Special Topics in Tensors

22 May 2014

max planck institut
informatik

Special Topics in Tensors

1. CP-APR: Fitting Poisson Distribution
2. Boolean Tensor Factorizations

CP-APR: Motivation

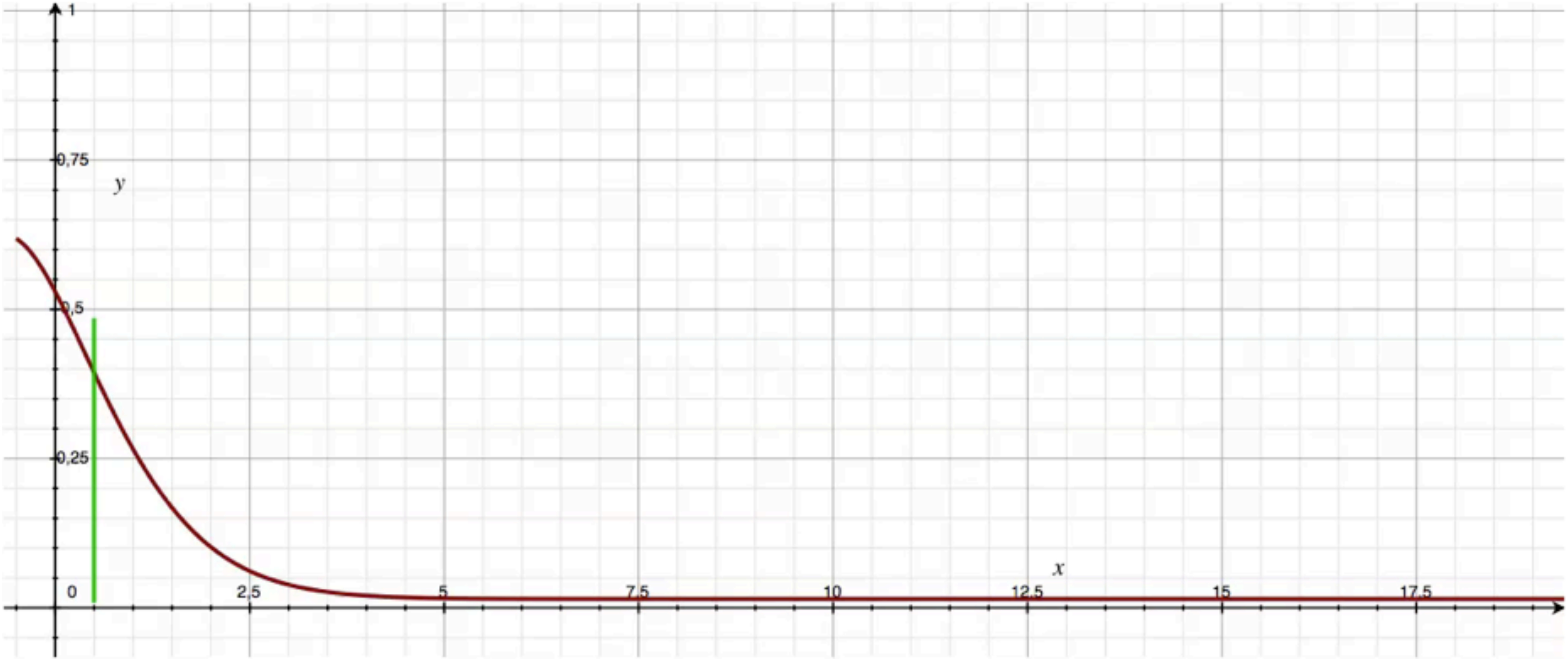
- Least-squares error has the (implicit) assumption that the noise is Gaussian
  - But this doesn't always make much sense
- Some data is counting
  - How many mails were sent from *i* to *j* using containing term *t*?
  - How many packages were sent from IP *i* to IP *j*, port *p*?
- Data like this is better explained using the **Poisson distribution**

Chi, E.C. & Kolda, T.G., 2012. On Tensors, Sparsity, and Nonnegative Factorizations. SIAM Journal on Matrix Analysis and Applications, 33(4), pp.1272–1299.

The Poisson Distribution

- The probability of number of events occurring in fixed interval if they occur on known average rate (and independently)
- One parameter $\lambda > 0$, the rate
- $f(k; \lambda) = \lambda^k e^{-\lambda}/k!$
- If $X \sim \text{Poisson}(\lambda)$, then $\text{E}[X] = \text{Var}[X] = \lambda$

The Effects of λ



$= \lambda$ $\quad = \frac{\lambda^k e^{-\lambda}}{k!}$

Modeling the Data

- We assume the values in the elements x_{ijk} of the data tensor $\mathcal{X}$ are i.i.d. Poisson distributed
- We assume the **parameters** of the distribution have low-rank non-negative CP decomposition
  - Exist non-negative $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ s.t. $x_{ijk} \sim \text{Poisson}(\sum_r a_{ir} b_{jr} c_{kr})$
- The error is measured using the **log-likelihood** of the observations
  - The log-likelihood of x_{ijk} is $x_{ijk} \cdot \ln(\lambda_{ijk}) - \lambda_{ijk} - \ln(x_{ijk}!)$
    - $\lambda_{ijk} = \sum_r a_{ir} b_{jr} c_{kr} =$ parameter
    - We **minimize** $\sum_{\text{i,j,k}} \lambda_{ijk} - x_{ijk} \cdot \log(\lambda_{ijk})$

Some Comments on Negative Log-Likelihood

- The function we minimize is the KL divergence
- We assume that $0 \cdot \log(y) = 0$ for all $y \geq 0$
- If $\lambda_{ijk} = 0$ but $x_{ijk} > 0$, then $x_{ijk} \cdot \log(\lambda_{ijk}) = -\infty$
  - Arbitrarily bad fit: we have observed something we model as impossible
  - We require this never happens: $\lambda_{ijk} > 0$ for all i, j, and k with $x_{ijk} > 0$

Interpreting CP-APR

- Data: non-negative integer tensor $\mathcal{X}$
- Model: non-negative CP decomposition $\mathcal{Y}$ s.t. $\mathcal{X}$ has high likelihood to be drawn from element-wise Poisson($\mathcal{Y}$)
  - Normalize columns of $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ s.t. they sum to 1 (values from [0,1])
    - Store the normalization values for each rank-1 tensor separately
- Interpretation: the higher the weight, the larger values the rank-1 tensor explains
  - The rank-1 tensor gives the (weighted) pattern for the weight
  - Individual factor matrices give the patterns for different modes

Solving CP-APR

- Let $\boldsymbol{\Pi} = (\boldsymbol{C} \odot \boldsymbol{B})^T$ and $\boldsymbol{1}$ be all-1s vector
- Using matricization, we can solve $\boldsymbol{A}$ from

$$\boldsymbol{A} = \arg\min_{\boldsymbol{A} \geq 0} \mathbf{1}^T \left(\boldsymbol{A}\boldsymbol{\Pi} - \boldsymbol{X}_{(1)} * \log(\boldsymbol{A}\boldsymbol{\Pi}) \right) \mathbf{1}$$

  - Similarly for $\boldsymbol{B}$ and $\boldsymbol{C}$
- We repeat this until we have converged

Solving CP-APR: The Subproblem

- Solving for $\boldsymbol{A}$ is non-trivial

$$\boldsymbol{A} = \underset{\boldsymbol{A}\geq 0}{\arg\min}\, \mathbf{1}^T\left(\boldsymbol{A\Pi} - \boldsymbol{X}_{(1)} * \log(\boldsymbol{A\Pi}\right)\mathbf{1}$$

- But we can repeatedly update A as

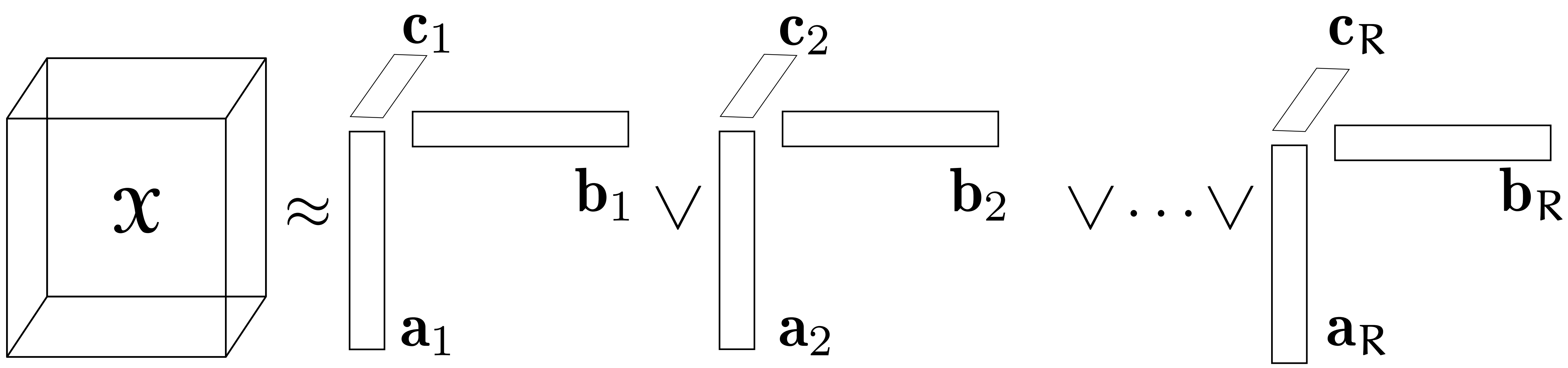
$$\boldsymbol{A} = \boldsymbol{A} * \left(\boldsymbol{X}_{(1)} \oslash (\boldsymbol{A\Pi})\right)\boldsymbol{\Pi}^T$$

  - $\oslash$ is element-wise division
  - If we update $\boldsymbol{A}$ only once, this is Lee and Seung's NMF algorithm for KL divergence

Boolean Tensor Decompositions

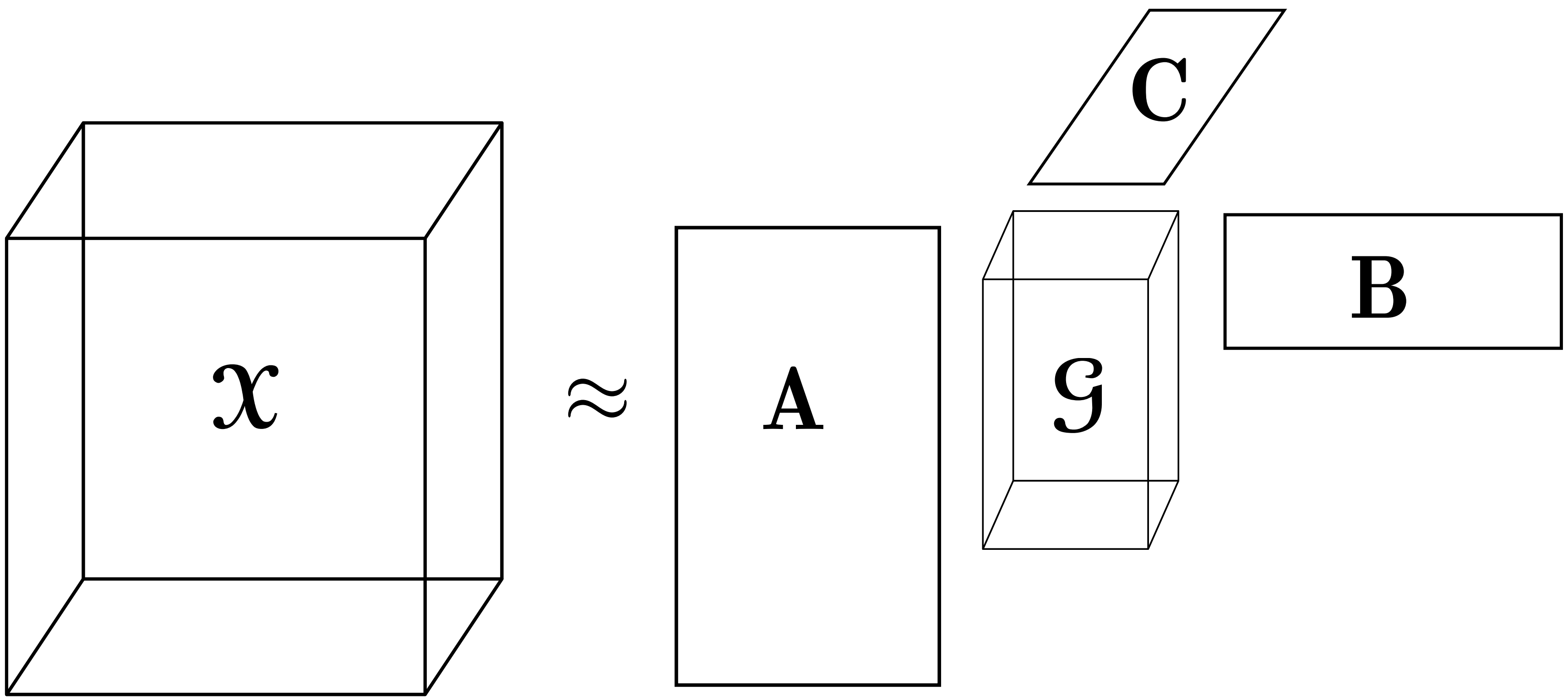
- The Poisson decomposition is still **additive**
  - The expected value of x_{ijk} is the sum of the values in the rank-1 tensors
- The **Boolean decomposition** is **idempotent**
  - The data is binary
  - The factor matrices and tensors are binary
  - The algebra is Boolean
    - 1+1 = 1, i.e. logical *or*

Boolean CP Decomposition



$$x_{ijk} = \bigvee_{r=1}^{R} a_{ir} b_{jr} c_{kr}$$

Boolean Tucker3 Decomposition



$$x_{ijk} \approx \bigvee_{p=1}^{P} \bigvee_{q=1}^{Q} \bigvee_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr}$$

Why Boolean Tensor Decompositions?

- **Interpretability**: binary in, binary out
  - Relations, sets, graphs etc. keep their interpretation
- **Non-additivity**: Finds different types of structures
  - Overlapping patterns don’t have added effect
- **Sparsity/space-efficiency**: it’s only bits
  - Sparse tensors have sparse factors

Boolean Tensor Rank

- Is the smallest R for which we have R rank-1 binary matrices whose Boolean sum is the tensor
  - Rank-1 binary tensor is the outer product of binary vectors ⇒ factor matrices in CP are binary
- Can be bigger than the smallest (or largest) dimension
  - But still no bigger than $\min\{IJ, IK, JK\}$
- There’s no Boolean border rank
- The essential uniqueness of CP doesn’t (probably) hold

Solving the Boolean CP

- The matricized equations stay almost the same
  - E.g. $\boldsymbol{X}_{(1)} = \boldsymbol{A} \boxtimes (\boldsymbol{C} \odot \boldsymbol{B})^T$
    - $\boxtimes$ is the Boolean matrix product
- But there isn’t any Boolean equivalent of the pseudo-inverse
  - In fact, the problem is computationally very hard
  - The optimization tends to stuck in local optima

Boolean CP: Walk’n’Merge

- Idea: For **exact** decomposition, each rank-1 tensor should correspond to an all-1s subtensor
  - Knowing these, we ”only” need to know how to use them
- For approximate decompositions, we need **dense** rank-1 subtensors
  - $\sum_{ijk} x_{ijk}[a_i=1][b_j=1][c_k=1] \approx ||\boldsymbol{a}||^2 \cdot ||\boldsymbol{b}||^2 \cdot ||\boldsymbol{c}||^2$

Erdős, D., & Miettinen, P. (2013). Walk’n’Merge: A Scalable Algorithm for Boolean Tensor Factorization (pp. 1037–1042). Presented at the 13th IEEE International Conference on Data Mining. doi:10.1109/ICDM.2013.141

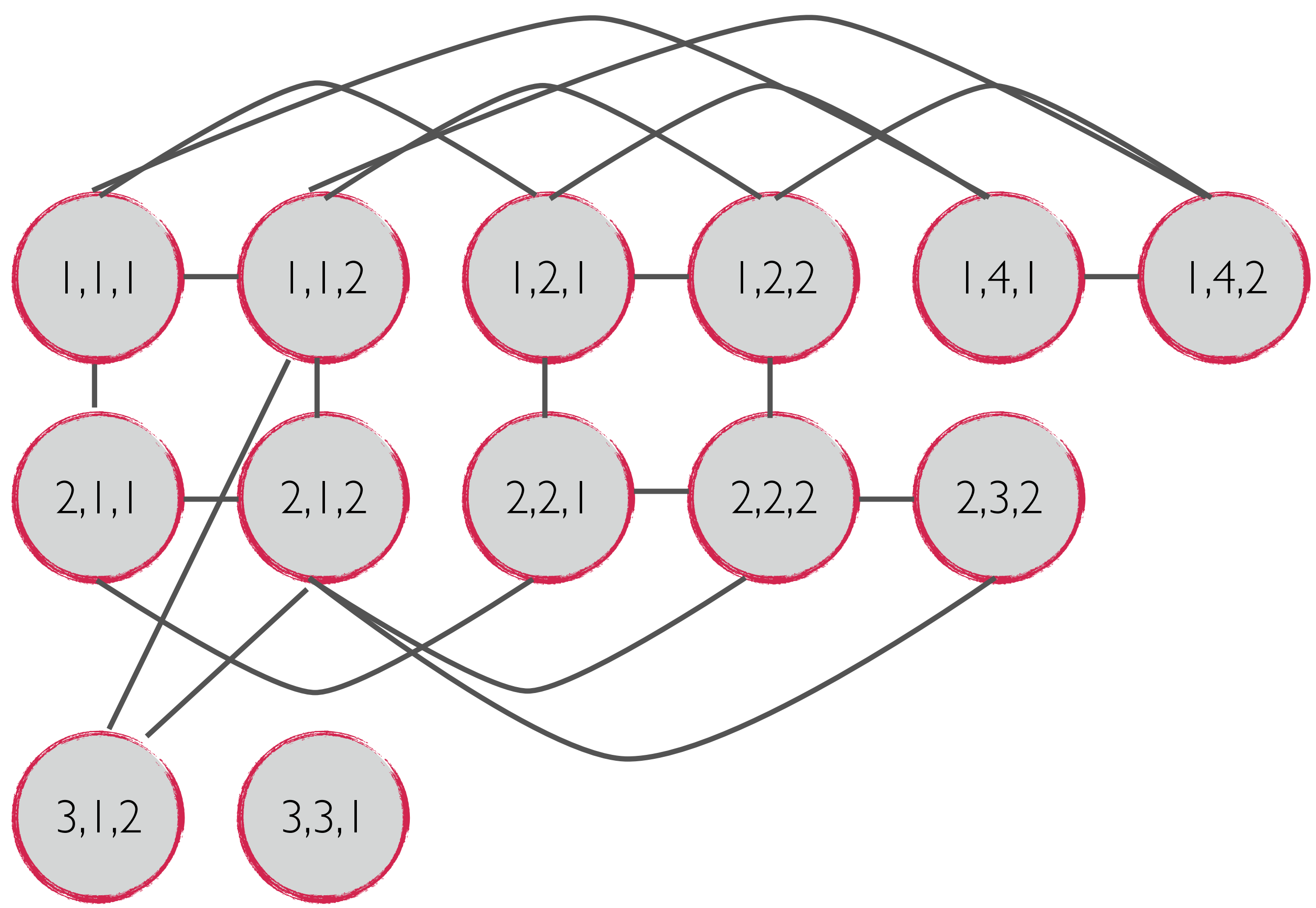
Finding Dense Subtensors: Graph POW

- Think the binary tensor $\mathcal{X}$ as a graph G
  - Every x_{ijk}=1 is a vertex
  - There's an edge between x_{ijk} and $x_{\alpha\beta\gamma}$ iff x_{ijk} and $x_{\alpha\beta\gamma}$ are on the same slice
    - $i=\alpha$ and $j=\beta$; or
    - $i=\alpha$ and $k=\gamma$; or
    - $j=\beta$ and $k=\gamma$

Graph Example

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

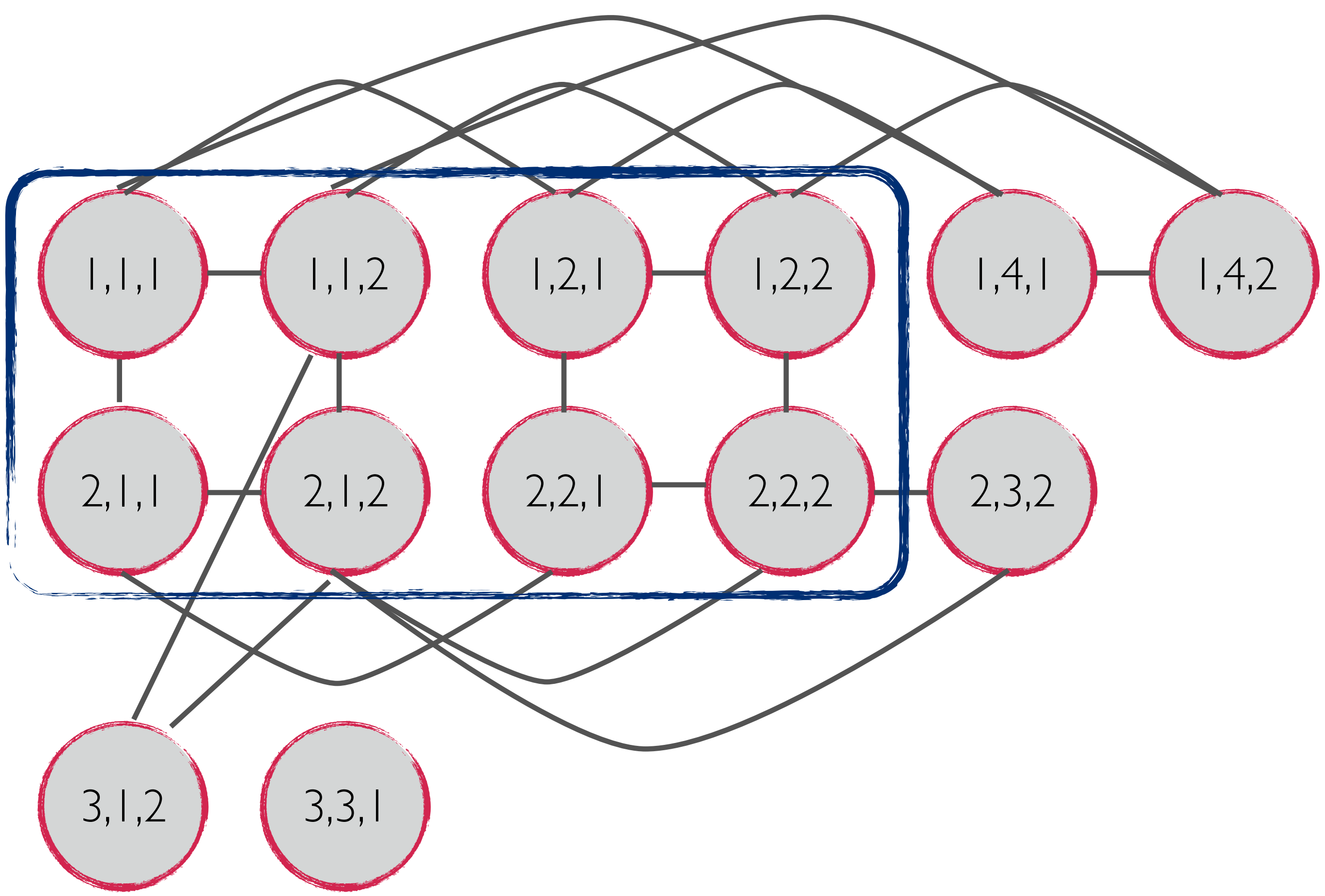


Dense Subtensors in Graphs

- Let S_1, S_2, and S_3 be sets of integers s.t. $x_{ijk} = 1$ for all $(i,j,k) \in S_1 \times S_2 \times S_3$
  - All-1s subtensor
- If (i,j,k), $(\alpha,\beta,\gamma) \in S_1 \times S_2 \times S_3$, then x_{ijk} is at most three steps from $x_{\alpha\beta\gamma}$ in the graph
  $\Rightarrow$ Dense subtensors = small-diameter subgraphs

Graph Example

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$



Finding Small-Diameter Subgraphs

- Small-diameter subgraphs can be found using **random walks with re-starts**
  1. Do a short random walk from a random node
  2. Do a new walk from a node you have visited
  3. Repeat 2 many times
  4. Take the smallest rank-1 binary subtensor containing all often-visited nodes and check if it is dense w.r.t. user-specified threshold

Post-Processing Dense Subtensors

- We might have found highly overlapping subtensors
  - Try merging overlapping subtensors if the result is dense enough
- We can also add all very small all-1s subtensors
  - E.g. 2-by-2-by-2
  - Hard to find using random walks

Final Steps

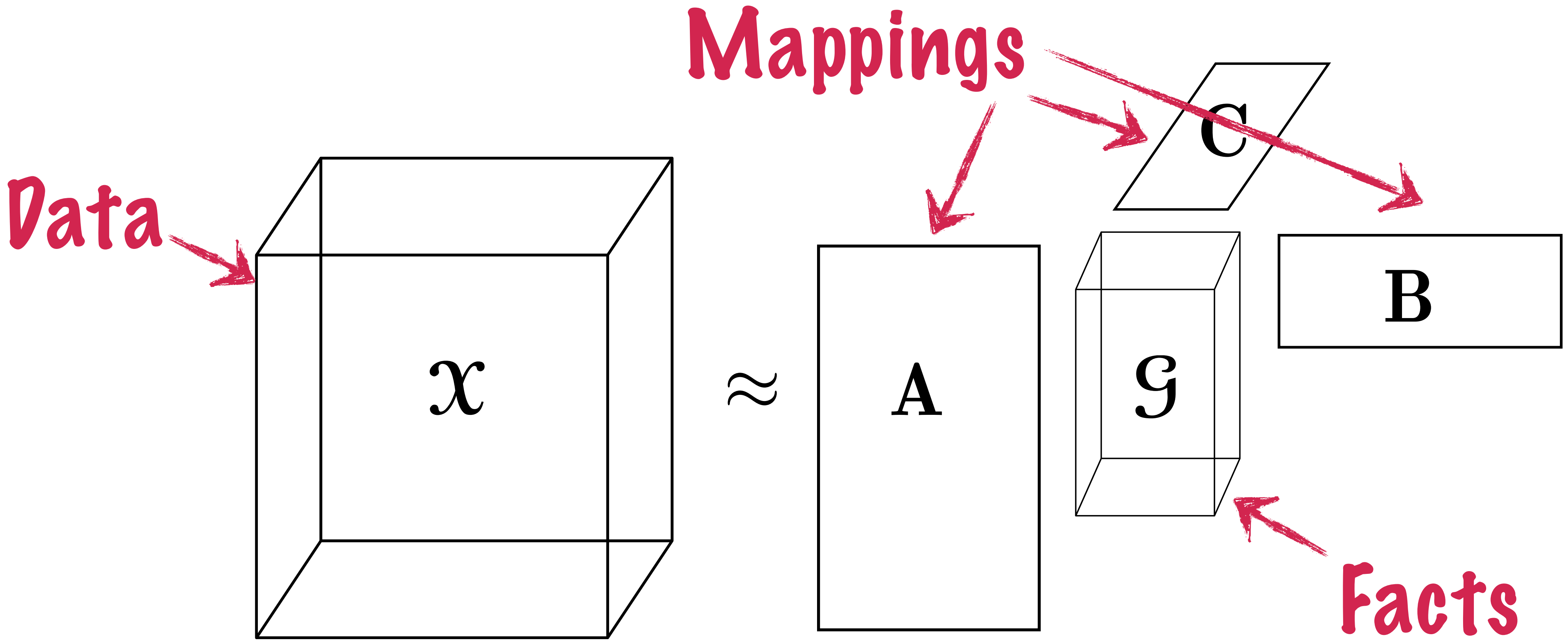
- To obtain a CP decomposition, select the best rank-1 components
  - Actually a complicated problem
- To obtain a Tucker3 decomposition:
  - Start with hyperdiagonal core
  - If two columns in a factor matrix are very similar, merge them, and correct the core accordingly
    - Remove a dimension
    - Add 1 off-hyperdiagonal

Boolean Tucker3 Application: Fact Discovery

- **Input**: noun phrase—verbal phrase—noun phrase triples
  - JFK—was shot in—Dallas
  - John F. Kennedy—was assasinated in—Dallas, TX
- **Goal**: find the entities, relations, and facts (entity—relation—entity triples)

Erdős, D., & Miettinen, P. (2013). Discovering Facts with Boolean Tensor Tucker Decomposition (pp. 1569–1572). Presented at the 2013 ACM International Conference on Information and Knowledge Management. doi:10.1145/2505515.2507846

Facts and Boolean Tucker3



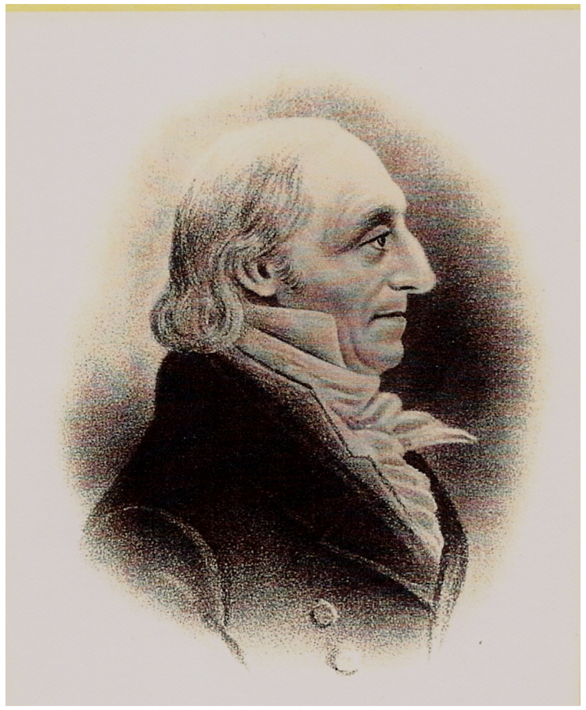
$$x_{ijk} \approx \bigvee_{p=1}^{P} \bigvee_{q=1}^{Q} \bigvee_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr}$$

Example Result

Subject: claude de lorimier, de lorimier, louis, jean-baptiste

Relation: was born, [[det]] born in

Object: borough of lachine, villa st. pierre, lachine quebec



39,500-by-8,000-by-21,000 tensor

with 804 000 non-zeros

Summary

- Not every tensor decomposition needs to use multi-linear algebra
  - The correct model depends on the data and what one wants to find
- Usually non-linear models are even harder to optimize

Suggested Reading

- All from the previous lectures
- Bottom-of-the-slides links
- Miettinen, P. (2011). Boolean Tensor Factorizations (pp. 447–456). Presented at the 11th IEEE International Conference on Data Mining.
  - Basics of Boolean tensor factorizations