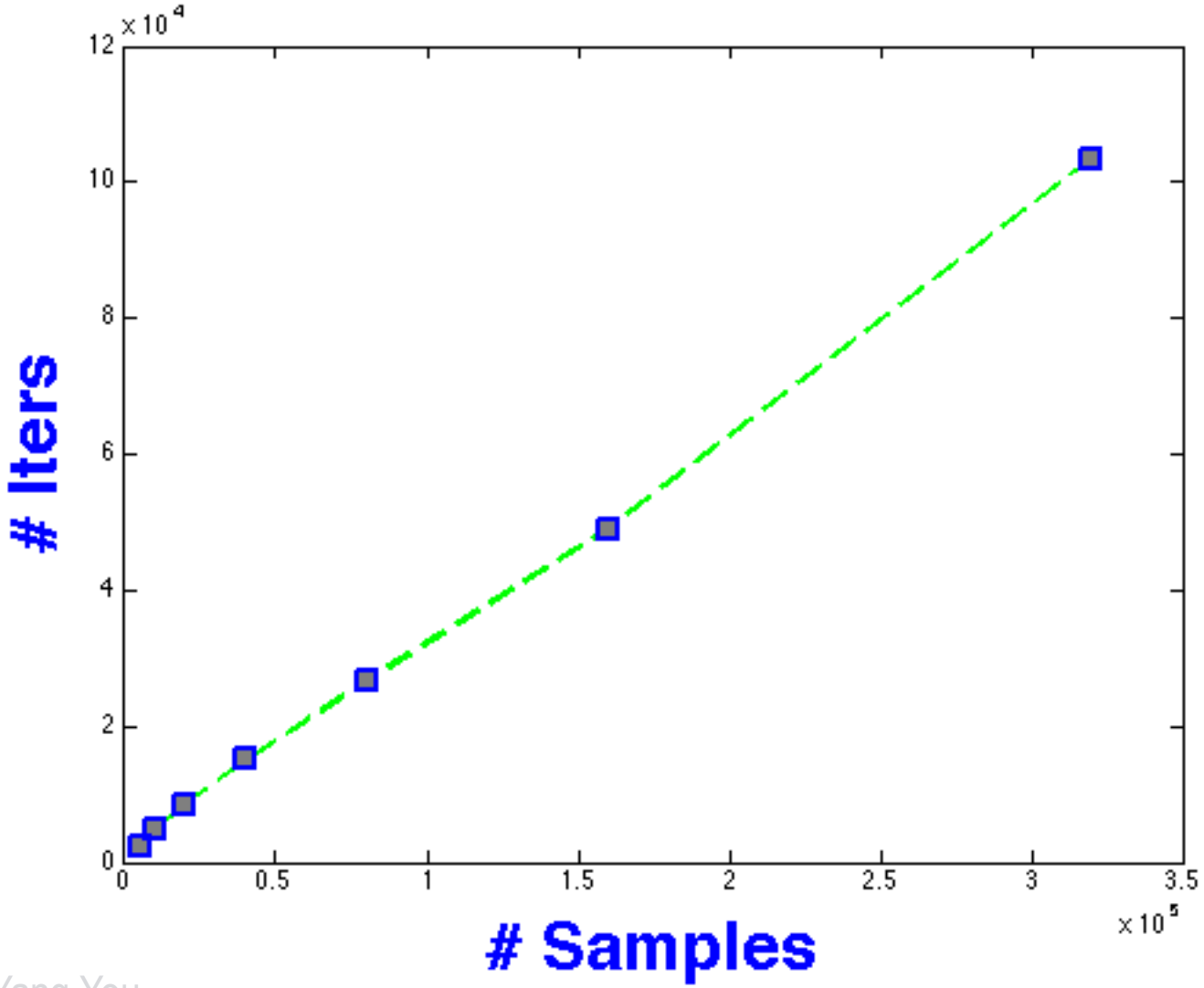
$$\boxed{\sum_{i=1}^{m} \alpha_i y_i = 0.}$$

The equality constraint

Repeat until convergence:

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Re-optimize W(α) with respect to α_i and α_j, while holding all the other α_k's fixed.

#Iterations = O(#samples), bad Weak Scaling!



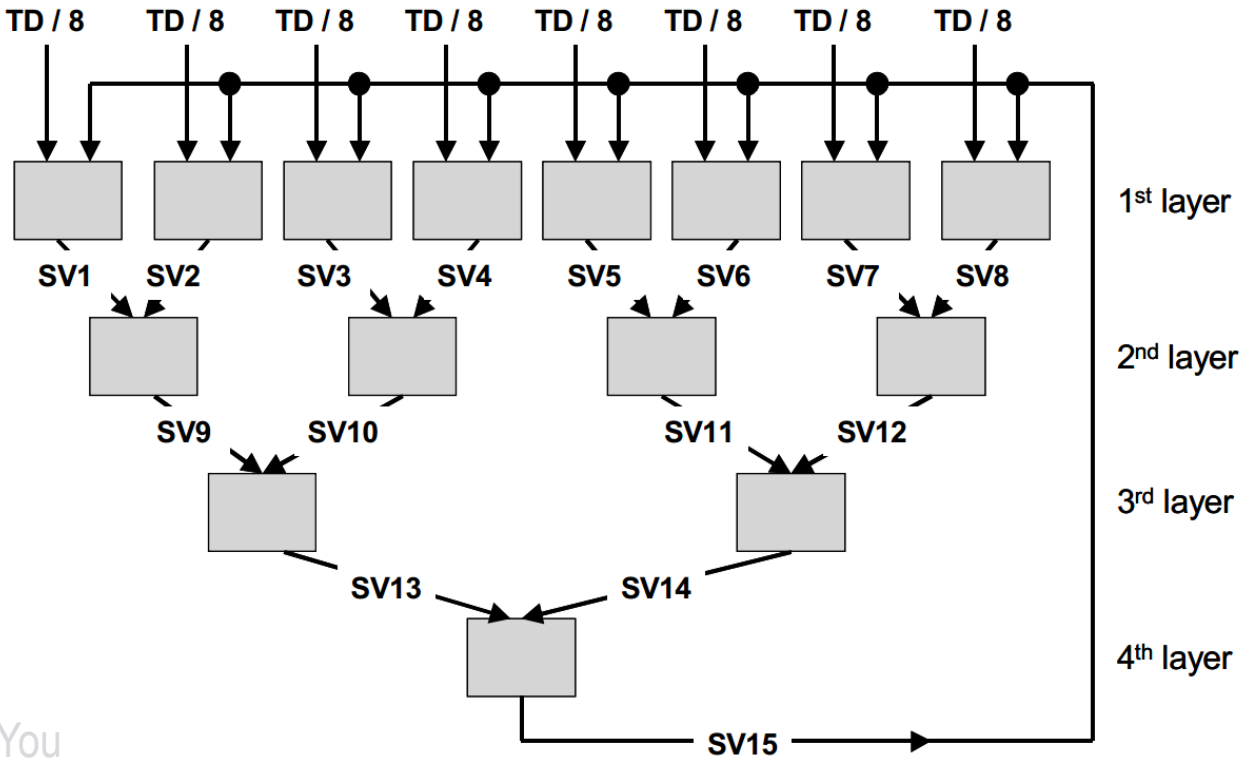
Slide: Yang You

Cascade SVM (NIPS'04)

Data is partitioned and processed by multi SVMs

Remove the non support vectors layer-by-layer

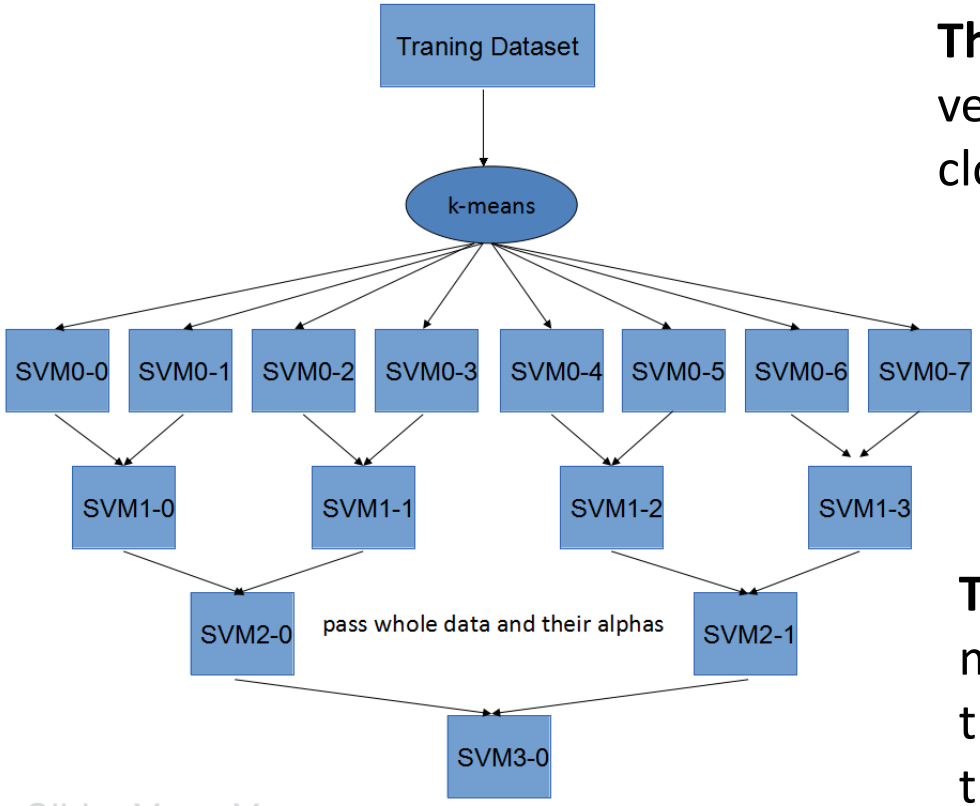
- data is the support vectors (SV) of previous layer
- pass parameters α_i of SVs to next layer for a better initiation (warm start)



Slide: Yang You

Divide-and-Conquer SVM: DC-SVM (ICML'14)

- Difference between DC-SVM and Cascade
  - DC-SVM passes all data layer-by-layer
  - DC-SVM uses kernel k-means to partition the dataset



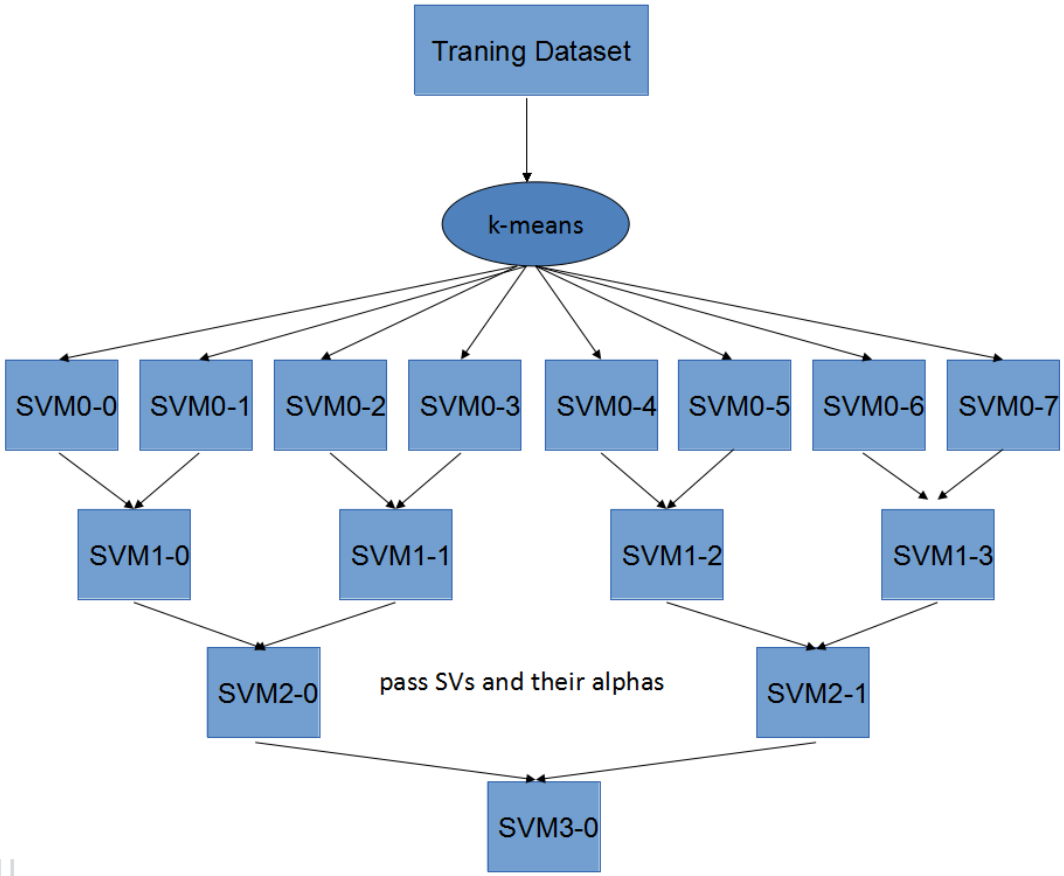
Theorem #1: the set of support vectors from the subproblems is close to that of the whole problem

Theorem #2: Kernel kmeans minimizes the difference between the solution of subproblems and of the whole problem

Slide: Yang You

Combine DC-SVM with Cascade: DC-Filter

- Only pass support vectors layer-by-layer (reduce workload)
- Use kernel k-means to divide the dataset



Slide: Yang You

Bottleneck of Cascade, DC-SVM, and DC-Filter

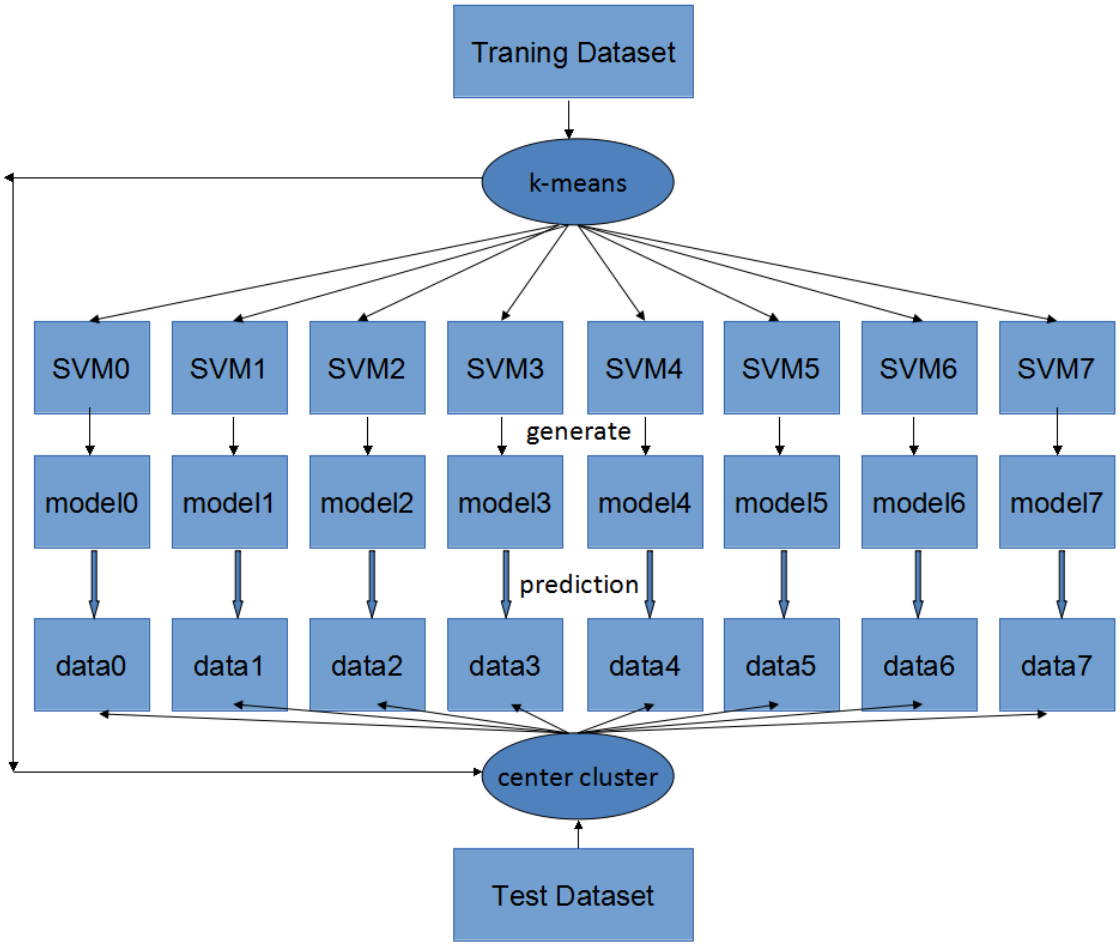
- Occupy P machines, but bottom level uses 1 machine
- Lower levels cost more time than top level

level 1^{st}	6000	6000	6000	6000	6000	6000	6000	6000
time: 5.49s	4.87	4.92	4.90	4.68	5.12	5.10	5.49	4.71
iter: 6168	5648	5712	5666	5415	5936	5904	6168	5453
SVs: 5532	746	715	717	718	686	707	721	699
level 2^{nd}	1461		1435		1393		1420	
time: 1.58s	1.58		1.50		1.35		1.45	
iter: 7485	7485		7211		6713		7035	
SVs: 5050	1292		1263		1256		1239	
level 3^{rd}	2555				2495			
time: 3.34s	3.34				3.30			
iter: 9081	8975				9081			
SVs: 4699	2388				2311			
level 4^{th}	4699							
time: 9.69s	9.69							
iter: 14052	14052							
SVs: 4475	4475							

Slide: Yang You

CP-SVM: Cluster Partition SVM (1-level divide-and-conquer)

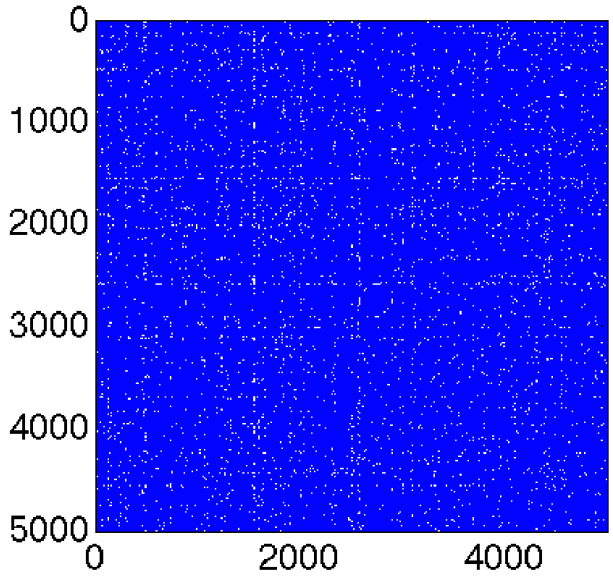
- Divide: K-means partitions data into P parts; P models
- Conquer: Euclidean distance to select best model

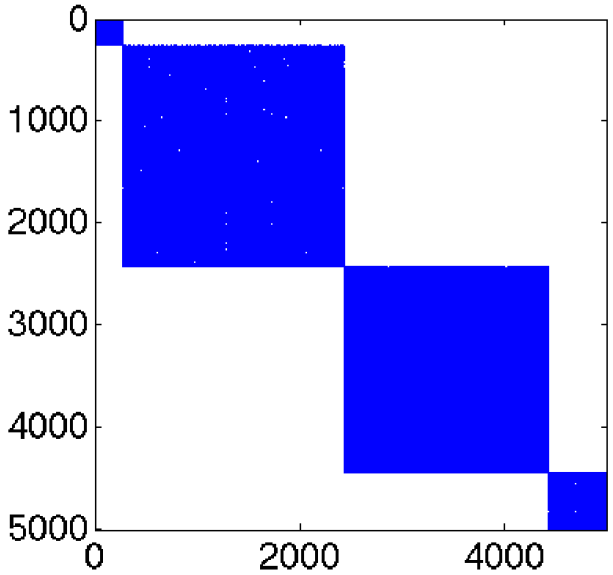


Slide: Yang You

Why CP-SVM works?

- When ||Xi - Xj||^2 is large, exp(-r||Xi - Xj||^2) is zero
- K-means maximize different groups' Euclidean distance
- These two matrices have similar F-norm
- Analysis assumes the Gaussian kernel: For a given sample, only the support vectors close to it can have an effect on the classification





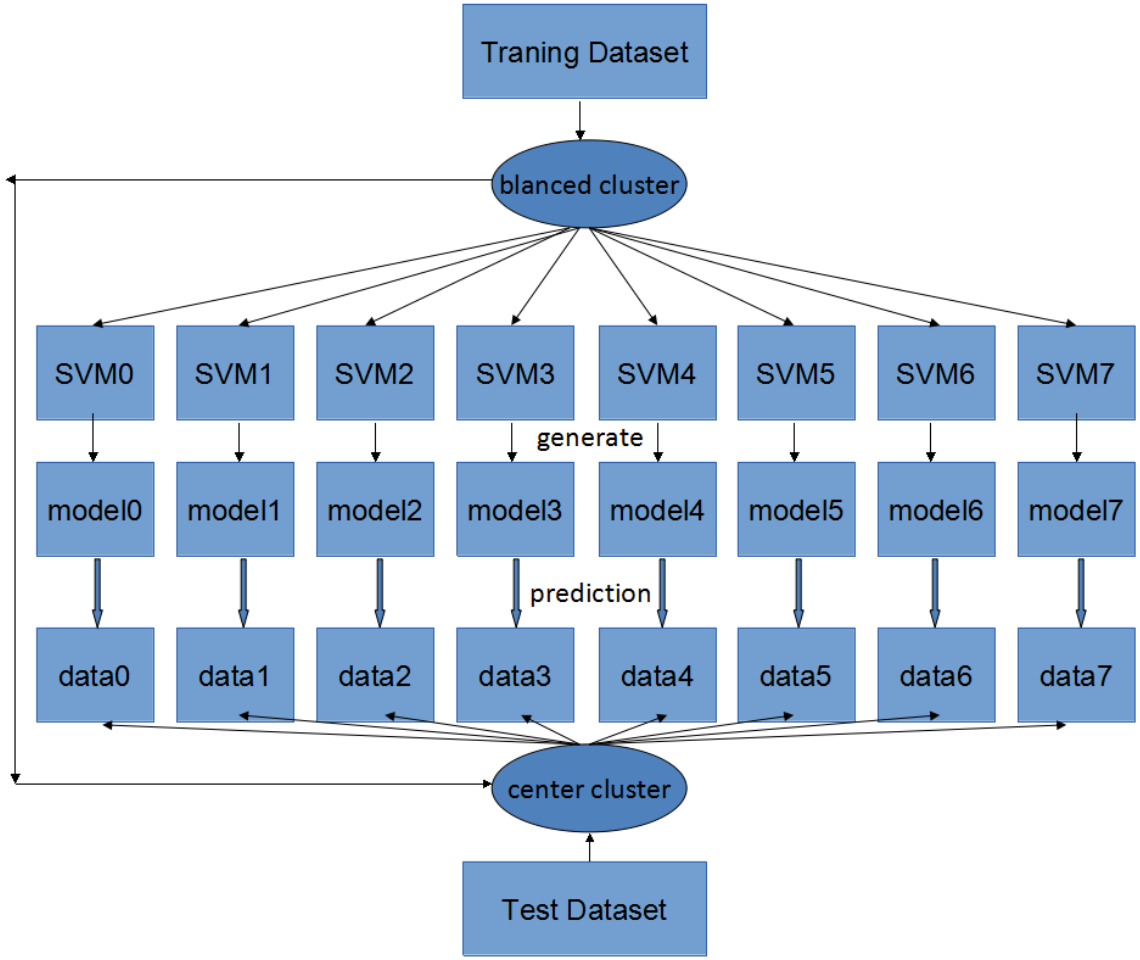
Slide: Yang You

CP-SVM's bottleneck: load imbalance (because of K-means)

- Machine 5, time: 0.75s, #iter: 1522, #samples: 4800
- Machine 1, time: 0.79s, #iter: 1384, #samples: 4800
- Machine 7, time: 0.94s, #iter: 1748, #samples: 4800
- Machine 6, time: 1.14s, #iter: 2337, #samples: 4800
- Machine 2, time: 1.14s, #iter: 2339, #samples: 4800
- Machine 4, time: 1.31s, #iter: 2856, #samples: 4800
- Machine 3, time: 5.48s, #iter: 6723, #samples: 9600
- Machine 3, time: 6.48s, #iter: 7915, #samples: 9600

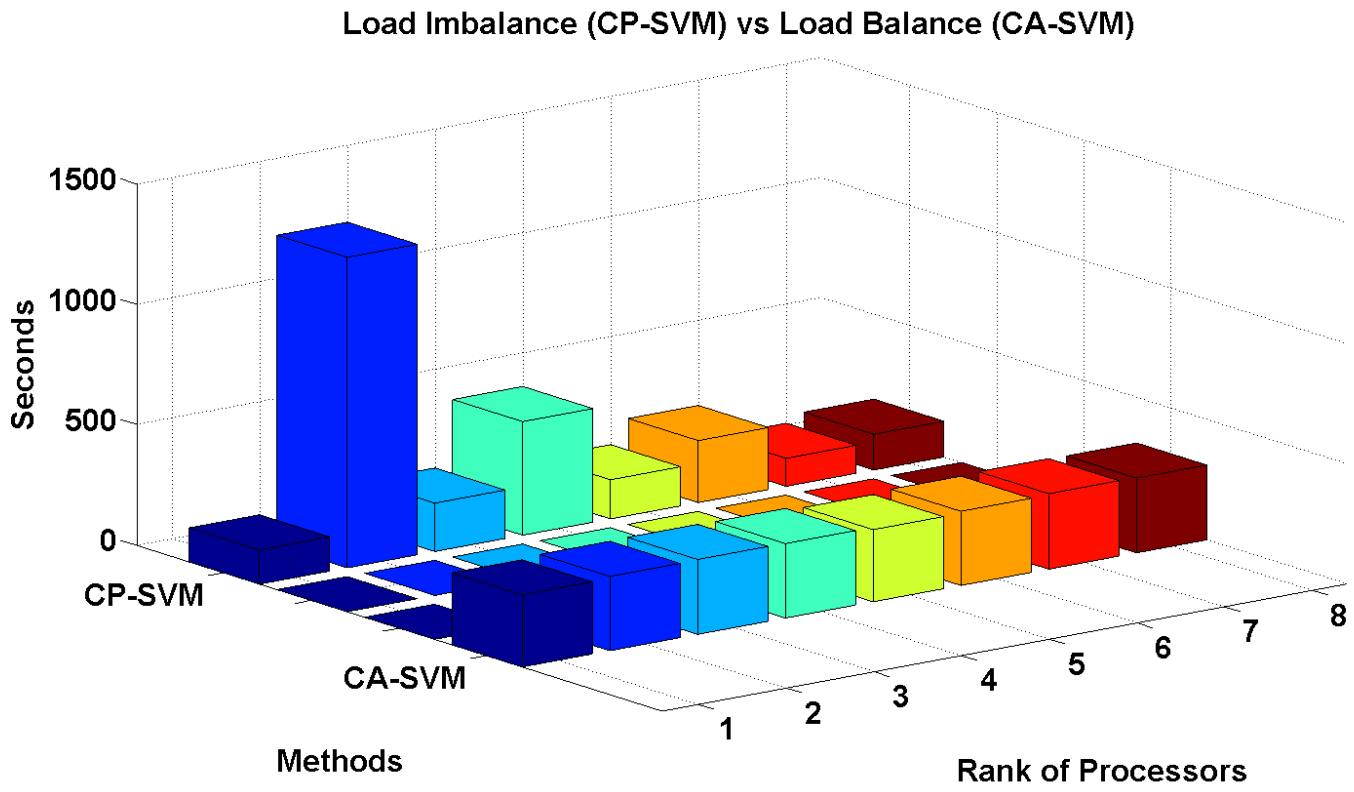
CA-SVM: Communication Avoiding SVM

- Design a balanced clustering to replace K-means
- Still approximating the distance separation property of k-means as much as possible.



Load Balance Comparison

- Test dataset is epsilon with 128k samples on 8 machines



Slide: Yang You

0.2% accuracy loss for 6.6x speedup

- Overall comparison for "IJCNN dataset"
- All methods use the same number of machines

Method	Accuracy	Iterations	Time (Init, Training)
Dis-SMO	98.7%	30,297	23.8s (0.008, 23.8)
Cascade	95.5%	37,789	13.5s (0.007, 13.5)
DC-SVM	98.3%	31,238	59.8s (0.04, 59.7)
DC-Filter	95.8%	17,339	8.4s (0.04, 8.3)
CP-SVM	98.7%	7,915	6.5s (0.04, 6.4)
CA-SVM	98.5%	7,450	3.6s (0.005, 3.55)

You, Yang, et al. "CA-SVM: Communication-Avoiding Support Vector Machines on Distributed Systems", IPDPS, 2015

Outline of the lecture

Intro and Supervised Learning

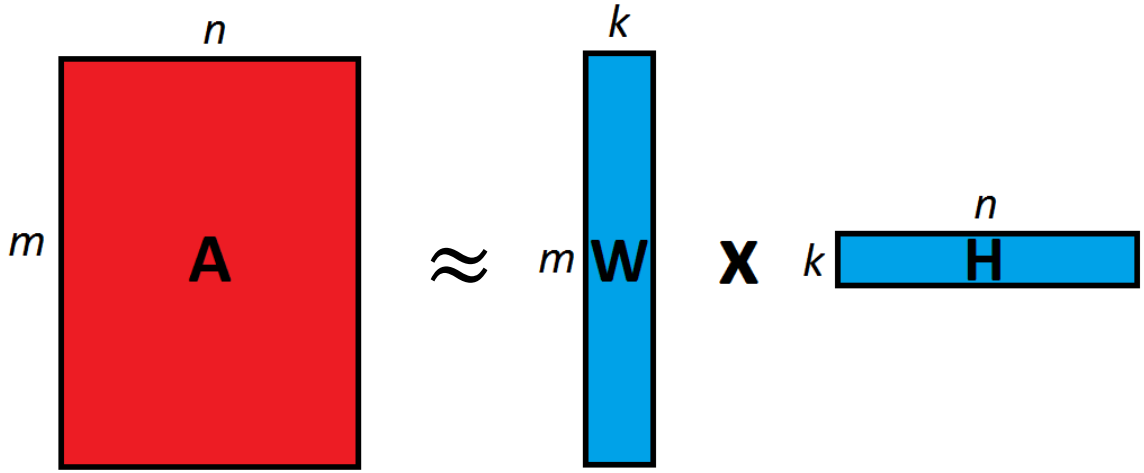
- Machine Learning & Parallelism Intro
- Neural Network Basics
- Support Vector Machines

Unsupervised Learning

- Non-Negative Matrix Factorization
- Spectral and Markov Clustering
- Sparse Inverse Covariance Matrix Estimation

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

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. HH^T and AH^T for updating W, given a fixed H
2. W^TW and W^TA 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

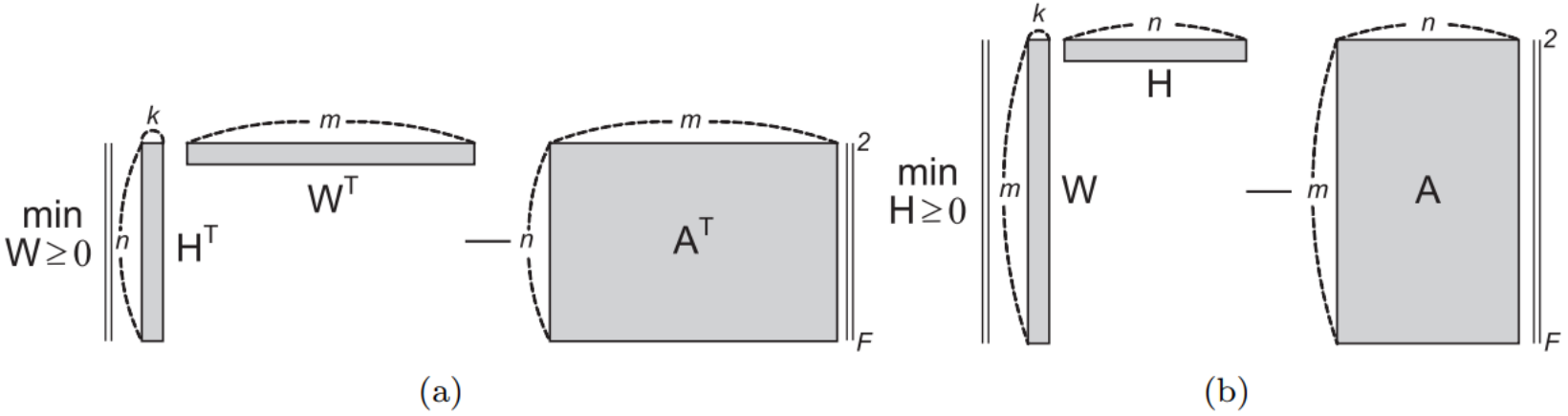


Figure: Kim and Park

The Alternating Updates Framework

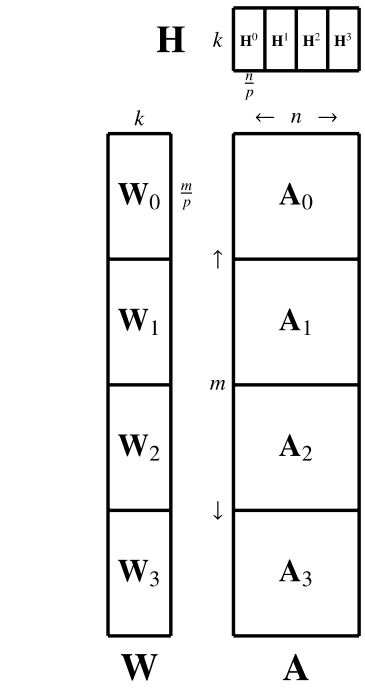
Main computation is large-scale matrix multiplications

Choose the best distribution and algorithm depending on:

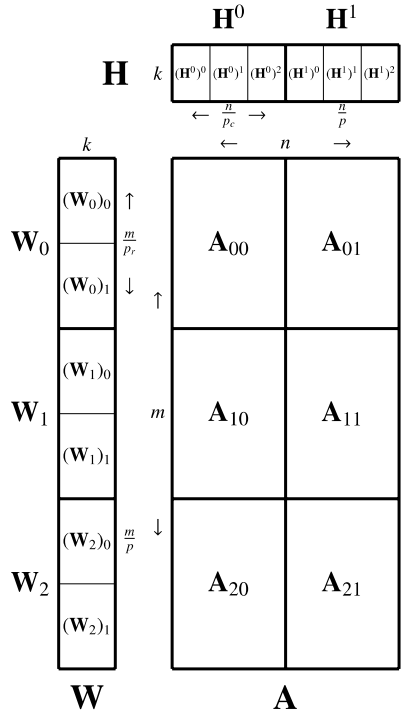
1- the relative sizes of the dimensions of the matrices

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 lecture

Intro and Supervised Learning

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

Unsupervised Learning

- Non-Negative Matrix Factorization
- Spectral and Markov Clustering
- Sparse Inverse Covariance Matrix Estimation

Clustering

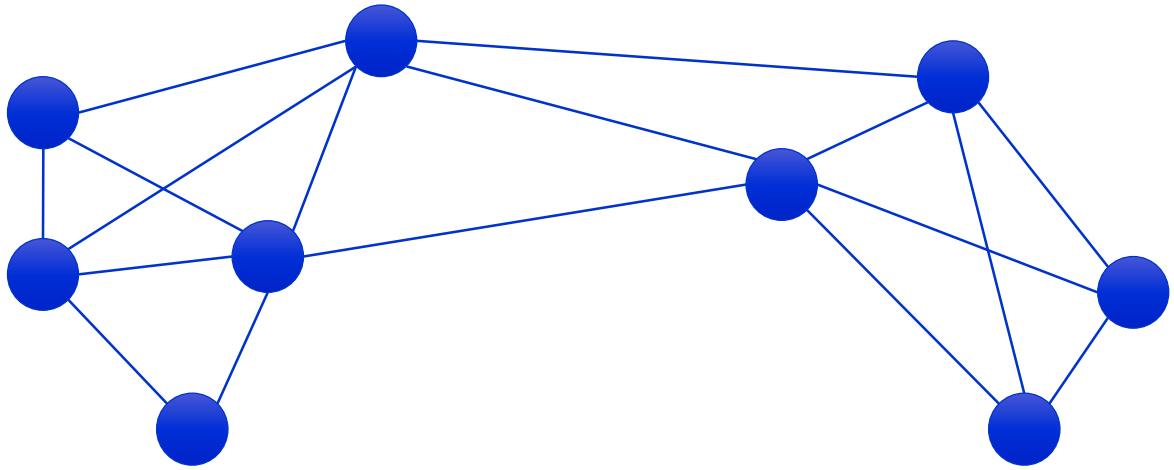
Many families of methods

- 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, ...
- 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

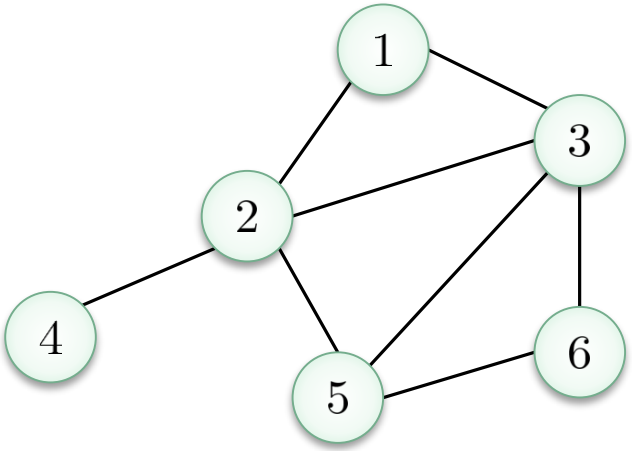
- Graph Laplacian
  - unnormalized graph Laplacian : $L = D - W$
  - normalized graph Laplacian

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

$$L_{rw} = D^{-1} L = I - D^{-1} W$$

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

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

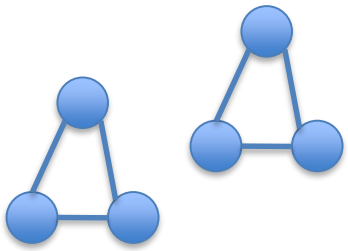
1. Construct a similarity graph and compute the normalized graph Laplacian L_{sym}.
2. Compute the k smallest eigenvectors $u_1, u_2, \cdots, u_k$ of L_{sym}.
3. Let $U = [\, u_1\ u_2\ \cdots\ u_k\,] \in \mathbb{R}^{n \times k}$.
4. Normalized the rows of U to norm 1.

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

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.

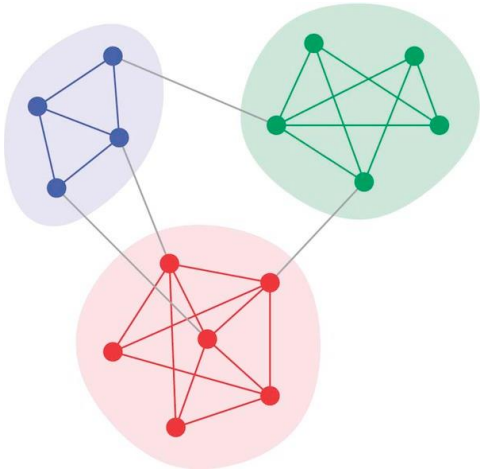
How to compute those smallest Eigenvectors?

- 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 (recall last 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)



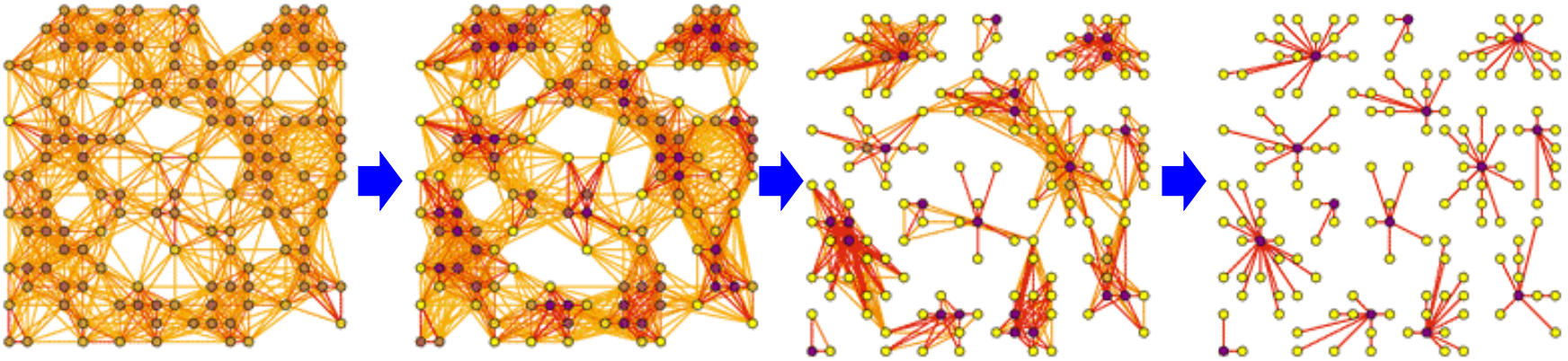
The number of **edges or higher-length paths** between two arbitrary nodes in a cluster is greater than the number of paths between nodes from different clusters

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)



Initial network | Iteration 1 | Iteration 2 | Iteration 3

At each iteration:

Step 1 (Expansion): Squaring the matrix

[corresponds to computing random walks of higher length]

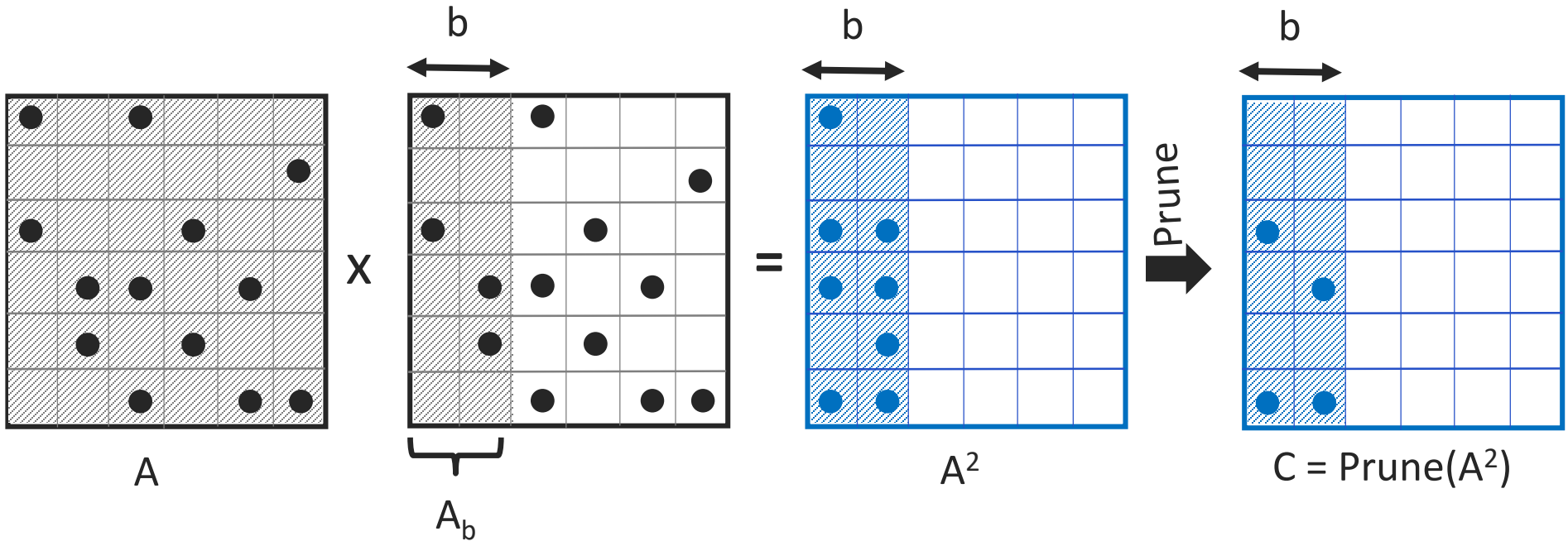
Step 2 (Inflation) : Hadamard power of a matrix (taking powers entrywise)

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

The expansion step of the MCL algorithm

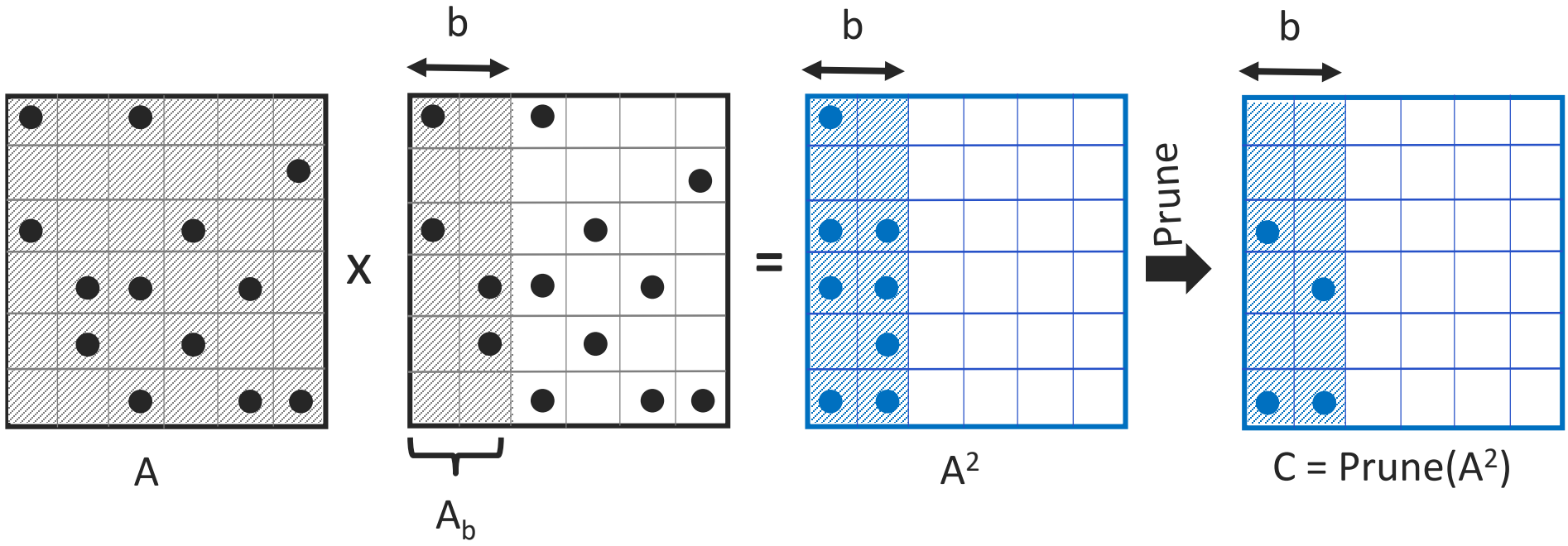
- Goal: Compute random walks of higher length
- Input: A column stochastic matrix (A)
- Algorithm
  1. Sparse matrix-sparse matrix multiplication (**SpGEMM**): A^2
  2. **Sparsify** A^2 by removing low probability terms
     - **Prune** entries in A^2 that are smaller than a threshold
     - **Recover** (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

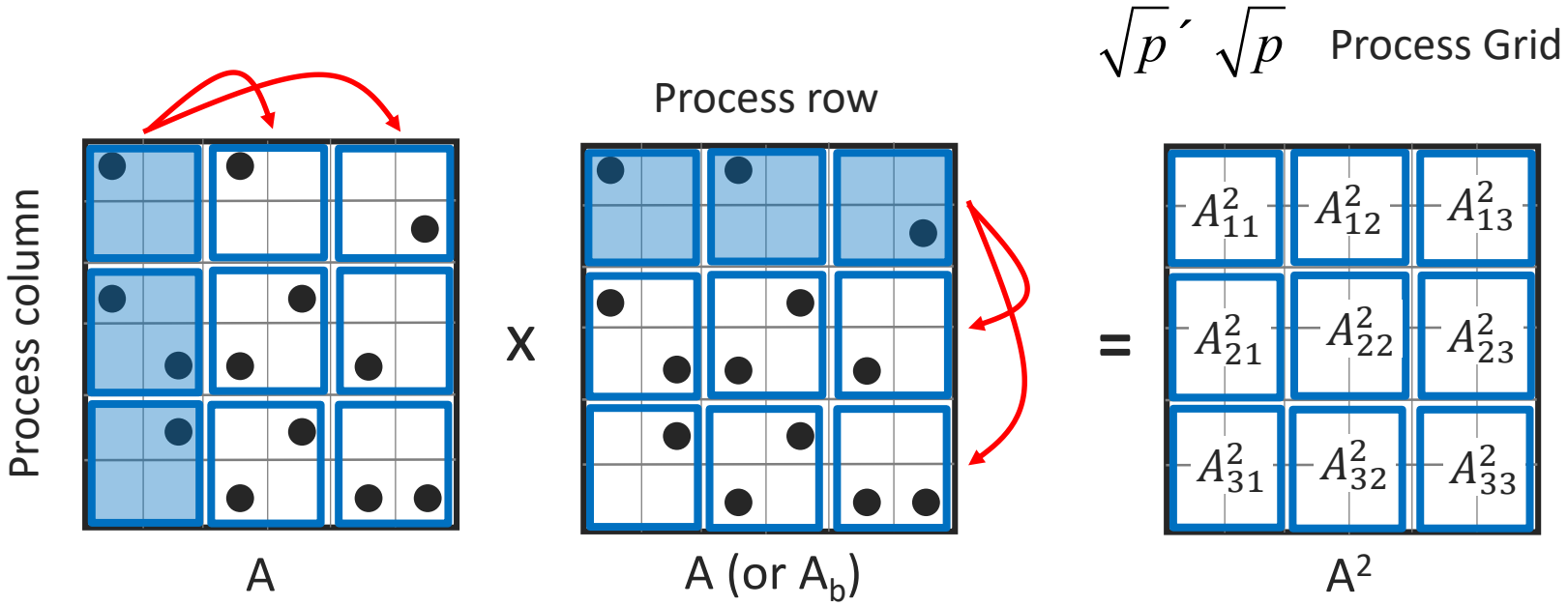
A combined expansion and pruning step



- 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.



Other algorithmic steps of HipMCL

- There is more than sparse matrix multiply here.
  - Parallel k-selection algorithm for each column of the matrix
  - Parallel pruning algorithm
  - Parallel connected component algorithm (to identify clusters after MCL is converged).
  - Parallel file I/O.

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.

Outline of the lecture

Intro and Supervised Learning

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

Unsupervised Learning

- Non-Negative Matrix Factorization
- Spectral and Markov Clustering
- Sparse Inverse Covariance Matrix Estimation

Sparse Inverse Covariance Matrix Estimation

- Precision matrix = Inverse covariance matrix
- Goal: Estimating graphical model structure
- "The zeros of a precision matrix correspond to zero partial correlation, a necessary and sufficient condition for conditional independence (Lauritzen, 1996)"
- Sparsity often enforced by regularization
- One algorithm (HP-CONCORD)'s objective function:

$$\underset{\Omega \in \mathbf{R}^{p \times p}}{\text{minimize}} \; -\log\det(\Omega_D^2) + \mathbf{tr}(\Omega S \Omega) + \lambda_1 \|\Omega_X\|_1 + \frac{\lambda_2}{2}\|\Omega\|_F^2,$$

- Ω is the sparse inverse covariance matrix we are trying to estimate

Why do we care? Finding Direct Associations

Partial Correlation (a.k.a. sparse inverse covariance estimation): direct association without confounders

- Gene Regulatory Network (GRN) estimation
- Joint modeling of SNPs and GRN
- Linkage Disequilibrium (LD) estimation
- Canonical Correlation Analysis (CCA)
- Genome-wide association studies (GWAS)

Data-driven hypothesis generation!

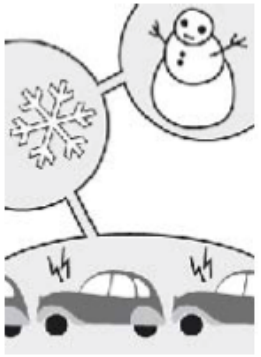


Fig. 1. Conditionally on the height of snow, the number of snowmen is independent of the intensity of traffic jams. This is represented by a two edges graph.

- Computationally challenging;

HP-CONCORD Algorithm

Algorithm 2 The Cov variant of HP-CONCORD, for computing a sparse estimate of the inverse covariance matrix.

Input: data matrix $X \in \mathbf{R}^{n \times p}$; tuning parameters $\lambda_1, \lambda_2 \geq 0$; optimization tolerance $\epsilon > 0$
Output: estimate $\hat{\Omega} \in \mathbf{R}^{p \times p}$ of the underlying inverse covariance matrix Ω^0

$\Omega^{(0)} \leftarrow I$
Compute $S \leftarrow \frac{1}{n} X^T X$ ▷ Compute (once) via a distributed dense-dense matrix multiplication
Compute $W^{(0)} \leftarrow \Omega^{(0)} S$ ▷ Compute via a distributed sparse-dense matrix multiplication
for $k = 0, 1, 2, \ldots$
  Form $(W^{(k)})^T$ ▷ Form via a distributed matrix transpose
  $G^{(k)} \leftarrow -(\Omega_D^{(k)})^{-1} + \frac{1}{2}((W^{(k)})^T + W^{(k)}) + \lambda_2 \Omega^{(k)}$ ▷ Use $W^{(k)}, (W^{(k)})^T$
  $g(\Omega^{(k)}) \leftarrow -2\sum_i \log(\Omega_{ii}^{(k)}) + \mathbf{tr}(W^{(k)}\Omega^{(k)}) + \frac{\lambda_2}{2}\|\Omega^{(k)}\|_F^2$ ▷ Use $(W^{(k)})^T$; see text for details
  for $\tau = 1, \frac{1}{2}, \frac{1}{4}, \ldots$
    $\Omega^{(k+1)} \leftarrow \mathcal{S}_{\tau\lambda_1}(\Omega^{(k)} - \tau G^{(k)})$ ▷ Apply the soft-thresholding operator, $\mathcal{S}_{\tau\lambda_1}$, in a distributed manner
    Compute $W^{(k+1)} \leftarrow \Omega^{(k+1)} S$ ▷ Compute via a distributed sparse-dense matrix multiplication
    $g(\Omega^{(k+1)}) \leftarrow -2\sum_i \log(\Omega_{ii}^{(k+1)}) + \mathbf{tr}(W^{(k+1)}\Omega^{(k+1)}) + \frac{\lambda_2}{2}\|\Omega^{(k+1)}\|_F^2$ ▷ See text for details
  until $g(\Omega^{(k+1)}) \leq g(\Omega^{(k)}) - \mathbf{tr}((\Omega^{(k)} - \Omega^{(k+1)})^T G^{(k)}) + \frac{1}{2\tau}\|\Omega^{(k)} - \Omega^{(k+1)}\|_F^2$ ▷ See text for details
until a stopping criterion is satisfied, using ϵ
return the estimate $\hat{\Omega} \leftarrow \Omega^{(k)}$

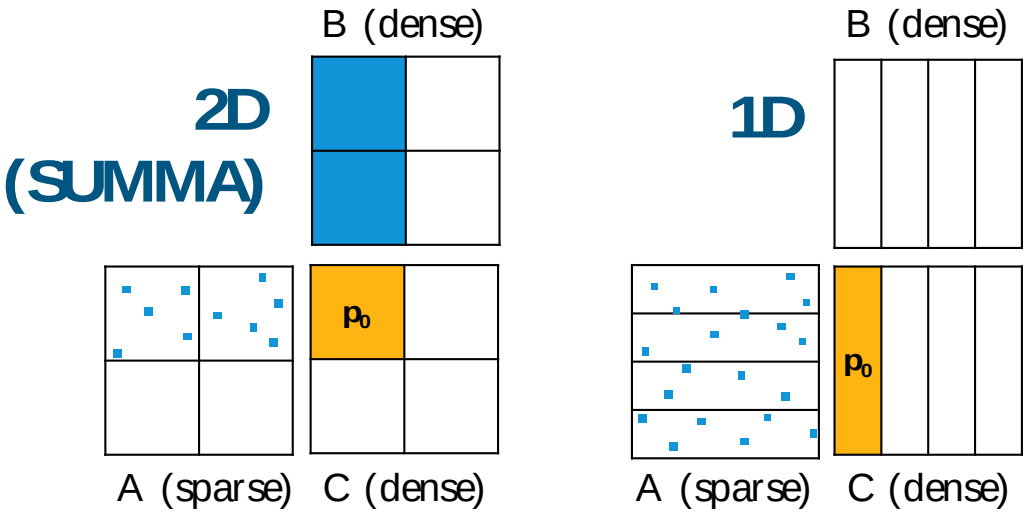
- Repeated use of sparse times dense matrix multiplication (SpDM^3)
- SpDM^3 is the bottleneck by a large margin.

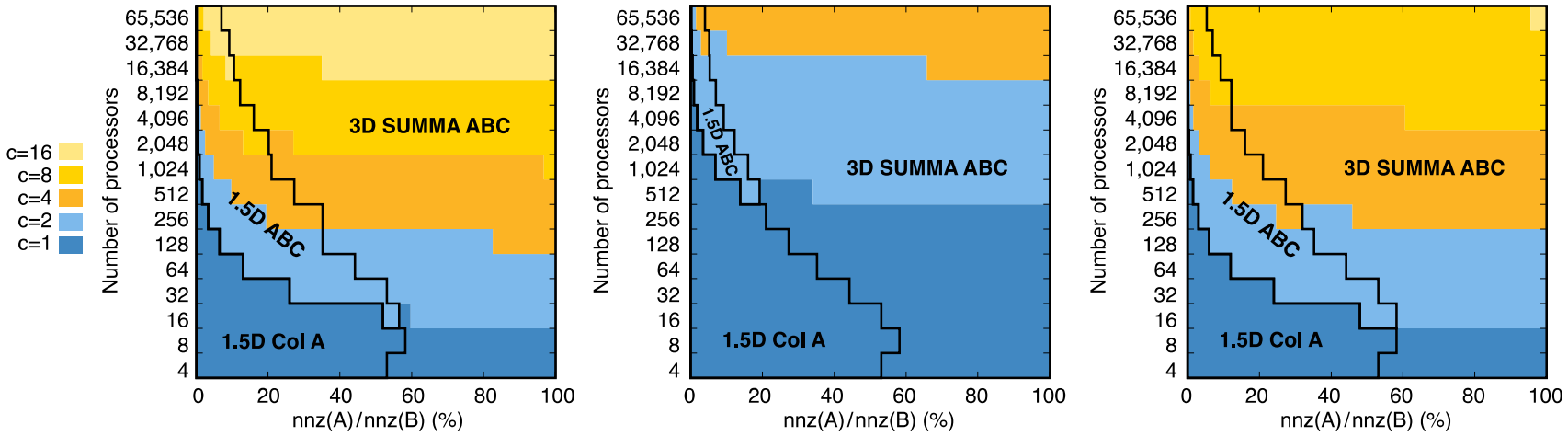
Koanantakool et al. Communication-avoiding optimization methods for distributed massive-scale sparse inverse covariance estimation. In AISTATS, 2018.

Sparse Matrix times Dense Matrix

Sometimes it pays off to communicate A instead.

How much of these ranges apply to real life NMF scenarios is open question





Koanantakool, Penporn, et al. "Communication-avoiding parallel sparse-dense matrix-matrix multiplication." IPDPS, 2016

HP-CONCORD Advantages

- HP-CONCORD makes fewer assumptions about the data (in particular, no Gaussianity is assumed) compared to competitors
- Thanks to **communication-avoiding matrix multiplication algorithms**, it reaches unprecedented scales

- BigQUIC: previous state-of-the-art
- Obs-K are the other variant of HP-CONCORD algorithm (K: number of nodes)
- Experiment is trying to recover a random graph structure.