CS 598 EVS: Tensor Computations

Tensor Decomposition

Edgar Solomonik

University of Illinois at Urbana-Champaign

CP Decomposition Rank

- The *canonical polyadic or CANDECOMP/PARAFAC (CP) decomposition* expresses an order d tensor in terms of d factor matrices
  - *For $\mathcal{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, a rank R CP decomposition is defined by matrices $\boldsymbol{U}^{(i)} \in \mathbb{R}^{s_i \times R}$ so that*

$$t_{i_1 \ldots i_d} = \sum_{r=1}^{R} \prod_{j=1}^{d} u_{i_j r}^{(j)}$$

  - *The CP decomposition is also often denoted by*

$$\mathcal{T} = [\![\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{U}^{(d)}]\!]$$

  - *First proposed by Hitchcock in 1927*
  - *Given a tensor, the smallest R for which it has a CP decomposition is the tensor rank, also sometimes referred to as the CP rank or canonical rank*
  - *Finding the CP rank and associated decomposition enables automatic derivation of bilinear algorithms among other applications reviewed later in this lecture, which mostly follows T. Kolda and B. Bader "Tensor Decompositions and Applications", SIAM Review 2009.*

Tensor Rank Properties

- Tensor rank does not satisfy many of the properties of matrix rank
  - *Rank of a real-valued tensor can be different over the complex field, e.g., for*

$$\mathcal{T} = \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \right]$$

which is a perfectly conditioned tensor, the rank over $\mathbb{R}$ *is* 3 *but over* $\mathbb{C}$ *it is* 2*,*

$$\mathcal{T} = [\![\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -i & i \end{bmatrix}, \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix}, \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ i & -i \end{bmatrix}]\!]$$

  - *The maximal possible rank of tensors of particular dimensions is often unequal to the typical tensor rank, which is any rank for which the set of tensors of a given size with that rank has positive volume*
  - $2 \times 2 \times 2$ *tensors have typical ranks* 2 *(79%) and* 3 *(21%) over* $\mathbb{R}$*, and typical rank* 2 *over* $\mathbb{C}$

Typical Rank and Generic Rank

- When there is only a single typical tensor rank, it is the *generic rank*
  - *For decomposition over $\mathbb{C}$, tensors have a single generic rank*
  - *If we restrict to symmetric tensors of order d and dimension n, the generic rank over $\mathbb{C}$ is*

$$R = \left\lceil \binom{n+d-1}{d} / n \right\rceil$$

except when $(d, n) \in \{(3,5), (4,3), (4,4), (4,5)\}$ in which cases it should be increased by one

  - *This rank bound makes sense, as the total amount of information in a single factor matrix is $nR \approx \binom{n+d-1}{d}$, which matches the number of unique entries in the symmetric tensor*
  - *For maximal rank of an $n_1 \times n_2 \times n_3$ tensor, the maximal rank is bounded (weakly) by $R \leqslant \min(n_1 n_2, n_1 n_3, n_2 n_3)$, which follows by the same intuition*

Uniqueness Sufficient Conditions

- Unlike the low-rank matrix case, the CP decomposition can be unique
  - *In the matrix case, given $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{V}^T$, for any invertible $\boldsymbol{M}$ we can obtain a new factorization $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{M}(\boldsymbol{V}\boldsymbol{M}^{-1})^T$*
  - *In CP decomposition, the indeterminacy is generally limited to permutation of the R rank-1 factors and scaling of their components*
  - *Modulo permutation and scaling, strong conditions exist on uniqueness of the CP decomposition*
  - *Define the k-rank of a matrix as the maximum value of k such that any k columns of the matrix are linearly independent*
  - *For an order 3 tensor with CP decomposition $[\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$ where the factor matrices have k-ranks k_A, k_B, and k_C, a sufficient condition for uniqueness is*

$$k_A + k_B + k_C \geqslant 2R + 2$$

  - *For order d tensors whose CP decomposition is composed of matrices with has k ranks $k_1, \ldots, k_d$, the sufficient condition is*

$$\sum_{i=1}^{d} k_i \geqslant 2R + (d - 1)$$

Uniqueness Necessary Conditions

- Necessary conditions for uniqueness of the CP decomposition also exist
  - *A simple necessary condition for uniqueness is*

$$\min_{l \in \{1,\ldots,d\}} \Big(\prod_{i=1, i \neq l}^{d} \operatorname{rank}(\boldsymbol{U}^{(i)}) \Big) \geqslant R$$

  - *This condition stems from the more general restriction that*

$$\min_{l \in \{1,\ldots,d\}} \operatorname{rank} \Big(\bigodot_{i=1, i \neq l}^{d} \boldsymbol{U}^{(i)} \Big) \geqslant R$$

  - *When one of the d Khatri-Rao products is rank deficient, multiple (infinite) choices of the lth factor matrices must bring the residual to zero*

Degeneracy

- The best rank-k approximation may not exist, a problem known as *degeneracy* of a tensor
  - *Consider a rank* 3 *tensor*

$$\mathcal{T} = \boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \otimes \boldsymbol{c}_2 + \boldsymbol{a}_1 \otimes \boldsymbol{b}_2 \otimes \boldsymbol{c}_1 + \boldsymbol{a}_2 \otimes \boldsymbol{b}_1 \otimes \boldsymbol{c}_1$$

  where the factor matrices are defined to be orthogonal
  - *The tensor can be approximated arbitrarily closely by*

$$\mathcal{W}^{(\alpha)} = \alpha(\boldsymbol{a}_1 + \frac{1}{\alpha}\boldsymbol{a}_2) \otimes (\boldsymbol{b}_1 + \frac{1}{\alpha}\boldsymbol{b}_2) \otimes (\boldsymbol{c}_1 + \frac{1}{\alpha}\boldsymbol{c}_2) - \alpha \boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \otimes \boldsymbol{c}_1$$

  in particular

$$\lim_{\alpha \to \infty} \|\mathcal{W}^{(\alpha)} - \mathcal{T}\| = 0$$

  - *Consequently, the best rank-*2 *approximation does not exist for this tensor, as in the limit* $\mathcal{W}^{(\alpha)}$ *converges to an order* 3 *tensor*

Border Rank

- Degeneracy motivates an approximate notion of rank, namely *border rank*
  - *The border rank of a tensor $\mathcal{T}$ is defined as the smallest R such that, for any $\epsilon > 0$, there exists a rank R tensor $\mathcal{W}$ such that*

$$\|\mathcal{T} - \mathcal{W}\| < \epsilon$$

  - *The border rank is always less than the rank of a tensor, but can also be smaller*
  - *The concept of border rank has been intensively used to find fast bilinear algorithms for matrix multiplication*
  - *The border rank and rank of the $4 \times 4 \times 4$ multiplication tensor are both 7, yielding Strassen's algorithm*
  - *For the $9 \times 9 \times 9$ tensor defining multiplication of 3×3 matrices, determining rank and border rank is an open problem, the rank is between 19 and 23, while the border rank is between 14 and 21*

Approximation by CP Decomposition

- Approximation via CP decomposition is a nonlinear optimization problem
  - *Given order d tensor $\mathcal{T}$ with all dimensions equal to n, the rank-R CP approximation problem can be written as*

$$\min_{\boldsymbol{U}^{(1)},\ldots,\boldsymbol{U}^{(n)}\in\mathbb{R}^{n\times R}} \underbrace{\frac{1}{2}\|\mathcal{T}-[\![\boldsymbol{U}^{(1)},\ldots,\boldsymbol{U}^{(d)}]\!]\|_F^2}_{\phi(\boldsymbol{U}^{(1)},\ldots,\boldsymbol{U}^{(d)})}$$

  - *The gradient of this objective function is*

$$\nabla\phi = \left[d\phi/d\boldsymbol{U}^{(1)}\ldots d\phi/d\boldsymbol{U}^{(d)}\right]$$

  - *Each component of the gradient has the form*

$$\frac{d\phi}{d\boldsymbol{U}^{(i)}}(\boldsymbol{U}^{(1)},\ldots,\boldsymbol{U}^{(d)}) = \boldsymbol{U}^{(i)} *_{j=1,j\neq i}^{d} \boldsymbol{U}^{(j)^T}\boldsymbol{U}^{(j)} - \underbrace{\mathcal{T}_{(i)} \bigodot_{j=1,j\neq i}^{d} \boldsymbol{U}^{(j)}}_{\text{MTTKRP}}$$

  - *Unless R is very large, computing $\frac{d\phi}{d\boldsymbol{U}^{(i)}}(\boldsymbol{U}^{(1)},\ldots,\boldsymbol{U}^{(d)})$ is not much cheaper than minimizing ϕ w.r.t. $\boldsymbol{U}^{(i)}$ by solving for $\boldsymbol{U}^{(i)}$ in $\frac{d\phi}{d\boldsymbol{U}^{(i)}}(\boldsymbol{U}^{(1)},\ldots,\boldsymbol{U}^{(d)}) = \boldsymbol{0}$*

Alternating Least Squares Algorithm

- The standard approach for finding an approximate or exact CP decomposition of a tensor is the *alternating least squares (ALS) algorithm*
  - *Consider rank R decomposition of a tensor $\mathcal{T} \in \mathbb{R}^{n \times n \times n}$ over $\mathbb{R}$*
  - *A sweep takes as input $[\![\boldsymbol{U}^{(k)}, \boldsymbol{V}^{(k)}, \boldsymbol{W}^{(k)}]\!]$ solves 3 quadratic optimization problems to obtain $[\![\boldsymbol{U}^{(k+1)}, \boldsymbol{V}^{(k+1)}, \boldsymbol{W}^{(k+1)}]\!]$, updating each factor matrix in sequence, typically via the normal equations:*

$$(\boldsymbol{V}^{(k)^T}\boldsymbol{V}^{(k)} * \boldsymbol{W}^{(k)^T}\boldsymbol{W}^{(k)})\boldsymbol{U}^{(k+1)} = \boldsymbol{T}_{(1)}(\boldsymbol{V}^{(k)} \odot \boldsymbol{W}^{(k)})$$

$$(\boldsymbol{U}^{(k+1)^T}\boldsymbol{U}^{(k+1)} * \boldsymbol{W}^{(k)^T}\boldsymbol{W}^{(k)})\boldsymbol{V}^{(k+1)} = \boldsymbol{T}_{(2)}(\boldsymbol{U}^{(k+1)} \odot \boldsymbol{W}^{(k)})$$

$$(\boldsymbol{U}^{(k+1)^T}\boldsymbol{U}^{(k+1)} * \boldsymbol{V}^{(k+1)^T}\boldsymbol{V}^{(k+1)})\boldsymbol{W}^{(k+1)} = \boldsymbol{T}_{(3)}(\boldsymbol{U}^{(k+1)} \odot \boldsymbol{V}^{(k+1)})$$

  - *Residual decreases monotonically, since the subproblems in each subset of nR variables are quadratic*
  - *Forming the linear equations has cost $O(dnR^2)$ while forming the right-hand-sides requires an MTTKRP with cost $O(n^dR)$*

Properties of Alternating Least Squares for CP

- *CP-ALS achieves linear local convergence to local minima of our objective ϕ*
  - *this follows from the equivalence of the optimality conditions (vanishing gradient) and the ALS update rule*
  - *no global convergence guarantees are available, and in practice algorithm convergence can stagnate, typically due to the factor matrix iterates becoming ill-conditioned*
- *CP-ALS guarantees monotonic decrease in residual*
  - *the exact minimizer is found for each quadratic subproblem, which cannot be worse than the previous choice*
- *the equations for each subproblem are formed by a Khatri-Rao product, which makes subproblems amenable to fast approximate methods*

Alternating Least Squares for Tucker Decomposition

- For Tucker decomposition, an analogous optimization procedure to ALS is referred to as *high-order orthogonal iteration (HOOI)*
  - *Each component of the derivative of the Tucker approximation objective function with respect to the product of a factor matrix and the core tensor is a TTMc (as opposed to MTTKRP in the CP case)*

$$\psi(\mathcal{Z}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) = \frac{1}{2}(t_{ijk} - \sum_{pqr} z_{pqr} u_{ip} v_{jq} w_{kr})^2$$

$$\frac{d\psi}{d(\mathcal{Z} \times_1 \boldsymbol{U})}(\mathcal{Z}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) = \sum_{j,k} t_{ijk} v_{jq} w_{kr} - \sum_{pq'r'} z_{pqr} u_{ip} \underbrace{(\sum_j v_{jq'} v_{jq})}_{\delta(q,q')} \underbrace{(\sum_k w_{kr'} w_{kr})}_{\delta(r,r')}$$

  - *Consequently, we can find the minimizing $\mathcal{Z} \times_1 \boldsymbol{U}$ by SVD of the mode-1 unfolding of the TTMC $\mathcal{T} \times_2 \boldsymbol{V}^T \times_3 \boldsymbol{W}^T$, which is a $s \times R^2$ matrix*
  - *Optimizing for a single factor matrix in this way costs $O(s^d R + s R^d)$*
  - *A sweep of HOOI requires forming N such TTMcs and computing their SVDs*

Dimension Trees for ALS

- The cost of ALS can be reduced by amortizing computation common terms
  - *The cost of ALS is typically dominated by MTTKRPs, d of which are computed for each sweep, for $d = 3$,*

$$\boldsymbol{T}_{(1)}(\boldsymbol{V}^{(k)} \odot \boldsymbol{W}^{(k)}), \boldsymbol{T}_{(2)}(\boldsymbol{U}^{(k+1)} \odot \boldsymbol{W}^{(k)}), \boldsymbol{T}_{(3)}(\boldsymbol{U}^{(k+1)} \odot \boldsymbol{V}^{(k+1)})$$

  - *Note that given $\mathcal{Z} = \mathcal{T} \times_3 \boldsymbol{W}^{(k)^T}$, we can compute the first two MTTKRPs with $O(s^2R)$ cost, since*

$$\sum_{j,l} t_{ijl} v_{jr}^{(k)} w_{lr}^{(k)} = \sum_{j} z_{ijr} v_{jr}^{(k)} \quad \textit{and} \quad \sum_{j,l} t_{ijl} u_{ir}^{(k+1)} w_{lr}^{(k)} = \sum_{i} z_{ijr} u_{ir}^{(k+1)}$$

  - *In general, we can reuse a single TTM to compute the next $d-1$ sets of right-hand-sides (MTTKRPs) in ALS (in this sweep or the next sweep)*
  - *The amortized cost of each ALS sweep (assuming Strassen-like matrix-multiplication algorithms are not used) is then given by $\frac{2d}{d-1} s^d R + O(ds^{d-1}R) + O(dR^3)$ where the final term comes from Cholesky factorization of the matrices $\boldsymbol{G}^{(i)} = *_{j=1, j\neq i}^{d} \boldsymbol{U}^{(j)^T} \boldsymbol{U}^{(j)}$*

Fast Residual Norm Calculation

- Calculating the norm of the residual has cost $2ds^dR$, but can be done more cheaply within ALS
  - *We can expand the square of the residual norm as follows*

$$\|\boldsymbol{\mathcal{T}} - [\![\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}]\!]\|_F^2 = \|\boldsymbol{\mathcal{T}}\|_F^2 - \sum_{ijkr} t_{ijk} u_{ir} v_{jr} w_{kr} + \sum_{rs} (\sum_i u_{ir} u_{is})(\sum_j v_{jr} v_{js})(\sum_k w_{kr} w_{ks})$$

  - *The first term does not change across ALS sweeps, so it suffices to compute it once*
  - *The second term can be obtained cheaply from any MTTKRP with $d-1$ of the factor matrices*
  - *The third term can be computed from Gram matrices that are needed for ALS anyway*

Pairwise Perturbation Algorithm

- A route to further reducing the cost of ALS is to perform it approximately via *pairwise perturbation*
  - *ALS convergence can be slow with factor matrices changing little at every iteration*
  - *In such cases, can leverage perturbative approximation to compute each MTTKRP approximately using old intermediates*
  - *Given order 3 tensor $\mathcal{T}$ can compute intermediates*

$$\mathcal{Z}^{(1)} = \mathcal{T} \times_1 \boldsymbol{U}^{(k)^T}, \mathcal{Z}^{(2)} = \mathcal{T} \times_2 \boldsymbol{V}^{(k)^T}, \mathcal{Z}^{(3)} = \mathcal{T} \times_3 \boldsymbol{W}^{(k)^T}$$

and approximate for subsequent iteration $k' > k$ so long as $\|\boldsymbol{U}^{(k')} - \boldsymbol{U}^{(k)}\|_F, \|\boldsymbol{V}^{(k')} - \boldsymbol{V}^{(k)}\|_F, \|\boldsymbol{W}^{(k')} - \boldsymbol{W}^{(k)}\|_F < \epsilon$,

$$\sum_{j,l} t_{ijl} v_{jr}^{(k')} w_{lr}^{(k')} = \sum_j z_{ijr}^{(3)} v_{jr}^{(k')} + \sum_{j,l} t_{ijl} v_{jr}^{(k')} (w_{lr}^{(k')} - w_{lr}^{(k)}) \approx \sum_j z_{ijr}^{(3)} v_{jr}^{(k')}$$

  - *For general order d tensors, each approximated sweep for $k' > k$ has cost $d(d-1)s^2R + O(dR^3)$*
  - *For Tucker, dimension trees and pairwise perturbation work similarly*

Pairwise Perturbation Second Order Correction

- When approximating a tensor using CP, the partially converged CP factors can sometimes be used in place of the tensor to accelerate cost
  - *An example of this idea is the second order correction to pairwise perturbation*
  - *The truncated term in the pairwise perturbation algorithm can be approximated by approximating the tensor $\mathcal{T}$ with its current approximate CP decomposition*

$$\sum_{j,l} t_{ijl} v_{jr}^{(k')} (w_{lr}^{(k')} - w_{lr}^{(k)}) \approx \sum_{j,l,q} u_{iq}^{(k')} v_{jq}^{(k')} w_{lq}^{(k')} v_{jr}^{(k')} (w_{lr}^{(k')} - w_{lr}^{(k)})$$

  - *These contractions can be evaluated with cost $O(sR^2)$*

$$\sum_{j,l,q} u_{iq}^{(k')} v_{jq}^{(k')} w_{lq}^{(k')} v_{jr}^{(k')} (w_{lr}^{(k')} - w_{lr}^{(k)}) = \sum_{q} u_{iq}^{(k')} (\sum_{j} v_{jq}^{(k')} v_{jr}^{(k')}) (\sum_{l} w_{lq}^{(k')} (w_{lr}^{(k')} - w_{lr}^{(k)}))$$

Gauss-Newton Algorithm

- ALS generally achieves linear convergence, while Newton-based methods can converge quadratically
  - *Derive these by casting CP as a nonlinear least squares problem,*

$$\phi(\boldsymbol{x}) = \frac{1}{2} \| \underbrace{\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x})}_{\boldsymbol{r}(\boldsymbol{x})} \|^2$$

  - *Newton's method computes* $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{H}_\phi(\boldsymbol{x})^{-1} \nabla \phi(\boldsymbol{x})$
  - *For nonlinear least squares problems, the gradient and Hessian are*

$$\nabla \phi(\boldsymbol{x}) = \boldsymbol{J}_r^T(\boldsymbol{x}) \boldsymbol{r}(\boldsymbol{x}),$$
$$\boldsymbol{H}_\phi(\boldsymbol{x}) = \boldsymbol{J}_r^T(\boldsymbol{x}) \boldsymbol{J}_r(\boldsymbol{x}) + \sum_i r_i(\boldsymbol{x}) \boldsymbol{H}_{r_i}(\boldsymbol{x})$$

  - *The Gauss-Newton method approximates* $\boldsymbol{H}_\phi(\boldsymbol{x}) \approx \boldsymbol{J}_r^T(\boldsymbol{x}) \boldsymbol{J}_r(\boldsymbol{x})$*, so*

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{s}^{(k)}, \quad \boldsymbol{s}^{(k)} = (\boldsymbol{J}_r^T(\boldsymbol{x}^{(k)}) \boldsymbol{J}_r(\boldsymbol{x}^{(k)}))^{-1} \boldsymbol{J}_r^T(\boldsymbol{x}^{(k)}) \boldsymbol{r}(\boldsymbol{x}^{(k)}),$$
$$\boldsymbol{J}_r(\boldsymbol{x}^{(k)}) \boldsymbol{s}^{(k)} \cong \boldsymbol{r}(\boldsymbol{x}^{(k)})$$

Gauss-Newton for CP Decomposition

- CP decomposition for order $d = 3$ tensors ($d > 3$ is similar) minimizes

$$\phi(\boldsymbol{U}^{(1)}, \boldsymbol{U}^{(2)}, \boldsymbol{U}^{(3)}) = \frac{1}{2} \sum_{ijk} \left(t_{ijk} - \sum_{r=1}^{R} u_{ir}^{(1)} u_{jr}^{(2)} u_{kr}^{(3)} \right)^2$$

  - *The Gauss-Newton approximate Hessian is* $dnR \times dnR$*,*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}^{(1,1)} & \cdots & \boldsymbol{H}^{(1,d)} \\ \vdots & \ddots & \vdots \\ \boldsymbol{H}^{(d,1)} & \cdots & \boldsymbol{H}^{(d,d)} \end{bmatrix}, \text{ where } \boldsymbol{H}^{(q,q)} = \boldsymbol{G}^{(n,n)} \otimes \boldsymbol{I}$$

while for $q \neq p$, $\quad h_{krlz}^{(q,p)} = u_{kz}^{(q)} u_{lr}^{(p)} g_{rz}^{(q,p)}$,

where in both cases $\quad g_{rz}^{(n,p)} = \prod_{m=1, m \neq q,p}^{d} \left(\sum_{i} u_{ir}^{(m)} u_{iz}^{(m)} \right)$

Gauss-Newton for CP Decomposition

- A step of Gauss-Newton requires solving a linear system with $\boldsymbol{H}$
  - *Cholesky of* $\boldsymbol{H}$ *requires* $O(d^2n^2R^2)$ *memory and cost* $O(d^3n^3R^3)$
  - *Matrix-vector product with* $\boldsymbol{H}$ *can be computed with cost* $O(d^2nR^2)$
  - *Can use CG method with implicit matrix-vector product*[1]
  - *Can formulate product* $\boldsymbol{u} = \boldsymbol{H}\boldsymbol{v}$ *using tensor contractions each with cost* $O(nR^2)$

```
u = []
for q in range(d):
  u.append(zeros((n,R)))
  for p in range(d):
    if q == p:
      u[q] += einsum("rz,kz->kr",G[q,p],v[p])
    else:
      u[q] += einsum("kz,lr,rz,lz->kr", \
                     U[q],U[p],G[q,p],v[p])
```

[1] P. Tichavsky, A. H. Phan, and A. Cichocki, 2013

Matrix Sketching

Randomized methods provide accurate approximate solutions to linear least squares problems, which can be applied to accelerate ALS, as well as more basic problems

- *Consider a linear least squares problem with* $\boldsymbol{A} \in \mathbb{R}^{m \times n}$

$$\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$$

- *We seek to find a sketch matrix* $\boldsymbol{S} \in \mathbb{R}^{s \times m}$, $s < m$ *so as to obtain an approximate solution* $\hat{\boldsymbol{x}} \approx \boldsymbol{x}$ *via*

$$\boldsymbol{S}\boldsymbol{A}\hat{\boldsymbol{x}} \cong \boldsymbol{S}\boldsymbol{b}$$

- *We choose* $\boldsymbol{S}$ *from a random distribution so that* $\boldsymbol{S}\boldsymbol{A}$ *is efficient to compute, and with high-probability, the residual is not much larger than optimal,*

$$\frac{\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}\|_2 - \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2}{\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2} \leqslant \epsilon$$

Random Projections

Accuracy of sketching techniques is theoretically characterized by statistical analysis

- *To ascertain accuracy of sketching for matrix/vector products and linear least squares, we want to bound error of* $\langle \boldsymbol{Su}, \boldsymbol{Sv} \rangle$ *relative to* $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ *for some set of pairs* $\boldsymbol{u}, \boldsymbol{v}$
- *Generally, its easy to enforce that the result computed in expectation agrees with the exact solution*
- *To bound error, it then suffices to bound the variance of the random variable, e.g., via Chernoff bounds*

Johnson-Lindenstrauss Lemma

The Johnson-Lindenstrauss lemma is a powerful tool for obtaining error bounds in a projected vector space

$$\boldsymbol{SA\hat{x}} \cong \boldsymbol{Sb}$$

- *For $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ if $\boldsymbol{S} \in \mathbb{R}^{k \times d}$ is elementwise Gaussian-random, then if $k \geqslant 9 \log n/(\epsilon^2 - \epsilon^3)$, then with probability 1/2, all projected distances $\|\boldsymbol{u}_i - \boldsymbol{u}_j\|_2$ are preserved to within a relative error of ϵ*
- *Since the squared distance $\|\boldsymbol{u}_i - \boldsymbol{u}_j\|_2^2 = \|\boldsymbol{u}_i\|_2^2 + \|\boldsymbol{u}_j\|_2^2 - 2\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle$ and the projected vector norms are also preserved accurately w.h.p., it follows that we can also sketch $\langle \boldsymbol{u}_i, \boldsymbol{u}_j \rangle$ accurately*
- *For a more comprehensive review of these results and derivations, see the following notes by Michael Mahoney* `https://arxiv.org/pdf/1608.04481.pdf`

Matrix Sketching

The best choice of sketch matrix depends on the desired accuracy and the structure of $\boldsymbol{A}$

- *the matrix $\boldsymbol{A}$ may be sparse or structured, e.g., for ALS $\boldsymbol{A}$ is a Khatri-Rao product, while for HOOI, $\boldsymbol{A}$ is a Kronecker product of orthogonal matrices*
- *for dense matrices, random Gaussian sketches or sketches that can be applied with FFT or FFT-like algorithms are fastest*
- *for sparse matrices, ideally we wish to sample the rows of $\boldsymbol{A}$ or at least minimally mix them, for which we need a sparse $\boldsymbol{S}$*
- *the sketched solve can also be applied as a preconditioner in order to improve accuracy*

Matrix Sketching via Sampling

Uniform sampling of rows is insufficient to obtain good accuracy in general

- *Consider an $\boldsymbol{A}$ with a single row that is orthogonal to the rest, if it is not sampled, all of $\boldsymbol{x}$ spanned by this row will be missed in $\hat{\boldsymbol{x}}$*
- *The component of the right-hand-side in the span of the columns of $\boldsymbol{A} = \boldsymbol{QR}$ is $\boldsymbol{QQ}^T\boldsymbol{b}$*
- *After sketching, we reduce the row space, and can capture less of the right-hand side $\boldsymbol{SQ}(\boldsymbol{SQ})^\dagger\boldsymbol{Sb}$*
- *The largest components of $\boldsymbol{QQ}^T\boldsymbol{b}$ will depend on the largest row-norms of $\boldsymbol{Q}$*

Leverage score sampling provides better accuracy guarantees

- *The leverage scores of $\boldsymbol{A}$ are given by $l_i = \|\boldsymbol{q}_i\|_2^2$ where $\boldsymbol{q}_i$ is the ith row of $\boldsymbol{Q}$*
- *We can define $\boldsymbol{S}$ to sample the rows of $\boldsymbol{A}$ with probability l_i/n, since $\sum_i l_i = n$*
- *With this choice, a sample size of $s = O(n/\epsilon^2)$ suffices to get within $1+\epsilon$ of the optimal residual*
- *While reducing row dimension from m to $O(n/\epsilon^2)$ is beneficial for small n, computing leverage scores is expensive (requires QR factorization)*

Mixing Techniques

To circumvent leverage score sampling, we can mix rows randomly

- *Choosing elements of S from a Gaussian distribution, results in intermixing of rows of Q*
- *In effect, this corresponds to uniform sampling with close-to-uniform leverage scores*
- *The resulting sample complexity is as good as leverage-score-based sampling*

Instead of choosing elements of S randomly, pseudo-random distributions allow S to be applied more rapidly

- *Choose $S = PFD$ where D is a random diagonal sign matrix, F is the DFT or Hadamard transform, and P performs uniform-random sampling*
- *Application of orthogonal matrix FD results in a random intermixing of rows, does not modify solution x*
- *P performs sampling of a matrix with close-to-uniform leverage scores*
- *Same sample size complexity as Gaussian mixing and leverage score sampling*

Approximate CP ALS using Random Sampling

- Another approach to approximating ALS is to sample the least-squares equations[2]
  - *Recall, that in the least squares view, in the first step of ALS we seek to optimize* $\boldsymbol{U}$ *in* $\boldsymbol{\mathcal{T}} \approx [\![\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}]\!]$*, yielding*

$$(\boldsymbol{V} \odot \boldsymbol{W})\boldsymbol{U}^T \cong \boldsymbol{T}_{(1)}^T$$

  - *Instead, we can sample* S *of the equations and solve*

$$\boldsymbol{S}(\boldsymbol{V} \odot \boldsymbol{W})\boldsymbol{U}^T \cong \boldsymbol{S}\boldsymbol{T}_{(1)}^T$$

    where the rows of $\boldsymbol{S} \in \mathbb{R}^{S \times n^2}$ *are sampled uniformly (with replacement) from the identity matrix*
  - *This approach yields cost* $O(S \times R^2 + n^2 SR)$ *per iteration, as opposed to the usual* $O(n^3 R + R^3)$
  - *An variation of this method is to mix the equations, e.g., by picking* $\boldsymbol{S} = \boldsymbol{S}'(\boldsymbol{F}\boldsymbol{D}_1 \otimes \boldsymbol{F}\boldsymbol{D}_2)\boldsymbol{D}$ *where* $\boldsymbol{D}_1$ *and* $\boldsymbol{D}_2$ *are random diagonal sign matrices,* $\boldsymbol{F}$ *is the FFT matrix, and* $\boldsymbol{S}'$ *is a sampling matrix as before*
  - *The FFTs can then be applied to the tensor as a preprocessing step*

[2] C. Battaglino, G. Ballard, T. G. Kolda, 2018

Tensor Completion

- The *tensor completion* problem seeks to build a model (e.g., CP decomposition) for a partially-observed tensor
  - *Completion differs from decomposition of a sparse tensor with zeros for entries that are unobserved, as the CP decomposition would be fitting the zeros, which we do not want*
  - *For an order three tensor $\mathcal{T} \in \mathbb{R}^{n \times n \times n}$, given a set of observed entries t_{ijk} for $(i,j,k) \in \Omega$, we seek to minimize*

$$f(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) = \sum_{(i,j,k) \in \Omega} (t_{ijk} - \sum_{r} u_{ir} v_{jr} w_{kr})^2 + \lambda^2 (\|\boldsymbol{U}\|_2^2 + \|\boldsymbol{V}\|_2^2 + \|\boldsymbol{W}\|_2^2)$$

- The problem was partially popularized by the Netflix prize collaborative filtering problem
  - *This problem involved building a model for predicting user ratings of movies, given the set of movies they have already rated, with each rating corresponding to a tuple (user, movie, date), which can be enumerated in a tensor*
  - *Aside from this problem, tensor completion is a natural generalization of the matrix completion problem, which is intensively used and studied in machine learning*

CP Tensor Completion Gradient and Hessian

- The gradient of the tensor completion objective function is sparsified according to the set of observed entries
  - *Lets restrict attention to optimizing for the ith row of the first factor matrix, define Ω_i so that $(j,k) \in \Omega_i$ if s $(i,j,k) \in \Omega$, then*

$$\phi(\boldsymbol{u}_i) = \sum_{(j,k)\in\Omega_i} (t_{ijk} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j, \boldsymbol{w}_k \rangle)^2 + \lambda \|\boldsymbol{u}_i\|_2^2 \quad \textit{where} \quad \langle \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \rangle = \sum_r x_r y_r z_r$$

  - *Consider the derivative with respect to the ith row of the first factor matrix*

$$\nabla\phi(\boldsymbol{u}_i) = \frac{d\phi}{d\boldsymbol{u}_i}(\boldsymbol{u}_i) = 2 \sum_{(j,k)\in\Omega_i} (\boldsymbol{v}_j * \boldsymbol{w}_k)(\langle \boldsymbol{u}_i, \boldsymbol{v}_j, \boldsymbol{w}_k \rangle - t_{ijk}) + 2\lambda \boldsymbol{u}_i$$

- ALS for tensor decomposition solves quadratic optimization problem for each row of each factor matrix, in the completion case, Newton's method on these subproblems yields different Hessians
  - *The Hessian $\boldsymbol{H}_i^{(\phi)}$ depends on the set of entries $\Omega_i = \{(j,k) : \exists (i,j,k) \in \Omega\}$,*

$$\boldsymbol{H}_i^{(\phi)} = \frac{d\phi^2}{d\boldsymbol{u}_i d\boldsymbol{u}_i}(\boldsymbol{u}_i) = \sum_{(j,k)\in\Omega_i} (\boldsymbol{v}_j * \boldsymbol{w}_k)(\boldsymbol{v}_j * \boldsymbol{w}_k)^T + 2\lambda \boldsymbol{I}$$

Methods for CP Tensor Completion

- ALS for tensor completion with CP decomposition incurs additional cost
  - *For each $(i, j, k) \in \Omega$, need to accumulate $(\boldsymbol{v}_j * \boldsymbol{w}_k)(\boldsymbol{v}_j * \boldsymbol{w}_k)^T$ to $\boldsymbol{H}_i^{(\phi)}$*
  - *While the n outer products can be amortized with cost $O(nR^2)$, no easy way to do so for their partial sums, leading to cost $O(|\Omega|R^2)$*
- Alternative methods for tensor completion include coordinate descent and stochastic gradient descent
  - *Stochastic gradient descent (SGD) would compute subgradients for each (i, j, k) which are summands in the sum over Ω_i in $\nabla\phi(\boldsymbol{u}_i)$*
  - *SGD can be implemented efficiently, by computing a sum over a random set of subgradients at a time, via subsampling of Ω*
  - *Coordinate descent optimizes an entry of each factor matrix at a time*
  - *Variants of coordinate descent select different orderings of entries to optimize, e.g., alternating among columns of factor matrices then factor matrices or vice versa*

Coordinate Descent for CP Tensor Completion

- Coordinate descent avoids the need to solve linear systems of equations
  - *The coordinate-wise objective function is*

$$\psi(u_{ir}) = \sum_{(j,k)\in\Omega_i} (\rho_{ijk}^{(r)} - u_{ir}v_{jr}w_{kr})^2 + \lambda u_{ir}^2 \text{ where } \rho_{ijk}^{(r)} = t_{ijk} - \langle \boldsymbol{u}_i, \boldsymbol{v}_j, \boldsymbol{w}_k \rangle + u_{ir}v_{ir}w_{kr}$$

  above $\rho_{ijk}^{(r)}$ is equal to an entry of the residual tensor with the rth rank-one component of the CP decomposition excluded
  - *Taking its derivative, we obtain*

$$\psi'(u_{ir}) = -2 \sum_{(j,k)\in\Omega_i} v_{jr}w_{kr}(\rho_{ijk}^{(r)} - u_{ir}v_{jr}w_{kr}) + 2\lambda u_{ir}$$

  - *Setting this derivative to zero, we can solve for u_{ir}*

$$u_{ir}^{(new)} = \frac{\sum_{(j,k)\in\Omega_i} v_{jr}w_{kr}\rho_{ijk}^{(r)}}{\lambda + \sum_{(j,k)\in\Omega_i} v_{jr}^2 w_{kr}^2}$$

  - *This can be implemented efficiently by keeping track of a residual tensor and obtaining $\rho_{ijk}^{(r)}$ as a modification thereof when working on the rth column*

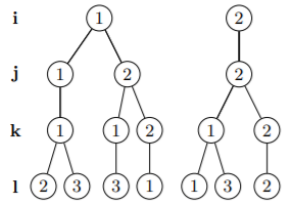
Sparse Tensor Contractions

- Tensor completion and sparse tensor decomposition require operations on sparse tensors
  - *In many publicly available sparse tensor datasets, the density is extremely low, e.g., 10^{-7}, i.e., there can be $O(n)$ nonzeros in interesting $n \times n \times n$ tensors*
  - *For both decomposition and completion, tensor sparsity does not generally imply sparsity of CP or Tucker factors, and these are typically assumed to be dense*
- Sparse tensor contractions often correspond to products of *hypersparse* matrices, i.e., matrices with mostly zero rows
  - *Consider TTM with a $n \times n \times n$ tensor $\mathcal{T}$ containing $O(n)$ nonzeros, $\boldsymbol{T}_{(1)}^T \boldsymbol{M}$, the matrix $\boldsymbol{T}_{(1)}^T$ has $O(n)$ nonzeros, but n^2 rows, while $\boldsymbol{T}_{(1)}^T \boldsymbol{M}$ has $O(n)$ dense rows and all other $O(n^2)$ rows are zero*
  - *To reduce sparse tensor contractions to sparse matrix multiplication kernels, need support for hypersparse matrix formats (e.g., compressed sparse-row (CSR) format would require $\Theta(n^2)$ storage for $\boldsymbol{T}_{(1)}$) and ideally specialized formats for matrices such as $\boldsymbol{T}_{(1)}^T \boldsymbol{M}$ (e.g., dense matrix consisting of nonzero rows and vector of row indices)*

Sparse Tensor Formats

- The overhead of transposition, and non-standard nature of the arising sparse matrix products, motivates sparse data structures for tensors that are suitable for tensor contractions of interest
  - *Particularly important, especially for tensor decomposition, are MTTKRP (suffices to CP ALS) and TTMc (suffices for HOOI)*
  - *TTM is also prevalent, but is a less attractive primitive in the sparse case than MTTKRP and TTMc, as these yield dense, low-order outputs, while the output of TTM can be sparse and larger than the starting tensor*
- The *compressed sparse fiber (CSF)* format provides an effective representation for sparse tensors
  - *CSF can be visualized as a tree (diagram taken from original CSF paper, by Shaden Smith and George Karpis, IA^3, 2015)*

i	j	k	l
1	1	1	2
1	1	1	3
1	2	1	3
1	2	2	1
2	2	1	1
2	2	1	3
2	2	2	2



Operations in Compressed Format

- CSF permits efficient execution of important sparse tensor kernels
  - Analogous to CSR format, which enables efficient implementation of the sparse matrix vector product
  - where `row[i]` stores a list of column indices and nonzeros in the ith row of $\boldsymbol{A}$

```
for i in range(n):
  for (a_ij,j) in row[i]:
    y[i] += a_ij * x[j]
```

  - In CSF format, a multilinear function evaluation $\boldsymbol{f}^{(\mathcal{T})}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{T}_{(1)}(\boldsymbol{x} \odot \boldsymbol{y})$ can be implemented as

```
for (i,T_i) in T_CSF:
  for (j,T_ij) in T_i:
    for (k,t_ijk) in T_ij:
      z[i] += t_ijk * x[j] * y[k]
```

MTTKRP in Compressed Format

- MTTKRP and CSF pose additional implementation opportunities and challenges
  - MTTKRP $u_{ir} = \sum_{j,k} t_{ijk} v_{jr} w_{kr}$ can be implemented by adding a loop over r to our code for $f^{(\mathcal{T})}$, but would then require $3mr$ operations if m is the number of nonzeros in $\mathcal{T}$, can reduce to $2mr$ by amortization

```
for (i,T_i) in T_CSF:
  for (j,T_ij) in T_i:
    for r in range(R):
      f_ij = 0
      for (k,t_ijk) in T_ij:
        f_ij += t_ijk * w[k,r]
      u[i,r] = f_ij * v[j,r]
```

  - However, this amortization is harder (requires storage or iteration overheads) if the index i is a leaf node in the CSF tree
  - Similar challenges in achieving good reuse and obtaining good arithmetic intensity arise in implementation of other kernels, such as TTMc

All-at-once Contraction

- When working with sparse tensors, it is often more efficient to contract multiple operands in an all-at-once fashion
  - *Given chain of matrix products $\boldsymbol{ABC}\cdots$, dimension of overall iteration space scales with number of matrices, but by contracting pairwise, obtain cubic cost in matrix dimension with linear dependence on number of matrices*
  - *A case when such pairwise contraction is not a good idea, is the sampled dense-dense matrix-multiplication (SDDMM),*

$$c_{ij} = \sum_{r}^{R} a_{ij} u_{ir} v_{jr} \Leftrightarrow \boldsymbol{C} = \boldsymbol{A} * (\boldsymbol{U}\boldsymbol{V}^T)$$

    where $\boldsymbol{A}$ is sparse with m nonzeros, while $\boldsymbol{U}$ and $\boldsymbol{V}$ are dense
  - *Since the sparsity pattern of $\boldsymbol{C}$ is the same as of $\boldsymbol{A}$, suffices to iterate over nonzeros of $\boldsymbol{A}$ and multiply each by inner product of a row of $\boldsymbol{U}$ and a row of $\boldsymbol{V}$, with cost $O(mR)$*
  - *Pairwise contraction is inefficient, contracting first $\boldsymbol{A}$ with $\boldsymbol{U}$ would yield a third-order intermediate, while contracting $\boldsymbol{U}$ with $\boldsymbol{V}^T$ would have cost $O(n^2R)$*
  - *Generalizing SDDMM to higher order gives the tensor-times tensor-product (TTTP), an application of which is computing the residual in tensor completion*

Constrained Tensor Decomposition

- Many applications of tensor decomposition in data science, feature additional structure, which can be enforced by constraints
  - *A basic and common constraint is nonnegativity of factor matrices, which often makes sense when working with a tensor that is nonnegative (e.g., count data)*
  - *Most of the methods we've discussed can be generalized to handle nonnegativity, e.g., one could perform ALS by solving each subproblem subject to nonnegativity constraints*
  - *Another common constraint is factor matrix orthogonality, which can be incorporated similarly into subproblems*
  - *For symmetric tensors, repeating factors are often desired, which can be formulated via constraints or by using an appropriate method (two good alternatives are ALS with subiterations to converge updates to repeated factors, or Gauss-Newton, which automatically preserves repeating factors when working with a symmetric tensor)*

Nonnegative Tensor Factorization

- *Nonnegative tensor factorization (NTF)*, such as CP decomposition with $\mathcal{T} \geqslant 0$ and $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W} \geqslant 0$ are widespread and a few classes of algorithms have been developed
  - *Optimization for one of $\boldsymbol{U}$, $\boldsymbol{V}$, or $\boldsymbol{W}$ (while the other two are fixed) is a convex optimization problem*
  - *Many methods based on alternating optimization/updates in the style of ALS*
  - *A basic approach is to 'clip' result of ALS step so that each factor matrix is nonnegative after update*
  - *Block coordinate descent (BCD) methods update on or more columns of $\boldsymbol{U}$, $\boldsymbol{V}$, or $\boldsymbol{W}$ based on a coordinate-descent-like update rule*
  - *Proximal gradient methods are multicolumn BCD methods, which approximately solve each subproblem by minimizing a constrained objective derived based on a proximally projected gradient*
  - *All-at-once methods that update all factor matrices, such as Gauss-Newton with an augmented Lagrangian objective function (sequential quadratic programming)*

Nonnegative Matrix Factorization

- NTF algorithms with alternating updates have a close correspondence with alternating update algorithms for *Nonnegative matrix factorization (NMF)*
  - *The rank-r NMF problem is to find, given matrix $\boldsymbol{A} \in \mathbb{R}_+^{n \times n}$, the minimizer $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}_+^{n \times r}$ to*

$$f^{(\boldsymbol{A})}(\boldsymbol{U}, \boldsymbol{V}) = \|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^T\|_F$$

  - *Solutions to NMF are not easy to compute directly as for the unconstrained low-rank matrix factorization case*
  - *Methods often minimize $\phi_{\boldsymbol{V}}^{(\boldsymbol{A})}(\boldsymbol{U}) = f^{(\boldsymbol{A})}(\boldsymbol{U}, \boldsymbol{V})$ and $\psi_{\boldsymbol{U}}^{(\boldsymbol{A})}(\boldsymbol{V}) = f^{(\boldsymbol{A})}(\boldsymbol{U}, \boldsymbol{V})$ in an alternating fashion via block coordinate descent*
  - *Alternating optimization problems for NTF are essentially the same as in NMF, for*

$$g^{(\boldsymbol{\mathcal{T}})}(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) = \|\boldsymbol{\mathcal{T}} - [\![\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}]\!]\|_F$$

  minimizing $\phi_{\boldsymbol{V},\boldsymbol{W}}^{(\boldsymbol{\mathcal{T}})}(\boldsymbol{U}) = g^{(\boldsymbol{\mathcal{T}})}(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W})$ is the same as the NMF subproblem $\phi_{\boldsymbol{V}\odot\boldsymbol{W}}^{(\boldsymbol{T}_{(1)})}(\boldsymbol{U}) = f^{(\boldsymbol{T}_{(1)})}(\boldsymbol{U}, \boldsymbol{V} \odot \boldsymbol{W})$

Coordinate Descent for NMF and NTF

- Coordinate descent gives optimal closed-form updates for variables in NMF and NTF
  - *We can write an optimization subproblem for a single column $\boldsymbol{u}_i$ as minimizing*

$$\phi_i^{(\boldsymbol{A})}(\boldsymbol{u}_i) = \|\boldsymbol{A} - \sum_{r=1}^{R} \boldsymbol{u}_r \boldsymbol{v}_r^T\|_2 \quad \textit{s.t.} \quad \boldsymbol{u}_i \geqslant 0$$

$$\boldsymbol{u}_i^{\textit{new}} = |\boldsymbol{u}_i + \frac{\boldsymbol{A}\boldsymbol{v}_i - \boldsymbol{U}\boldsymbol{V}^T\boldsymbol{v}_i}{\boldsymbol{v}_i^T\boldsymbol{v}_i}|_+$$

  where $\boldsymbol{y} = |\boldsymbol{x}|_+$ gives $y_i = x_i$ if $x_i > 0$ and $y_i = 0$ otherwise
  - *Given $\boldsymbol{\rho}^{(i)} = \boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^T + \boldsymbol{u}_i\boldsymbol{v}_i^T = \boldsymbol{A} - \sum_{j \neq i} \boldsymbol{u}_j\boldsymbol{v}_j^T$, if columns of $\boldsymbol{V}$ are normalized, we can alternatively write the update as*

$$\boldsymbol{u}_i^{\textit{new}} = |\frac{\boldsymbol{\rho}^{(i)}\boldsymbol{v}_i}{\boldsymbol{v}_i^T\boldsymbol{v}_i}|_+$$

Generalized Tensor Decomposition

- Aside from addition of constraints, the objective function may be modified by using different elementwise loss functions
  - *The standard loss function is $(x-m)^2$ where x is an element of the tensor and m is its approximation via CP*
  - *For count data, the Poisson loss function $m - x\log(m)$ may be more appropriate, and typically comes along with nonnegativity constraints*
  - *Other distributions and loss functions of interest include Gamma, Rayleigh, Bernoulli, and NegBinom (see D. Hong, T. Kolda, J. Duersch SIAM Review 2020)*
- Some loss function admit ALS-like algorithms, while others may require gradient-based optimization
  - *Can compute (sub-)gradients given any loss function, by differentiating as necessary*
  - *For Poisson, like for the standard loss function, ALS subproblems may be solved explicitly, allowing more robust convergence*