Accelerating Nonnegative-X by extrapolation

where X $\in$ {Least Square, Matrix Factorization, Tensor Factorization}

Andersen Ang

Mathématique et recherche opérationnelle
UMONS, Belgium

Email: manshun.ang@umons.ac.be

Homepage: angms.science

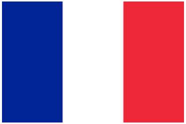
April 1, 2019

Department of Electrical and Electronic Engineering
University of Hong Kong

Quel est le sujet de cette présentation

- Algorithmes numérique pour les problèmes d'optimisation non linéaire avec contraintes non-négativité

- Algorithmes de Factorisation Non-Négative de Matrices/Tenseurs



Niet-negatieve matrixfactorisatie

De algemene formulering van een matrixfactorisatie is:

$$\mathbf{X} = \mathbf{W}\mathbf{H},$$

waarbij de dimensies als volgt zijn: $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{W} \in \mathbb{R}^{m \times r}$, en $\mathbf{H} \in \mathbb{R}^{r \times n}$.

Voor de op voorhand zelf te specificeren dimensie r geldt:

$$0 < r < \min\{n, m\}.$$

Happy April Fool's day !!!!!!!!!!!!!

1. Introduction - Non-negative Matrix Factorization
2. Computing NMF
   - Variations on BCD
   - A-HALS
   - Matrix-wise Projected Gradient Update and the Multiplicative update
3. Find $(\mathbf{W}, \mathbf{H})$ numerically fast : acceleration via extrapolation
   - Recall : acceleration in single variable problem
   - Accelerating NMF algorithms using extrapolation
4. Computing NTF
5. Computing NNLS

Outline

1. Introduction - Non-negative Matrix Factorization
2. Computing NMF
   - Variations on BCD
   - A-HALS
   - Matrix-wise Projected Gradient Update and the Multiplicative update
3. Find $(\mathbf{W}, \mathbf{H})$ numerically fast : acceleration via extrapolation
   - Recall : acceleration in single variable problem
   - Accelerating NMF algorithms using extrapolation
4. Computing NTF
5. Computing NNLS

Non-negative Matrix Factorization (NMF)

Given :

- A matrix $\mathbf{X} \in \mathbb{R}_{+}^{m \times n}$.
- A positive integer $r \in \mathbb{N}$.

Find :

- Matrices $\mathbf{W} \in \mathbb{R}_{+}^{m \times r}, \mathbf{H} \in \mathbb{R}_{+}^{r \times n}$ such that $\mathbf{X} = \mathbf{W}\mathbf{H}$.

Non-negative Matrix Factorization (NMF)

Given :

- A matrix $\mathbf{X} \in \mathbb{R}_{+}^{m \times n}$.
- A positive integer $r \in \mathbb{N}$.

Find :

- Matrices $\mathbf{W} \in \mathbb{R}_{+}^{m \times r}, \mathbf{H} \in \mathbb{R}_{+}^{r \times n}$ such that $\mathbf{X} = \mathbf{WH}$.
- Everything is non-negative.

Notation : we use $\mathbf{WH}$ instead of $\mathbf{WH}^\top$.

Exact NMF: given $(\mathbf{X}, r)$, find $(\mathbf{W}, \mathbf{H})$ s.t. $\mathbf{X} = \mathbf{WH}$.

It is NP-hard (Vavasis, 2007).

Vavasis, "On the complexity of nonnegative matrix factorization", SIAM J. Optim.

Exact and approximate NMF

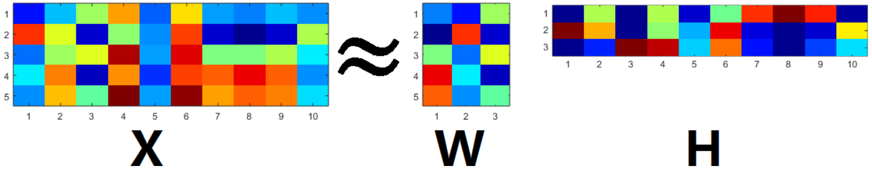
Exact NMF: given $(\mathbf{X}, r)$, find $(\mathbf{W}, \mathbf{H})$ s.t. $\mathbf{X} = \mathbf{WH}$.

It is NP-hard (Vavasis, 2007).

Vavasis, "On the complexity of nonnegative matrix factorization", SIAM J. Optim.

This talk : (Low-rank) *approximate NMF*

$$\mathbf{X} \approx \mathbf{WH}, \; 1 \leq r \leq \min\{m, n\}.$$



Compute $(\mathbf{W}, \mathbf{H})$ numerically

We solve

$$[\mathbf{W},\ \mathbf{H}] = \underset{\mathbf{W}\geq\mathbf{0},\mathbf{H}\geq\mathbf{0}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{WH}\|_F.$$

- Minimizing the distance between $\mathbf{X}$ and the approximator $\mathbf{WH}$ in F-norm†.
- $\geq$ is element-wise (not positive semi-definite).
- Such minimization problem is
  - Bi-variate : two variables
  - Non-convex but block-convex (strongly convex/strictly convex)
  - Non-smooth : on the boundary between $\mathbb{R}_+$ and $\mathbb{R}_-$
  - Ill-posed and NP-hard (Vavasis, 2007)

†This talk does not consider other distance functions.

NMF downgrade = Non-negative Least Squares

Given $(\mathbf{x}_j \in \mathbb{R}_+^m, \mathbf{W} \in \mathbb{R}_+^{m\times r})$, find $\mathbf{h}_j \in \mathbb{R}_+^r$ s.t. $\mathbf{x}_j \approx \mathbf{W}\mathbf{h}_j$ via solving

$$\mathbf{h}_j = \underset{\mathbf{h}\geq\mathbf{0}}{\operatorname{argmin}} \|\mathbf{x}_j - \mathbf{W}\mathbf{h}_j\|_2.$$

- $\geq$ is element-wise.
- Such minimization problem is
  - Single variable
  - Non-smooth : on the boundary between $\mathbb{R}_+$ and $\mathbb{R}_-$
  - No analytic solution
  - Convex : depends on $\mathbf{W}$ $\to$ strongly convex / strictly convex
  - Global minimizer "obtainable"
- In more "standard" notation

$$\text{(NNLS)}\quad \mathbf{x} = \underset{\mathbf{x}\geq\mathbf{0}}{\operatorname{argmin}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$$

NMF upgrade = Non-negative Tensor Factorization

Given $(\mathbf{X} \in \mathbb{R}_+^{I \times J \times K}, r \in \mathbb{N})$
Find $\mathbf{U} \in \mathbb{R}_+^{I \times r}, \mathbf{V} \in \mathbb{R}_+^{J \times r}$ and $\mathbf{W} \in \mathbb{R}_+^{K \times r}$ s.t.
$\mathbf{X} \approx \mathbf{U} * \mathbf{V} * \mathbf{W}$ via solving

$$\mathbf{X} = \underset{\{\mathbf{U},\mathbf{V},\mathbf{W}\} \geq \mathbf{0}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{U} * \mathbf{V} * \mathbf{W}\|_F.$$

- $\geq$ is element-wise.
- Such minimization problem is
  - Three variables
  - Non-smooth : on the boundary between $\mathbb{R}_+$ and $\mathbb{R}_-$
  - Non-convex but block-convex (strongly convex/strictly convex)
  - Global minimizer "obtainable" – unique solution under some conditions on I, J, K, r

†This talk does not consider other tensor norm.

The scope of this talk : computation of NNLS, NMF, NTF

Keywords : Numerical optimization, Continuous optimization, Algorithm, Convergence, Non-convex, Nesterov's Acceleration, Extrapolation

Non-keywords : Sparsity, Regularization, Applications, Non-negative rank, Extended Formulations, Separability, NP-Hardness

Focus : single-machine, serial, deterministic algorithm

Non-focus : multi-machine, parallel, distributed, stochastic algorithm

5 slides on why NMF

For non-NMF people : why NMF ?

- Interpretability

  NMF beats similar tools (PCA, SVD, ICA) due to the interpretability on non-negative data.

- Model correctness

  NMF can find ground truth (under certain conditions).

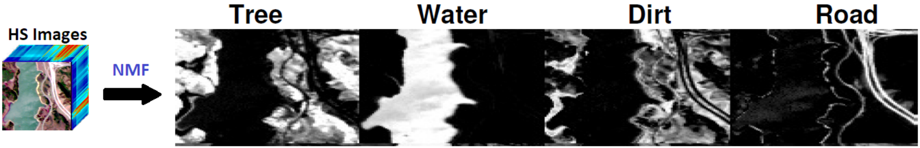
- Mathematical curiosity

  NMF is related to some serious problems in mathematics.

- ~~My boss tell me to do it.~~

Why NMF - Hyper-spectral image example

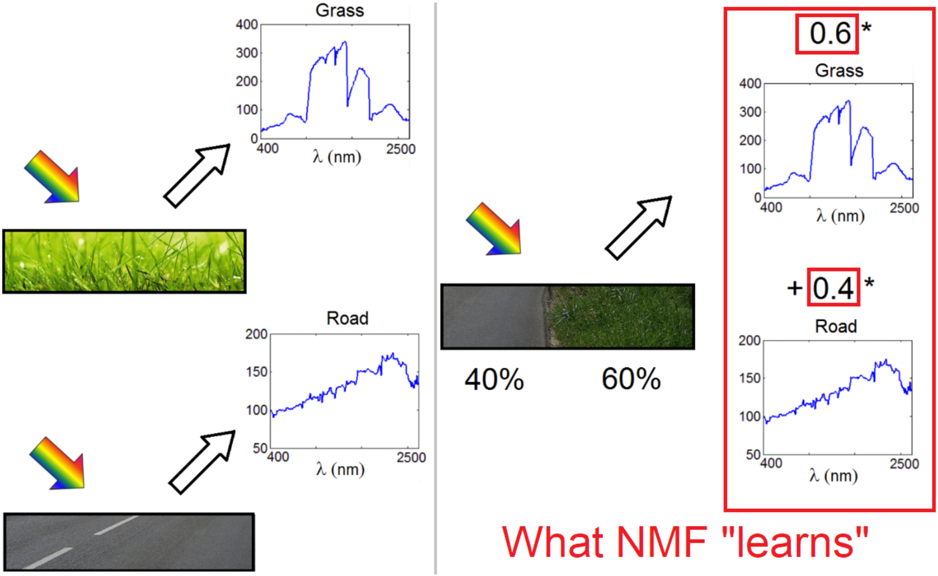
NMF gives good *unsupervised* image segmentation[1]



Hyper-spectral image decomposition. Figure from Zhu, F. et al., "Spectral unmixing via data-guided sparsity." IEEE Trans. Image Processing, 2014

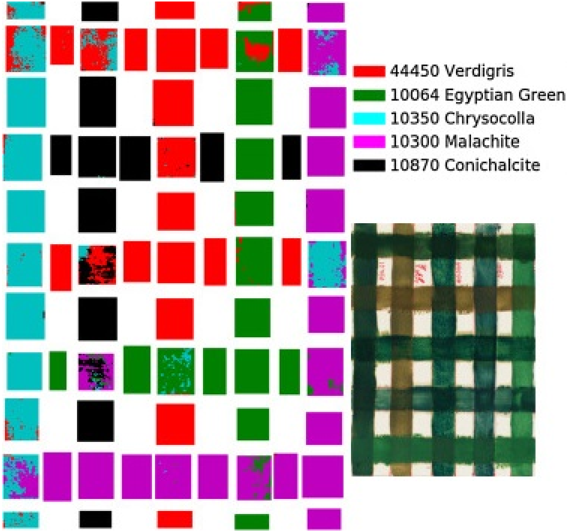
Comment est-ce possible ?!

[1] Modern fancy name : "super resolution"



Hyper-spectral imaging. Figure modified from the slide of Nicolas Gillis.

Why NMF - art work preservation example



Art work preservation. Figure from Grabowski, Bartosz, et al. “Automatic pigment identification from hyperspectral data.” J. Cultural Heritage 31 (2018): 1-12.

Why NMF - other examples

Application side

- Spectral unmixing in analytical chemistry (one of the earliest work)
- Representation learning on human face (the work that popularizes NMF)
- Topic modeling in text mining
- Probability distribution application on identification of Hidden Markov Model
- Bioinformatics : gene expression
- Time-frequency matrix decompositions for neuroinformatics
- (Non-negative) Blind source separation
- (Non-negative) Data compression
- Speech denoising
- Recommender system
- Face recognition
- Video summarization
- Radio
- Audio
- Forensics
- Art work conservation (identify true color used in painting)
- Medical imaging – image processing on small object
- Mid-infrared astronomy – image processing on large object
- Telling whether a banana or a fish is healthy

Theoretical numerical side

- A test-box for generic optimization programs : NMF is a constrained non-convex (but biconvex) problem
- Robustness analysis of algorithm
- Tensor
- Sparsity

Analytical side

- Non-negative rank rank^+ := smallest r such that
$$\mathbf{X} = \sum_{i=1}^{r} \mathbf{X}_i, \quad : \; \mathbf{X}_i \text{ rank-1 and non-negative.}$$
How to find / estimate / bound rank^+, e.g. $\text{rank}_{\text{psd}}(\mathbf{X}) \leq \text{rank}^+(\mathbf{X})$.
- Extended formulations and combinatorics
- Log-rank Conjecture of communication system
- 3-SAT, Exponential time hypothesis, $\mathbf{P} \neq \mathbf{NP}$

Outline

1. Introduction - Non-negative Matrix Factorization
2. Computing NMF
   - Variations on BCD
   - A-HALS
   - Matrix-wise Projected Gradient Update and the Multiplicative update
3. Find $(\mathbf{W}, \mathbf{H})$ numerically fast : acceleration via extrapolation
   - Recall : acceleration in single variable problem
   - Accelerating NMF algorithms using extrapolation
4. Computing NTF
5. Computing NNLS

Problem $(\mathcal{P})$: given $(\mathbf{X}, r)$, solve

$$[\mathbf{W},\ \mathbf{H}] = \underset{\mathbf{W}\geq\mathbf{0},\mathbf{H}\geq\mathbf{0}}{\operatorname{argmin}}\ \Phi(\mathbf{W},\mathbf{H}) = \|\mathbf{X}-\mathbf{WH}\|_F.$$

- Equivalent objective function : $\frac{1}{2}\|\mathbf{X}-\mathbf{WH}\|_F^2$.
- Simplify notation : hide some $\geq \mathbf{0}, \frac{1}{2}, F$ and write

$$\min_{\mathbf{W},\mathbf{H}} \Phi(\mathbf{W},\mathbf{H}) = \|\mathbf{X}-\mathbf{WH}\|^2.$$

Standard framework to solve $(\mathcal{P})$

$(\mathcal{P})$: $\min\limits_{\mathbf{W},\mathbf{H}} \Phi(\mathbf{W},\mathbf{H}) = \|\mathbf{X} - \mathbf{WH}\|^2$.

Approach : BCD (Block Coordinate Descent)[2]

Algorithm BCD framework for $\mathcal{P}$

Input: $\mathbf{X} \in \mathbb{R}_+^{m\times n}$, $r \in \mathbb{N}$, an initialization $\mathbf{W} \in \mathbb{R}_+^{m\times r}$, $\mathbf{H} \in \mathbb{R}_+^{r\times n}$

Output: $\mathbf{W}$ and $\mathbf{H}$

1: **for** $k = 1, 2, \ldots$ **do**

2: Update$[\mathbf{W}]$: do something with $\Phi, \mathbf{X}, \mathbf{W}, \mathbf{H}$.

3: Update$[\mathbf{H}]$: do something with $\Phi, \mathbf{X}, \mathbf{W}, \mathbf{H}$.

4: **end for**

[2] Other names : Gauss-Seidel iteration, alternating minimization (for 2 blocks)

Standard framework to solve ($\mathcal{P}$)

$(\mathcal{P})$: $\min\limits_{\mathbf{W},\mathbf{H}} \Phi(\mathbf{W},\mathbf{H}) = \|\mathbf{X} - \mathbf{W}\mathbf{H}\|^2$.

Approach : BCD (Block Coordinate Descent)[2]

Algorithm BCD framework for $\mathcal{P}$

Input: $\mathbf{X} \in \mathbb{R}_+^{m\times n}$, $r \in \mathbb{N}$, an initialization $\mathbf{W} \in \mathbb{R}_+^{m\times r}$, $\mathbf{H} \in \mathbb{R}_+^{r\times n}$
Output: $\mathbf{W}$ and $\mathbf{H}$

1: **for** $k = 1, 2, \dots$ **do**
2: Update$[\mathbf{W}]$: do something with $\Phi, \mathbf{X}, \mathbf{W}, \mathbf{H}$.
3: Update$[\mathbf{H}]$: do something with $\Phi, \mathbf{X}, \mathbf{W}, \mathbf{H}$.
4: **end for**

The goal of "do something" is to achieve

$$\Phi(\mathbf{W}^{k+1}, \mathbf{H}^{k+1}) \leq \Phi(\mathbf{W}^{k+1}, \mathbf{H}^{k}) \leq \Phi(\mathbf{W}^{k}, \mathbf{H}^{k}).$$

(Actually non-increasing condition is not enough, need sufficient decrease condition)

[2] Other names : Gauss-Seidel iteration, alternating minimization (for 2 blocks)

Example 1 : alternating minimization

Algorithm BCD framework for $\mathcal{P}$

Input: $\mathbf{X} \in \mathbb{R}_+^{m \times n}$, $r \in \mathbb{N}$, an initialization $\mathbf{W} \in \mathbb{R}_+^{m \times r}$, $\mathbf{H} \in \mathbb{R}_+^{r \times n}$
Output: $\mathbf{W}$ and $\mathbf{H}$

for $k = 1, 2, \dots$ **do**
  Update$[\mathbf{W}]$ as $\mathbf{W} \leftarrow \underset{\mathbf{W} \geq 0}{\operatorname{argmin}} \Phi(\mathbf{W}) = \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$.
  Update$[\mathbf{H}]$ as $\mathbf{H} \leftarrow \underset{\mathbf{H} \geq 0}{\operatorname{argmin}} \Phi(\mathbf{H}) = \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$.
end for

Example 1 : alternating minimization

Algorithm BCD framework for $\mathcal{P}$

Input: $\mathbf{X} \in \mathbb{R}_+^{m \times n}$, $r \in \mathbb{N}$, an initialization $\mathbf{W} \in \mathbb{R}_+^{m \times r}$, $\mathbf{H} \in \mathbb{R}_+^{r \times n}$
Output: $\mathbf{W}$ and $\mathbf{H}$

1: **for** $k = 1, 2, \ldots$ **do**
2: Update$[\mathbf{W}]$ as $\mathbf{W} \leftarrow \underset{\mathbf{W} \geq 0}{\operatorname{argmin}} \Phi(\mathbf{W}) = \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$.
3: Update$[\mathbf{H}]$ as $\mathbf{H} \leftarrow \underset{\mathbf{H} \geq 0}{\operatorname{argmin}} \Phi(\mathbf{H}) = \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$.
4: **end for**

Symmetry : $\|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2 = \|\mathbf{X}^\top - \mathbf{H}^\top\mathbf{W}^\top\|_F^2$,
$\rightarrow$ we can use the same scheme on both variables.
We can focus on one variable, says $\mathbf{H}$ (or $\mathbf{W}$).

If asymmetric regularization exists on $\mathbf{W}$ (or $\mathbf{H}$) : we have to handle them separately.

Example 2: alternating gradient descent

Algorithm BCD framework for $\mathcal{P}$

Input: $\mathbf{X} \in \mathbb{R}_+^{m \times n}$, $r \in \mathbb{N}$, an initialization $\mathbf{W} \in \mathbb{R}_+^{m \times r}$, $\mathbf{H} \in \mathbb{R}_+^{r \times n}$
Output: $\mathbf{W}$ and $\mathbf{H}$

1: **for** $k = 1, 2, \ldots$ **do**

2: Update$[\mathbf{W}]$ as

$$\mathbf{W} \leftarrow \underset{\mathbf{W} \geq 0}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{U}\mathbf{H}\|_F^2 + \langle \mathbf{W} - \mathbf{U}, \nabla \Phi(\mathbf{U}) \rangle + \frac{1}{2t} \|\mathbf{U} - \mathbf{W}\|^2.$$

3: Update$[\mathbf{H}]$ as

$$\mathbf{H} \leftarrow \underset{\mathbf{H} \geq 0}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{W}\mathbf{V}\|_F^2 + \langle \mathbf{H} - \mathbf{V}, \nabla \Phi(\mathbf{V}) \rangle + \frac{1}{2t} \|\mathbf{V} - \mathbf{H}\|^2.$$

4: **end for**

Local quadratic model : gradient descent minimizes the local quadratic model of the original objective function

Variations on BCD

$$\text{Update}[\mathbf{H}] : \ \mathbf{H} \leftarrow \underset{\mathbf{H} \geq 0}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$$

1. Block partitions : on how coordinate is being defined[†].
   Now : coordinate is $\mathbf{H}$ (matrix) or $\mathbf{H}(i,:)$ (vector).

Variations on BCD

$$\text{Update}[\mathbf{H}]: \ \mathbf{H} \leftarrow \underset{\mathbf{H}\geq 0}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{WH}\|_F^2$$

1. Block partitions : on how coordinate is being defined†.
   Now : coordinate is $\mathbf{H}$ (matrix) or $\mathbf{H}(i,:)$ (vector).
2. Index selection (indexing) : on how coordinate is being selected$^{\#}$.
   Now : cyclic indexing and A-HALS.

Variations on BCD

$$\text{Update}[\mathbf{H}] : \ \mathbf{H} \leftarrow \underset{\mathbf{H} \geq 0}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{WH}\|_F^2$$

1. Block partitions : on how coordinate is being defined†.
   Now : coordinate is $\mathbf{H}$ (matrix) or $\mathbf{H}(i, :)$ (vector).
2. Index selection (indexing) : on how coordinate is being selected#.
   Now : cyclic indexing and A-HALS.
3. Update scheme : on how coordinate is being updated#.
   Now : "exact" coordinate minimization using 1st order method (e.g. gradient descent).
   Exact = working on the original original objective function, no modification.
   Inexact = working on modified objective function. e.g. consider relaxation.

Variations on BCD

$$\text{Update}[\mathbf{H}] : \ \mathbf{H} \leftarrow \underset{\mathbf{H} \geq 0}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{WH}\|_F^2$$

1. Block partitions : on how coordinate is being defined†.
   Now : coordinate is $\mathbf{H}$ (matrix) or $\mathbf{H}(i, :)$ (vector).
2. Index selection (indexing) : on how coordinate is being selected#.
   Now : cyclic indexing and A-HALS.
3. Update scheme : on how coordinate is being updated#.
   Now : "exact" coordinate minimization using 1st order method (e.g. gradient descent).
   Exact = working on the original original objective function, no modification.
   Inexact = working on modified objective function. e.g. consider relaxation.
4. Other variants

† Kim-He-Park 2014,"Algo. for nonnegative matrix and tensor factorizations: a unified view based on block coordinate descent framework" J. Global Optimization.

#Shi-Tu-Xu-Yin 2017,"A primer on coordinate descent algorithms." arXiv:1610.00040

The idea of HALS and A-HALS

Says coordinates are vectors (col. of $\mathbf{W}$ and row of $\mathbf{H}$), we have

$$\Phi(\mathbf{w}_i, \mathbf{h}_i) = \|\mathbf{w}_i\|_2^2 \|\mathbf{h}_i\|_2^2 - 2\mathrm{tr}\left\langle \mathbf{X}_i, \mathbf{w}_i \mathbf{h}_i \right\rangle + c.$$

Says coordinates are vectors (col. of $\mathbf{W}$ and row of $\mathbf{H}$), we have

$$\Phi(\mathbf{w}_i, \mathbf{h}_i) = \|\mathbf{w}_i\|_2^2 \|\mathbf{h}_i\|_2^2 - 2\mathrm{tr}\left\langle \mathbf{X}_i, \mathbf{w}_i \mathbf{h}_i \right\rangle + c.$$

Alternating minimization using cyclic indexing

Other name : BCD with $r = 2$ with cyclic component selection

Domain name in NMF : HALS (Hierarchical alternating least squares[†])

Update order : $\mathbf{w}_1 \rightarrow \mathbf{h}_1 \rightarrow \mathbf{w}_2 \rightarrow \mathbf{h}_2 \rightarrow \mathbf{w}_3 \rightarrow \mathbf{h}_3 \rightarrow \ldots$

The idea of HALS and A-HALS

Says coordinates are vectors (col. of $\mathbf{W}$ and row of $\mathbf{H}$), we have

$$\Phi(\mathbf{w}_i, \mathbf{h}_i) = \|\mathbf{w}_i\|_2^2 \|\mathbf{h}_i\|_2^2 - 2\mathrm{tr}\left\langle \mathbf{X}_i, \mathbf{w}_i \mathbf{h}_i \right\rangle + c.$$

Alternating minimization using cyclic indexing

Other name : BCD with $r = 2$ with cyclic component selection

Domain name in NMF : HALS (Hierarchical alternating least squares†)

Update order : $\mathbf{w}_1 \to \mathbf{h}_1 \to \mathbf{w}_2 \to \mathbf{h}_2 \to \mathbf{w}_3 \to \mathbf{h}_3 \to \ldots$

A-HALS# (Accelerated-HALS)

A special kinds of cyclic coordinate selection

$$\text{Update order : } \underbrace{\mathbf{w}_1 \to \mathbf{w}_2 \to \cdots \to \mathbf{w}_r}_{\textbf{several times!!}} \to \underbrace{\mathbf{h}_1 \to \mathbf{h}_2 \to \cdots \to \mathbf{h}_r}_{\textbf{several times!!}} \to \ldots$$

† Cichocki-Zdunke-Amari 2007, "Hierarchical ALS Algorithms for Nonnegative Matrix and 3D Tensor Factorization", International Conf. on ICA.

Gillis-Glineur 2012, "Accelerated Multiplicative Updates and Hierarchical ALS Algo. for NMF", Neural Computation.

A-HALS = avoids repeated computations + re-uses

Projected† gradient descent with step size $t \geq 0$

$$\mathbf{w}_i = \mathbf{w}_i - t\,\underbrace{(\|\mathbf{h}_i\|_2^2\mathbf{w}_i - \mathbf{X}_i\mathbf{h}_i^\top)}_{\nabla_{\mathbf{w}_i}\Phi}, \quad \mathbf{h}_i = \mathbf{h}_i - t\,\underbrace{(\|\mathbf{w}_i\|_2^2\mathbf{h}_i - \mathbf{w}_i^\top\mathbf{X})}_{\nabla_{\mathbf{h}_i}\Phi}.$$

A-HALS = avoids repeated computations + re-uses

Projected† gradient descent with step size $t \geq 0$

$$\mathbf{w}_i = \mathbf{w}_i - t\,\underbrace{\left(\|\mathbf{h}_i\|_2^2\mathbf{w}_i - \mathbf{X}_i\mathbf{h}_i^\top\right)}_{\nabla_{\mathbf{w}_i}\Phi}, \quad \mathbf{h}_i = \mathbf{h}_i - t\,\underbrace{\left(\|\mathbf{w}_i\|_2^2\mathbf{h}_i - \mathbf{w}_i^\top\mathbf{X}\right)}_{\nabla_{\mathbf{h}_i}\Phi}.$$

Algorithm HALS

1: $\mathbf{w}_1 = \mathbf{w}_1 - t(\|\mathbf{h}_1\|_2^2\mathbf{w}_1 - \mathbf{X}_1\mathbf{h}_1^\top)$
2: $\mathbf{h}_1 = \mathbf{h}_1 - t(\|\mathbf{w}_1\|_2^2\mathbf{h}_1 - \mathbf{w}_1^\top\mathbf{X}_1)$
3: $\mathbf{w}_2 = \mathbf{w}_2 - t(\|\mathbf{h}_2\|_2^2\mathbf{w}_2 - \mathbf{X}_2\mathbf{h}_2^\top)$
4: $\mathbf{h}_2 = \mathbf{h}_2 - t(\|\mathbf{w}_2\|_2^2\mathbf{h}_2 - \mathbf{w}_2^\top\mathbf{X}_2)$
5: $\mathbf{w}_3 = \mathbf{w}_3 - t(\|\mathbf{h}_3\|_2^2\mathbf{w}_3 - \mathbf{X}_3\mathbf{h}_3^\top)$
6: $\mathbf{h}_3 = \mathbf{h}_3 - t(\|\mathbf{w}_3\|_2^2\mathbf{h}_3 - \mathbf{w}_3^\top\mathbf{X}_3)$
7: ...

Algorithm A-HALS

1: Compute $\mathbf{A} = \mathbf{H}\mathbf{H}^\top$, $\mathbf{B} = \mathbf{X}\mathbf{H}^\top$
2: $\mathbf{w}_1 = \mathbf{w}_1 - t(\|\mathbf{h}_1\|_2^2\mathbf{w}_1 - \mathbf{X}_1\mathbf{h}_1^\top)$
3: $\mathbf{w}_2 = \mathbf{w}_2 - t(\|\mathbf{h}_2\|_2^2\mathbf{w}_2 - \mathbf{X}_2\mathbf{h}_2^\top)$
4: $\mathbf{w}_3 = \mathbf{w}_3 - t(\|\mathbf{h}_3\|_2^2\mathbf{w}_3 - \mathbf{X}_3\mathbf{h}_3^\top)$
5: Compute $\mathbf{C} = \mathbf{W}^\top\mathbf{W}$, $\mathbf{D} = \mathbf{W}^\top\mathbf{X}$
6: $\mathbf{h}_1 = \mathbf{h}_1 - t(\|\mathbf{w}_1\|_2^2\mathbf{h}_1 - \mathbf{w}_1^\top\mathbf{X}_1)$
7: $\mathbf{h}_2 = \mathbf{h}_2 - t(\|\mathbf{w}_2\|_2^2\mathbf{h}_2 - \mathbf{w}_2^\top\mathbf{X}_2)$
8: $\mathbf{h}_3 = \mathbf{h}_3 - t(\|\mathbf{w}_3\|_2^2\mathbf{h}_3 - \mathbf{w}_3^\top\mathbf{X}_3)$
9: ...

A-HALS : Line 2-4, 6-8 repeated a few times.

A-HALS = avoids repeated computations + re-uses

Projected† gradient descent with step size $t \geq 0$

$$\mathbf{w}_i = \mathbf{w}_i - t\underbrace{\left(\|\mathbf{h}_i\|_2^2\mathbf{w}_i - \mathbf{X}_i\mathbf{h}_i^\top\right)}_{\nabla_{\mathbf{w}_i}\Phi}, \quad \mathbf{h}_i = \mathbf{h}_i - t\underbrace{\left(\|\mathbf{w}_i\|_2^2\mathbf{h}_i - \mathbf{w}_i^\top\mathbf{X}\right)}_{\nabla_{\mathbf{h}_i}\Phi}.$$

Algorithm HALS

1: $\mathbf{w}_1 = \mathbf{w}_1 - t(\|\mathbf{h}_1\|_2^2\mathbf{w}_1 - \mathbf{X}_1\mathbf{h}_1^\top)$
2: $\mathbf{h}_1 = \mathbf{h}_1 - t(\|\mathbf{w}_1\|_2^2\mathbf{h}_1 - \mathbf{w}_1^\top\mathbf{X}_1)$
3: $\mathbf{w}_2 = \mathbf{w}_2 - t(\|\mathbf{h}_2\|_2^2\mathbf{w}_2 - \mathbf{X}_2\mathbf{h}_2^\top)$
4: $\mathbf{h}_2 = \mathbf{h}_2 - t(\|\mathbf{w}_2\|_2^2\mathbf{h}_2 - \mathbf{w}_2^\top\mathbf{X}_2)$
5: $\mathbf{w}_3 = \mathbf{w}_3 - t(\|\mathbf{h}_3\|_2^2\mathbf{w}_3 - \mathbf{X}_3\mathbf{h}_3^\top)$
6: $\mathbf{h}_3 = \mathbf{h}_3 - t(\|\mathbf{w}_3\|_2^2\mathbf{h}_3 - \mathbf{w}_3^\top\mathbf{X}_3)$
7: ...

Algorithm A-HALS

1: Compute $\mathbf{A} = \mathbf{H}\mathbf{H}^\top$, $\mathbf{B} = \mathbf{X}\mathbf{H}^\top$
2: $\mathbf{w}_1 = \mathbf{w}_1 - t(\|\mathbf{h}_1\|_2^2\mathbf{w}_1 - \mathbf{X}_1\mathbf{h}_1^\top)$
3: $\mathbf{w}_2 = \mathbf{w}_2 - t(\|\mathbf{h}_2\|_2^2\mathbf{w}_2 - \mathbf{X}_2\mathbf{h}_2^\top)$
4: $\mathbf{w}_3 = \mathbf{w}_3 - t(\|\mathbf{h}_3\|_2^2\mathbf{w}_3 - \mathbf{X}_3\mathbf{h}_3^\top)$
5: Compute $\mathbf{C} = \mathbf{W}^\top\mathbf{W}$, $\mathbf{D} = \mathbf{W}^\top\mathbf{X}$
6: $\mathbf{h}_1 = \mathbf{h}_1 - t(\|\mathbf{w}_1\|_2^2\mathbf{h}_1 - \mathbf{w}_1^\top\mathbf{X}_1)$
7: $\mathbf{h}_2 = \mathbf{h}_2 - t(\|\mathbf{w}_2\|_2^2\mathbf{h}_2 - \mathbf{w}_2^\top\mathbf{X}_2)$
8: $\mathbf{h}_3 = \mathbf{h}_3 - t(\|\mathbf{w}_3\|_2^2\mathbf{h}_3 - \mathbf{w}_3^\top\mathbf{X}_3)$
9: ...

A-HALS : Line 2-4, 6-8 repeated a few times.

A-HALS avoids repeated computations of *constant terms* :

$$\mathbf{H}\mathbf{H}^\top_{(2n-1)m^2},\ \mathbf{X}\mathbf{H}^\top_{(2n-1)mr},\ \mathbf{W}^\top\mathbf{W}_{(2r-1)m^2},\ \mathbf{W}^\top\mathbf{X}_{(2m-1)rn},$$

pre-computing and *re-use* of these terms gain extra efficiency,
improvement is significant for big data#" — always A-HALS!

†Projection step not shown here. # Even more significant in terms of BLAS if the matrices are *sparse*.

The projected gradient descent update

The Projected Gradient Descent update of $\mathbf{W}$:

$$\mathbf{W}^{k+1} = \mathrm{Proj}_{\mathbb{R}_+}\Big(\mathbf{W}^k - t\nabla\Phi(\mathbf{W}^k, \mathbf{H})\Big).$$

How to pick the step-size ?

The projected gradient descent update

The Projected Gradient Descent update of $\mathbf{W}$:

$$\mathbf{W}^{k+1} = \mathrm{Proj}_{\mathbb{R}_+}\Big(\mathbf{W}^k - t\nabla\Phi(\mathbf{W}^k, \mathbf{H})\Big).$$

How to pick the step-size ?

A simple scheme $t = \dfrac{1}{L_{\Phi_\mathbf{W}}}$, where $L_{\Phi_\mathbf{W}} =$ the Lipschitz constant of $\nabla_\mathbf{W}\Phi$ (smoothness constant).

The projected gradient descent update

The Projected Gradient Descent update of $\mathbf{W}$:

$$\mathbf{W}^{k+1} = \mathrm{Proj}_{\mathbb{R}_+}\Big(\mathbf{W}^k - t\nabla\Phi(\mathbf{W}^k, \mathbf{H})\Big).$$

How to pick the step-size ?

A simple scheme $t = \dfrac{1}{L_{\Phi_{\mathbf{W}}}}$, where $L_{\Phi_{\mathbf{W}}} =$ the Lipschitz constant of $\nabla_{\mathbf{W}}\Phi$ (smoothness constant).

- $L_{\Phi_{\mathbf{W}}} =$ largest singular value of $\mathbf{H}\mathbf{H}^\top$
- $\mathrm{Proj}_{\mathbb{R}_+}$ is basically $[\ \cdot\ ]_+ = \max\{\cdot, 0\}$.

Hence in close form :

$$\mathbf{W}^{k+1} = \left[\mathbf{W}^k - \frac{1}{\sigma_{\max}(\mathbf{H}\mathbf{H}^\top)}\nabla\Phi(\mathbf{W}^k, \mathbf{H})\right]_+.$$

PGD update is much faster than the *Multiplicative Update*.

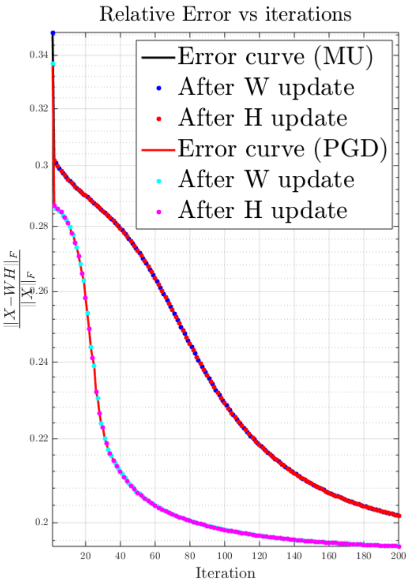
Multiplicative Update

MU :

- It takes a small step size t such that $\mathbf{W}^{k+1}$ stays within $\mathbb{R}_+$, no projection.
$$\mathbf{W}^{k+1} = \mathbf{W} . * \frac{\mathbf{X}\mathbf{H}^\top}{\mathbf{W}^k\mathbf{H}\mathbf{H}^\top},$$
where $*$ is Hadamard product and the division is Hadamard quotient.
- It converges **very slowly**. In general, don't use MU.
Why: to make sure $\mathbf{W}$ stays within $\mathbb{R}_+$, MU take small step $\implies$ slow !

PGD :

- It takes reasonably large step size, and
IF moved outside $\mathbb{R}_+$ **THEN** project back.
- $\mathrm{Proj}_{\mathbb{R}_+}$ practically costs nothing unless the data size is 10^{86}.



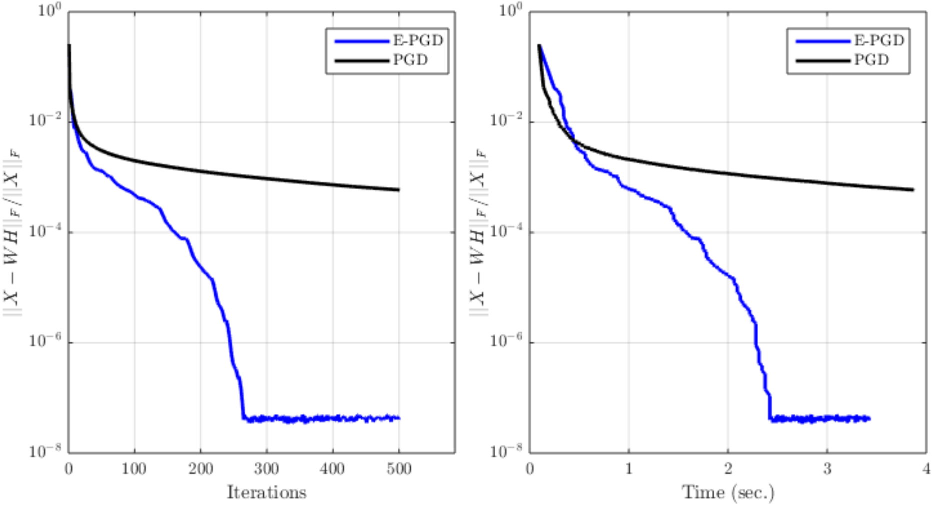
MU = timid, shy guy that is too cautious on making mistake.
PGD = brave guy that is fine of making mistake by doing correction.
Here "mistake" = "outside $\mathbb{R}_+$", "correction" = "$\mathrm{Proj}_{\mathbb{R}_+}$".

Outline

1. Introduction - Non-negative Matrix Factorization
2. Computing NMF
   - Variations on BCD
   - A-HALS
   - Matrix-wise Projected Gradient Update and the Multiplicative update
3. Find $(\mathbf{W}, \mathbf{H})$ numerically fast : acceleration via extrapolation
   - Recall : acceleration in single variable problem
   - Accelerating NMF algorithms using extrapolation
4. Computing NTF
5. Computing NNLS

Let's accelerate !

The next many slides : make PGD converges even more fast



Recall : NMF is **NP-Hard**.

What's the acceleration for : obtain a *local* solution faster.

Recall : acceleration in single variable problem

Problem $\min_{x \in \mathcal{C}} f(x)$, $\mathcal{C}$ convex set.

Recall : acceleration in single variable problem

Problem $\min_{x \in \mathcal{C}} f(x)$, $\mathcal{C}$ convex set.

At step k :

No acceleration : $x_{k+1} = \text{Update}[x_k]$.

With acceleration : $x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = \text{Extrapolate}[x_{k+1}, x_k]$.

Recall : acceleration in single variable problem

Problem $\min_{x \in \mathcal{C}} f(x)$, $\mathcal{C}$ convex set.

At step k :

No acceleration : $x_{k+1} = \text{Update}[x_k]$.

With acceleration : $x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = \text{Extrapolate}[x_{k+1}, x_k]$.

To be specific :

$$
\begin{aligned}
\text{PGD Update} \quad & x_{k+1} = \text{Proj}_{\mathcal{C}}(x_k - t_k \nabla f(x_k)). \\
\text{Linear extrapolation} \quad & x_{k+1} = \text{Proj}_{\mathcal{C}}(y_k - t_k \nabla f(y_k)). \\
& y_{k+1} = x_{k+1} + \beta_k (x_{k+1} - x_k).
\end{aligned}
$$

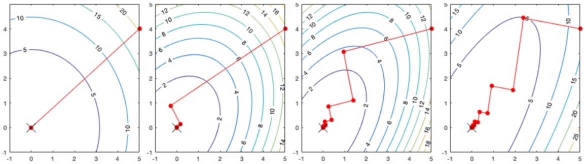
i.e. $\text{Extrapolate}[x_{k+1}, x_k]$ is modeled by β_k : a single extrapolation parameter.

Why extrapolation : gradient descent zig-zags on ellipse

Facts : consecutive update directions of GD are orthogonal ($\perp$).
If the landscape is not "spherical", GD zig-zags $\rightarrow$ slow.
e.g. : moving along a long narrow valley.

Observations

- When the level set of $f(\mathbf{x})$ is circular, GD goes to $\mathbf{x}^*$ very fast. (In fact, in 1 step GD goes to $\mathbf{x}^*$)
- When the level set of $f(\mathbf{x})$ is elliptic, GD zigzags (and slow).



Questions

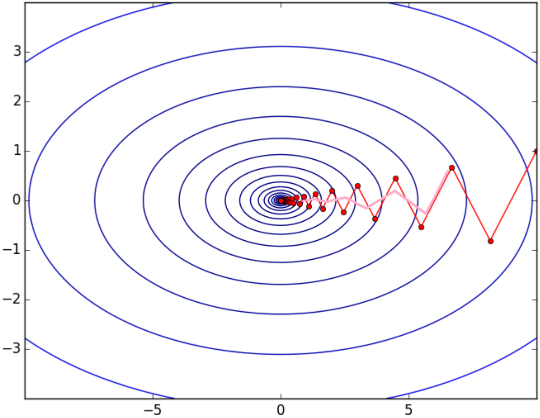
- Why? Where does this zigzag come from?
- How to deal with it : how to improve GD?

26 / 144

Picture from https://angms.science/doc/teaching/GDLS.pdf

What machine learning people do to counter zig-zag?

Do tricks on step size : don't move with step size t but $\dfrac{t}{\text{damping factor}}$.

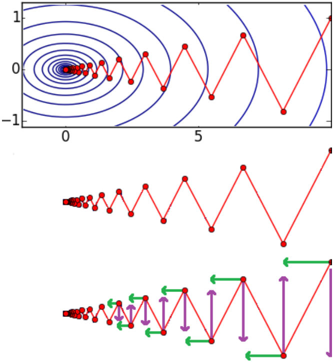


Length of pink segment < length of the corresponding red segment $\Longrightarrow$ points on pink segment is closer to axis $y = 0$, gradient stronger x-component $\Longrightarrow$ less oscillation along y-direction.

The idea behind **AdaGrad** and **AdaDelta** : shrink the step size when you see zig-zag (trace of the objective function appears to plateau).

What optimization people do to counter zig-zag?

Do tricks on direction : by extrapolation with momentum.



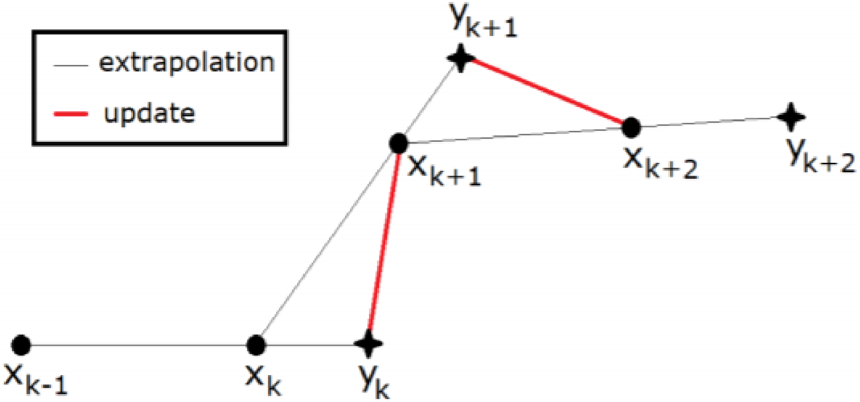
Idea : apply extrapolation.
Extrapolate = add gradient history.

(1) if gradients in consecutive steps have consistent direction
$\implies$ extrapolate = accelerate.
(2) if gradients in consecutive steps oscillates (continuously changing direction)
$\implies$ extrapolate = damp oscillation = acceleration.

Figure shows the trace of points decomposed into x- and y-component.
The x-components have consistent direction while y-components are not.

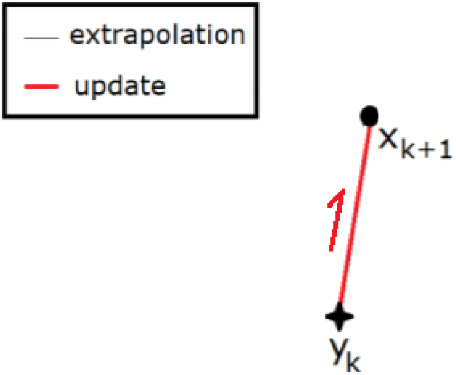
The geometry of the extrapolation

$$x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k).$$



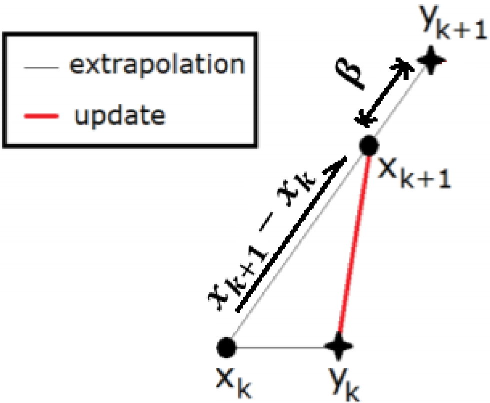
The geometry of the extrapolation

$$x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k).$$



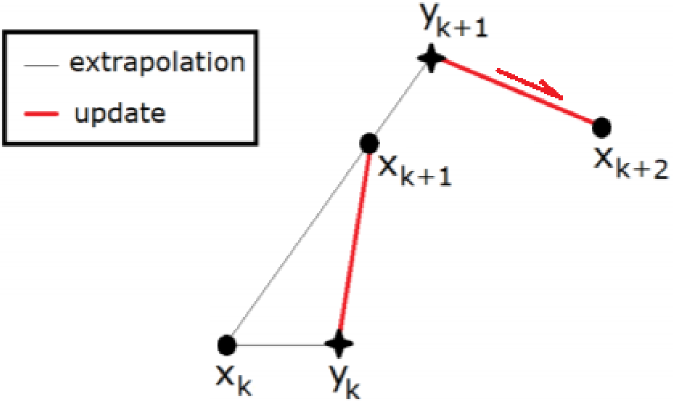
The geometry of the extrapolation

$$x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k).$$



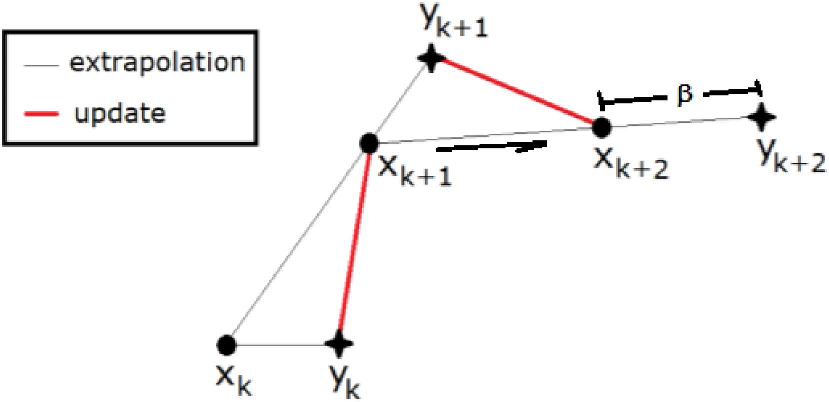
The geometry of the extrapolation

$$x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k).$$



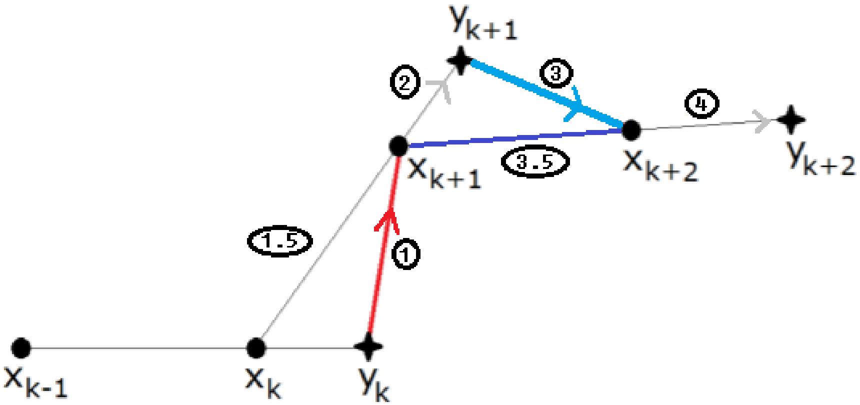
The geometry of the extrapolation

$$x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k).$$



The geometry of the extrapolation

$$x_{k+1} = \text{Update}[y_k], \quad y_{k+1} = x_{k+1} + \beta_k(x_{k+1} - x_k).$$

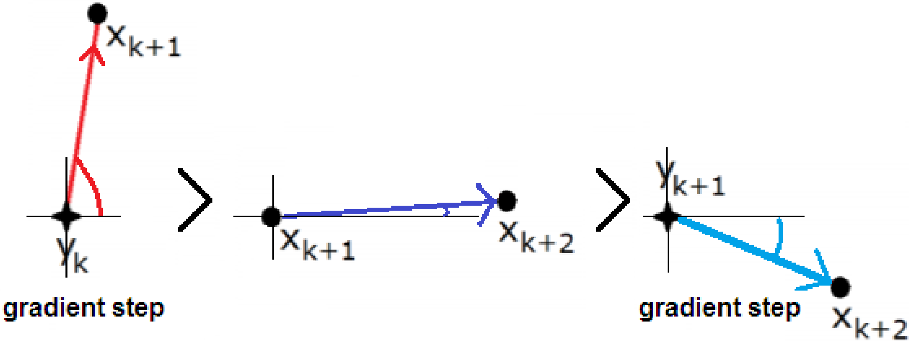


The geometry of extrapolation

We always have

$$\angle(x_{k+1} - y_k) \geq \angle(x_{k+2} - x_{x+1}) \geq \angle(x_{k+2} - y_{k+1})$$

i.e. the direction of the last step is **in between** the directions of previous two gradient steps : zig-zag effect is reduced !



Nesterov's acceleration

For **convex** (smooth strongly-convex) function

Other β_k schemes

Nesterov's parameters looks so complicated

$$\alpha_{k+1} = \frac{\sqrt{\alpha_k^4 + 4\alpha_k^2} - \alpha_k^2}{2}, \ \ \beta_k = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2 + \alpha_{k+1}}$$

Another Nesterov's parameters

$$\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2 + \kappa^{-1}\alpha_{k+1}, \ \ \beta_k = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2 + \alpha_{k+1}}$$

Yet another Nesterov's parameters

$$\alpha_{k+1} = \frac{1 + \sqrt{1 + 4\alpha_k^2}}{2}, \ \ \beta_k = \frac{1-\alpha_k}{\alpha_{k+1}}.$$

Paul Tseng parameter

$$\beta_k = \frac{k-1}{k+2}.$$

Using conditional number

$$\beta_k = \beta = \frac{1-\sqrt{\kappa'}}{1+\sqrt{\kappa'}}, \quad \kappa' = \frac{1}{\kappa}, \ \kappa = \frac{\sigma_{\max}(\mathbf{Q})}{\sigma_{\min}(\mathbf{Q})} = \frac{\lambda_{\max}(\mathbf{Q})}{\lambda_{\min}(\mathbf{Q})}$$

70 / 144

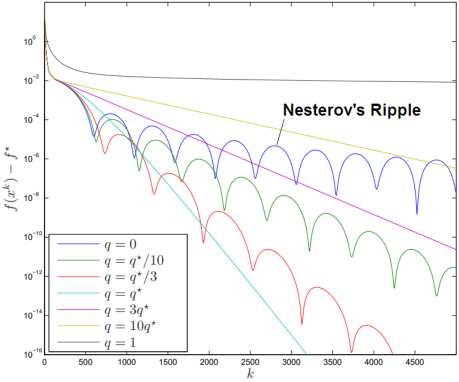
Picture from https://angms.science/doc/teaching/GDLS.pdf

Key : Nesterov's acceleration has a close-form formula for β_k

Extrapolation is not monotone, nor descent, nor greedy

GD is locally optimal/greedy $\implies$ extrapolation may $\uparrow$objective value

- Extrapolation = a risky move

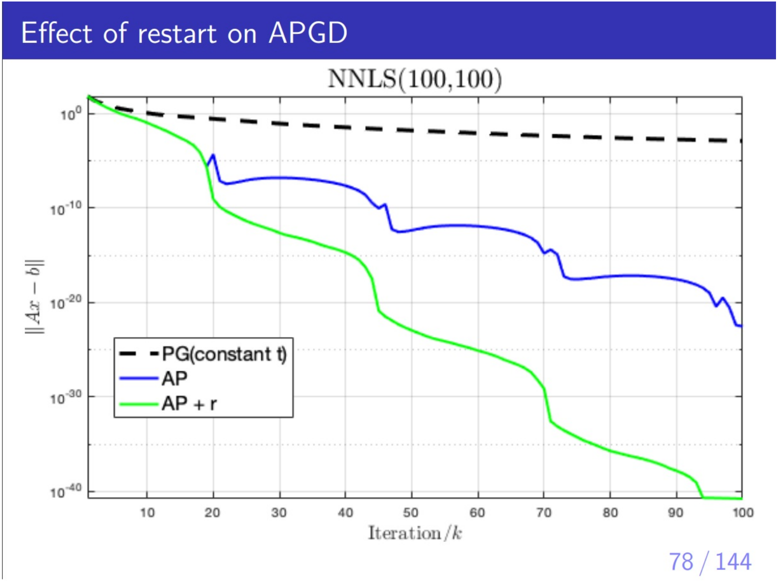


Picture from Donoghue-Candés 2015, "Adaptive Restart for Accelerated Gradient Schemes"

Acceleration comes from doing the risky move :

"sacrifice the **decreases** of objective value now for the better future"

Actually also sacrifice robustness : accelerated gradient is not stable to noise (Devolder-Glineur-Nesterov 2014)



Picture from https://angms.science/doc/teaching/GDLS.pdf

Our case : NMF is not cvx

$(\mathcal{P})$: $\left\{ \text{Given } (\mathbf{X}, r), \text{ solve } \min_{\mathbf{W},\mathbf{H}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|^2, \mathbf{W}, \mathbf{H} \in \mathbb{R}_+ \right\}$ is non-cvx.

- No strong cvx parameter, cannot use expression likes $\beta_k = \dfrac{1 - \sqrt{\kappa}}{1 + \sqrt{\kappa}}$.
- Direct application of Nesterov's β sequence on PGD/A-HALS will give erratic convergence behaviour

  Mitchell, Drew, Nan Ye, and Hans De Sterck. "Nesterov Acceleration of Alternating Least Squares for Canonical Tensor Decomposition." arXiv:1810.05846 (2018)

Our case : NMF is not cvx

$(\mathcal{P})$: $\left\{ \text{Given } (\mathbf{X}, r), \text{ solve } \min_{\mathbf{W},\mathbf{H}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|^2, \mathbf{W}, \mathbf{H} \in \mathbb{R}_+ \right\}$ is non-cvx.

- No strong cvx parameter, cannot use expression likes $\beta_k = \dfrac{1-\sqrt{\kappa}}{1+\sqrt{\kappa}}$.
- Direct application of Nesterov's β sequence on PGD/A-HALS will give erratic convergence behaviour

  Mitchell, Drew, Nan Ye, and Hans De Sterck. "Nesterov Acceleration of Alternating Least Squares for Canonical Tensor Decomposition." arXiv:1810.05846 (2018)

For the acceleration scheme of the two variables

$$
\begin{cases}
\text{On } \mathbf{W} & \begin{cases} \text{Update} & \mathbf{W}_{\text{new}} = \text{Update}[\mathbf{Y}_{\text{old}}, \mathbf{H}_{\text{old}}] \\ \text{Extrapolate} & \mathbf{Y}_{\text{new}} = \mathbf{W}_{\text{new}} + \beta_k^{\mathbf{W}}(\mathbf{W}_{\text{new}} - \mathbf{W}_{\text{old}}) \end{cases} \\
\text{On } \mathbf{H} & \begin{cases} \text{Update} & \mathbf{H}_{\text{new}} = \text{Update}[\mathbf{W}_{\text{new}}, \mathbf{G}_{\text{old}}] \\ \text{Extrapolate} & \mathbf{G}_{\text{new}} = \mathbf{H}_{\text{new}} + \beta_k^{\mathbf{H}}(\mathbf{H}_{\text{new}} - \mathbf{H}_{\text{old}}) \end{cases}
\end{cases}
$$

Need a way (close-/no close-form) to find β_k !

Approach : an ad hoc heurisitic in the "line search" style.

Why ad hoc heuristics ?

- (1) The ncvx problem is hard.
- (2) No better idea.
- No convergence theorem now yet (because of (1)).

What's so good ?

- Just a parameter tuning problem.
- Easy to implement.
- Easy to extend to other models.
- Faster than state-of-the-art methods with theoretical convergence proof !

† Xu-Yin 2013 "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion". SIAM J. Img Sci.

Details of the extrapolation

The key β_k

- β has to be smaller than 1 (same as the convex case)
- If $\beta \in (0, 1)$: extrapolation, doing risky step

Details of the extrapolation

The key β_k

- β has to be smaller than 1 (same as the convex case)
- If $\beta \in (0,1)$: extrapolation, doing risky step
- If $\beta = \{1,0\}$: doing {very risky, no} extrapolation

Details of the extrapolation

The key β_k

- β has to be smaller than 1 (same as the convex case)
- If $\beta \in (0,1)$: extrapolation, doing risky step
- If $\beta = \{1,0\}$: doing {very risky, no} extrapolation
- Can't use line search† to find β : experimentally found β close to 0
  - effectively doing no extrapolation, waste resource on line search

Details of the extrapolation

The key β_k

- β has to be smaller than 1 (same as the convex case)
- If $\beta \in (0,1)$: extrapolation, doing risky step
- If $\beta = \{1,0\}$: doing {very risky, no} extrapolation
- Can't use line search† to find β : experimentally found β close to 0
  - effectively doing no extrapolation, waste resource on line search

In the "walking person metaphor" :

MU	shy guy walking in caution with small step size
PGD	brave guy walking with reasonably step size
E-PGD	ambious guy walking with big step size
E-A-HALS	crazy guy walking with big step size in coordinate manner

†Line search to minimize the objective function directly – performed **before** the update

Details : Update[β_k]

Landscape of variable at each iteration is different $\implies$ dynamical update

Algorithm A dynamic **line search style**[†] **ad hoc heuristics**

Input: Parameters $1 < \bar{\gamma} < \gamma < \eta$, an initialization $\beta_1 \in (0,1)$
Output: β_k : the extrapolation parameter

1: Set $\bar{\beta} = 1$ (dynamic "upper bound" of β)
2: **if** error ↓ at iteration k **then**
3: Increase β_{k+1} : $\beta_{k+1} = \min\{\bar{\beta}, \gamma\beta_k\}$
4: (Increase $\bar{\beta}$ if $\bar{\beta} < 1$: $\bar{\beta} = \min\{1, \bar{\gamma}\bar{\beta}\}$)
5: **else**
6: Decrease β_{k+1} : $\beta_{k+1} = \beta_k/\eta$
7: Set $\bar{\beta} = \beta_k$
8: **end if**

γ, $\bar{\gamma}$, η : growth and decay parameters

† Line search **after** updates of $\mathbf{W}$ and $\mathbf{H}$ – performed **after** the update!

The logic of tuning β

The idea is to update β_k based on the increase or decrease of the objective function. Let $e^k = \Phi(\mathbf{W}^k, \mathbf{H}^k)$, then

$$\beta_{k+1} = \begin{cases} \min\{\gamma\beta_k, \bar{\beta}\} & \text{if } e^k \leq e^{k-1} \\ \dfrac{\beta_k}{\eta} & \text{if } e^k > e^{k-1} \end{cases} \tag{1}$$

where $\gamma > 1$, and $\eta > 1$ are constants and $\bar{\beta}_0 = 1$ with the update

The logic of tuning β

The idea is to update β_k based on the increase or decrease of the objective function. Let $e^k = \Phi(\mathbf{W}^k, \mathbf{H}^k)$, then

$$\beta_{k+1} = \begin{cases} \min\{\gamma\beta_k, \bar{\beta}\} & \text{if } e^k \leq e^{k-1} \\ \dfrac{\beta_k}{\eta} & \text{if } e^k > e^{k-1} \end{cases} \tag{1}$$

where $\gamma > 1$, and $\eta > 1$ are constants and $\bar{\beta}_0 = 1$ with the update

$$\bar{\beta}_{k+1} = \begin{cases} \min\{\bar{\gamma}\bar{\beta}_k, 1\} & \text{if } e^k \leq e^{k-1} \text{ and } \bar{\beta}_k < 1 \\ \beta_k & \text{if } e^k > e^{k-1} \end{cases}. \tag{2}$$

The detail logic flow of updating β_k ... (1/2)

Case 1. The error decreases : $e^k \leq e^{k-1}$

- It means the current β value is "good"

The detail logic flow of updating β_k ... (1/2)

Case 1. The error decreases : $e^k \leq e^{k-1}$

- It means the current β value is "good"
- We can be more ambitious on the extrapolation
  - i.e., we increase the value of β
  - How : multiplying it with a growth factor $\gamma > 1$

$$\beta_{k+1} = \beta_k \gamma$$

The detail logic flow of updating β_k ... (1/2)

Case 1. The error decreases : $e^k \leq e^{k-1}$

- It means the current β value is “good”
- We can be more ambitious on the extrapolation
  - i.e., we increase the value of β
  - How : multiplying it with a growth factor $\gamma > 1$

$$\beta_{k+1} = \beta_k \gamma$$

- Note that the growth of β cannot be indefinite
  - i.e., we put a ceiling parameter $\bar{\beta}$ to upper bound the growth
  - How : use $\min$

$$\beta_{k+1} = \min\{\beta_k \gamma, \bar{\beta}_k\}$$

  - $\bar{\beta}$ itself is also updated dynamically with a growth factor $\bar{\gamma}$ with the upper bound 1.

The detail logic flow of updating β_k ... (2/2)

Case 2. The error increases : $e^k > e^{k-1}$

- It means the current β value is "bad" (too large)

The detail logic flow of updating β_k ... (2/2)

Case 2. The error increases : $e^k > e^{k-1}$

- It means the current β value is "bad" (too large)
- We become less ambitious on the extrapolation
  - i.e., we decrease the value of β
  - How : dividing it with the decay factor $\eta > 1$

$$\beta_{k+1} = \frac{\beta_k}{\eta}$$

The detail logic flow of updating β_k ... (2/2)

Case 2. The error increases : $e^k > e^{k-1}$

- It means the current β value is "bad" (too large)
- We become less ambitious on the extrapolation
  - i.e., we decrease the value of β
  - How : dividing it with the decay factor $\eta > 1$

$$\beta_{k+1} = \frac{\beta_k}{\eta}$$

- As f is often a continuous and smooth, for β_k being too large, such value of β will also be too large at iteration $k+1$
  - i.e., we have to avoid β_{k+1} to grow back to β_k (the "bad" value) too soon
  - How : we set the ceiling parameter

$$\bar{\beta}_{k+1} = \beta_k$$

The full algo of Accelerated NMF using extrapolation

Input: $\mathbf{X}$, initialization $\mathbf{W}, \mathbf{H}$, parameters $hp \in \{1, 2, 3\}$ (extrapolation/projection of $\mathbf{H}$).
Output: $\mathbf{W}, \mathbf{H}$.

$\mathbf{W}_y = \mathbf{W}$; $\mathbf{H}_y = \mathbf{H}$; $e(0) = ||\mathbf{X} - \mathbf{W}\mathbf{H}||_F$.
for $k = 1, 2, \ldots$ **do**
  Compute $\mathbf{H}_n$ by $\min_{\mathbf{H}_n \geq 0} ||\mathbf{X} - \mathbf{W}_y \mathbf{H}_n||_F^2$ using $\mathbf{H}_y$ as initial iterate.
  if $hp \geq 2$ **then**
    Extrapolate: $\mathbf{H}_y = \mathbf{H}_n + \beta_k (\mathbf{H}_n - \mathbf{H})$.
  end if
  if $hp = 3$ **then**
    Project: $\mathbf{H}_y = \max\left(0, \mathbf{H}_y\right)$.
  end if
  Compute $\mathbf{W}_n$ by $\min_{\mathbf{W}_n \geq 0} ||\mathbf{X} - \mathbf{W}_n \mathbf{H}_y||_F^2$ using $\mathbf{W}_y$ as initial iterate.
  Extrapolate: $\mathbf{W}_y = \mathbf{W}_n + \beta_k (\mathbf{W}_n - \mathbf{W})$.
  if $hp = 1$ **then**
    Extrapolate: $\mathbf{H}_y = \mathbf{H}_n + \beta_k (\mathbf{H}_n - \mathbf{H})$.
  end if
  Compute error: $\mathrm{e}(k) = ||\mathbf{X} - \mathbf{W}_n \mathbf{H}_y||_F$.
  if $\mathrm{e}(k) > \mathrm{e}(k-1)$ **then**
    Restart: $\mathbf{H}_y = \mathbf{H}_n$; $\mathbf{W}_y = \mathbf{W}_n$.
  else
    $\mathbf{H} = \mathbf{H}_n$; $\mathbf{W} = \mathbf{W}_n$.
  end if
end for

Notation : $\mathbf{W}_n$ normal variable, $\mathbf{W}_y$ extrpolate variable, $\mathbf{W}$ previous $\mathbf{W}_n$
... too hard to read !!

Algorithm ($hp = 1$), simplified

Input: $\mathbf{X}$, initialization $\mathbf{W}, \mathbf{H}$
Output: $\mathbf{W}, \mathbf{H}$

$\mathbf{W}_y = \mathbf{W}$; $\mathbf{H}_y = \mathbf{H}$; $e(0) = ||\mathbf{X} - \mathbf{W}\mathbf{H}||_F$.
for $k = 1, 2, \ldots$ **do**
  Update[$\mathbf{H}_n$] w.r.t. $\mathbf{H}_n \geq 0$ with $\mathbf{X}, \mathbf{W}_y, \mathbf{H}_n$ using $\mathbf{H}_y$ as initial iterate.
  Update[$\mathbf{W}_n$] wr.t. $\mathbf{W}_n \geq 0$ with $\mathbf{X}, \mathbf{W}_n, \mathbf{H}_y$ using $\mathbf{W}_y$ as initial iterate.
  Extrapolate[$\mathbf{W}_y$] : $\mathbf{W}_y = \mathbf{W}_n + \beta_k(\mathbf{W}_n - \mathbf{W})$.
  Extrapolate[$\mathbf{H}_y$] : $\mathbf{H}_y = \mathbf{H}_n + \beta_k(\mathbf{H}_n - \mathbf{H})$.
  Compute error: $\mathrm{e}(k) = ||\mathbf{X} - \mathbf{W}_n\mathbf{H}_y||_F$.
  if $\mathrm{e}(k) > \mathrm{e}(k-1)$ **then**
    Restart: $\mathbf{H}_y = \mathbf{H}_n$; $\mathbf{W}_y = \mathbf{W}_n$.
  else
    $\mathbf{H} = \mathbf{H}_n$; $\mathbf{W} = \mathbf{W}_n$.
  end if
end for

"Up, Up, Ex, Ex"

Algorithm ($hp = 2$), simplified

Input: $\mathbf{X}$, initialization $\mathbf{W}, \mathbf{H}$
Output: $\mathbf{W}, \mathbf{H}$

```
W_y = W; H_y = H; e(0) = ||X − WH||_F.
for k = 1, 2, ... do
    Update[H_n] w.r.t. H_n ≥ 0 with X, W_y, H_n using H_y as initial iterate.
    Extrapolate[H_y] : H_y = H_n + β_k(H_n − H).
    Update[W_n] wr.t. W_n ≥ 0 with X, W_n, H_y using W_y as initial iterate.
    Extrapolate[W_y] : W_y = W_n + β_k(W_n − W).
    Compute error: e(k) = ||X − W_n H_y||_F.
    if e(k) > e(k − 1) then
        Restart: H_y = H_n; W_y = W_n.
    else
        H = H_n; W = W_n.
    end if
end for
```

"Up, Ex, Up, Ex"

Algorithm ($hp = 3$), simplified

Input: $\mathbf{X}$, initialization $\mathbf{W}, \mathbf{H}$
Output: $\mathbf{W}, \mathbf{H}$

$\mathbf{W}_y = \mathbf{W}$; $\mathbf{H}_y = \mathbf{H}$; $e(0) = ||\mathbf{X} - \mathbf{W}\mathbf{H}||_F$.
for $k = 1, 2, \ldots$ **do**
  Update[$\mathbf{H}_n$] w.r.t. $\mathbf{H}_n \geq 0$ with $\mathbf{X}, \mathbf{W}_y, \mathbf{H}_n$ using $\mathbf{H}_y$ as initial iterate.
  Extrapolate[$\mathbf{H}_y$] : $\mathbf{H}_y = \mathbf{H}_n + \beta_k(\mathbf{H}_n - \mathbf{H})$.
  Project: $\mathbf{H}_y = \max(0, \mathbf{H}_y)$.
  Update[$\mathbf{W}_n$] wr.t. $\mathbf{W}_n \geq 0$ with $\mathbf{X}, \mathbf{W}_n, \mathbf{H}_y$ using $\mathbf{W}_y$ as initial iterate.
  Extrapolate[$\mathbf{W}_y$] : $\mathbf{W}_y = \mathbf{W}_n + \beta_k(\mathbf{W}_n - \mathbf{W})$.
  Compute the error: $\mathrm{e}(k) = ||\mathbf{X} - \mathbf{W}_n\mathbf{H}_y||_F$.
  if $\mathrm{e}(k) > \mathrm{e}(k-1)$ **then**
    Restart: $\mathbf{H}_y = \mathbf{H}_n$; $\mathbf{W}_y = \mathbf{W}_n$.
  else
    $\mathbf{H} = \mathbf{H}$; $\mathbf{W} = \mathbf{W}_n$.
  end if
end for

"Up, Ex, Pro, Up, Ex"

Summary and notes (1/3)

Extrapolation may break NN (≥ 0) constraint :

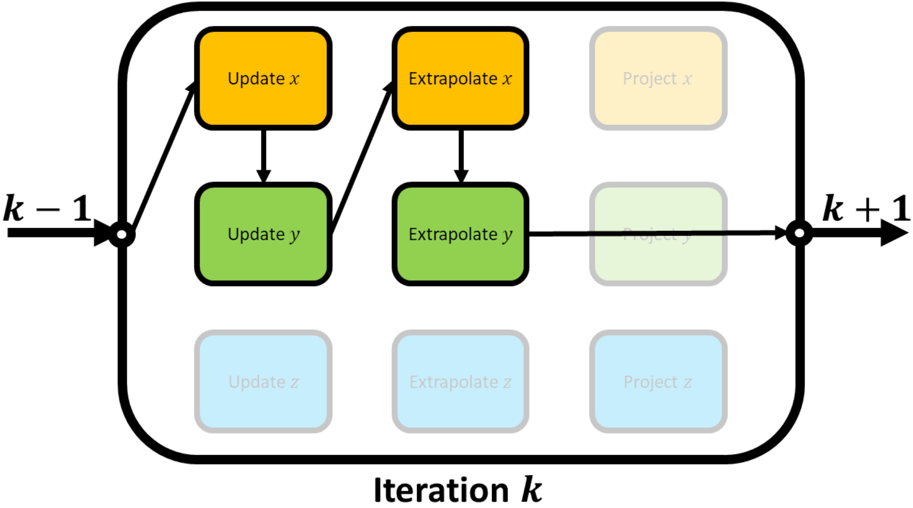
$hp = 1$		$hp = 2$		$hp = 3$	
(Up-Up-Ex-Ex)		(Up-Ex-Up-Ex)		(Up-Ex-Pro-Up-Ex)	
Step	NN?	Step	NN?	Step	NN?
Update$[\mathbf{H}_n]$	Y	Update$[\mathbf{H}_n]$	Y	Update$[\mathbf{H}_n]$	Y
Update$[\mathbf{W}_n]$	Y	Extrap$[\mathbf{H}_y]$	N	Extrap$[\mathbf{H}_y]$	N
				Project$[\mathbf{H}_y]$	Y
Extrap$[\mathbf{H}_y]$	N	Update$[\mathbf{W}_n]$	Y	Update$[\mathbf{W}_n]$	Y
Extrap$[\mathbf{W}_y]$	N	Extrap$[\mathbf{W}_y]$	N	Extrap$[\mathbf{W}_y]$	N

The chain structure of the update sequence

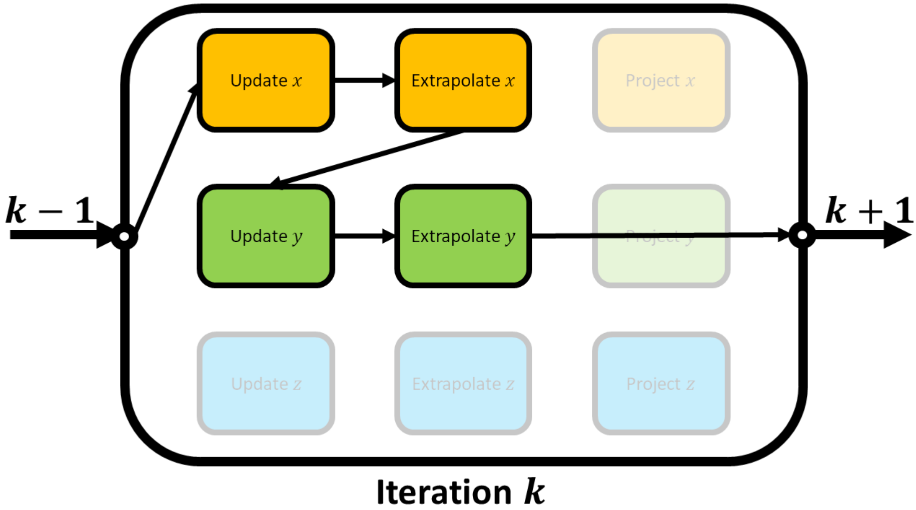
There are variations on the chain structure of the update, for examples

- Update $\mathbf{W}$ $\rightarrow$ extrapolate $\mathbf{W}$ $\rightarrow$ update $\mathbf{H}$ $\rightarrow$ extrapolate $\mathbf{H}$
- Update $\mathbf{W}$ $\rightarrow$ extrapolate $\mathbf{W}$ $\rightarrow$ update $\mathbf{H}$ $\rightarrow$ extrapolate $\mathbf{H}$ $\rightarrow$ project $\mathbf{H}$
- Update $\mathbf{W}$ $\rightarrow$ update $\mathbf{H}$ $\rightarrow$ extrapolate $\mathbf{W}$ $\rightarrow$ extrapolate $\mathbf{H}$

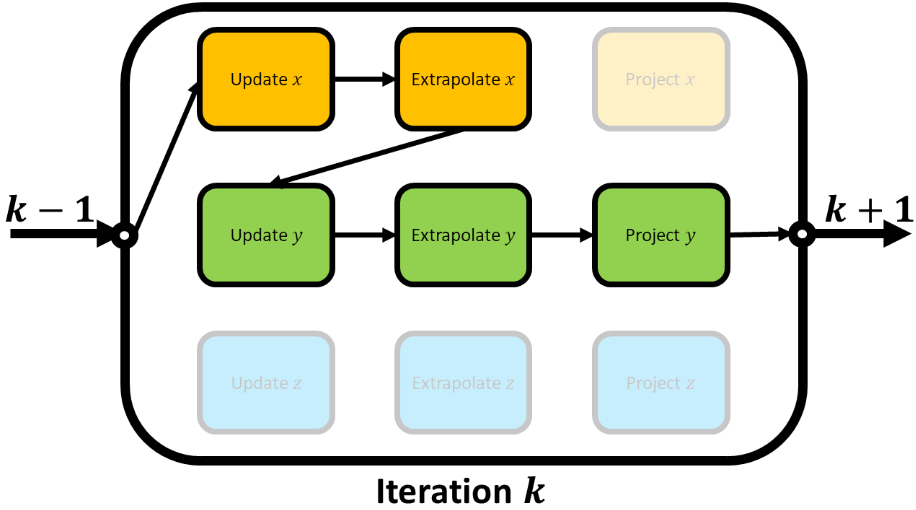
The comparisons of these three schemes : see the paper.



Update x
Extrapolate x
Project x
$k-1$
Update y
Extrapolate y
Project y
$k+1$
Update z
Extrapolate z
Project z
Iteration k



$k-1$
Update x
Extrapolate x
Project x
Update y
Extrapolate y
Project y
Update z
Extrapolate z
Project z
$k+1$
Iteration k



Open question : why certain structure has a better performance than others

Summary and notes (2/3)

Update using matrix with negative values :
Update[$\mathbf{H}_n$] w.r.t. $\mathbf{H}_n \geq 0$ with $(\mathbf{X}, \mathbf{W}_y, \mathbf{H}_n)$, using $\mathbf{H}_y$ as initial iterate
Update[$\mathbf{W}_n$] wr.t. $\mathbf{W}_n \geq 0$ with $(\mathbf{X}, \mathbf{W}_n, \mathbf{H}_y)$, using $\mathbf{W}_y$ as initial iterate

Summary and notes (3/3)

Restart using $e(k)$ as $\|\mathbf{X} - \mathbf{W}_n \mathbf{H}_y\|_F$ not $\|\mathbf{X} - \mathbf{W}_n \mathbf{H}_n\|_F$

Why :
(i) $\mathbf{W}_n$ was updated according to $\mathbf{H}_y$ (see point 2)

(ii) it gives the algorithm some degrees of freedom to possibly increase the objective function

(iii) computationally cheaper, as compute $\|\mathbf{X} - \mathbf{W}_n \mathbf{H}_n\|_F$ need $O(mnr)$ operations instead of $O(mr^2)$ by re-using previous computed terms :

$$\|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2 = \|\mathbf{X}\|_F^2 - 2\left\langle \mathbf{W}, \mathbf{X}\mathbf{H}^\top \right\rangle + \left\langle \mathbf{W}^\top \mathbf{W}, \mathbf{H}\mathbf{H}^\top \right\rangle$$

Note : if the variables converges, using $\mathbf{W}_n$, $\mathbf{W}_y$ is effectively the same as $\mathbf{W}_n^\infty = \mathbf{W}_y^\infty$ (after projection)

Experiments

Notations

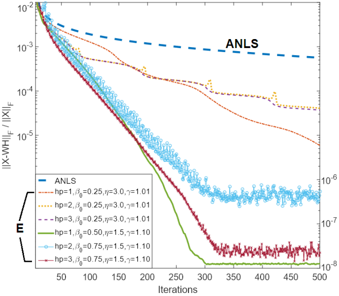
- A-HALS : vector-wise update, compute approximate solution
- ANLS : subproblem solved exactly using active-set methods
- E : extrapolation

Set up

- Average error over 10 trials
- $\mathbf{W}, \mathbf{H}, \mathbf{X}$ randomly generated $\sim \mathcal{U}[0,1]$, $m = n = 200$, $r = 20$
- Real $\mathbf{X}$ from real data is also used.
- Error comparisons : using lowest relative error e_{min} across all algorithms, at step k,

$$E(k) = \frac{\|\mathbf{X} - \mathbf{W}^k \mathbf{H}^k\|_F}{\|\mathbf{X}\|_F} - e_{\text{min}}$$

- It is possible $e_{\text{min}} = 0$ and not shown
- Extrapolation parmater $\beta_0 = [0.25, 0.5, 0.75]$
- $\eta_0 = [1.5, 2, 3]$
- $\gamma, \bar{\gamma} = [1.01, 1.005], [1.05, 1.01], [1.1, 1.05]$
- For display : only best and worst to illustrate sensitivity (for $k = 2$)



Low-rank synthetic data

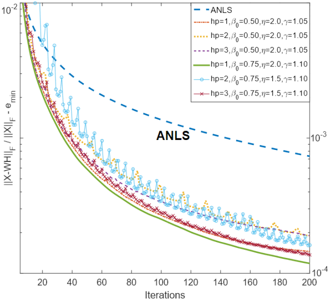


Image data

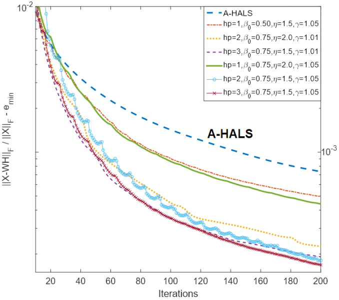
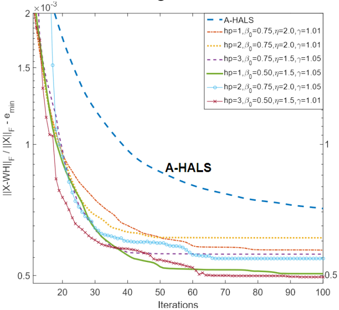


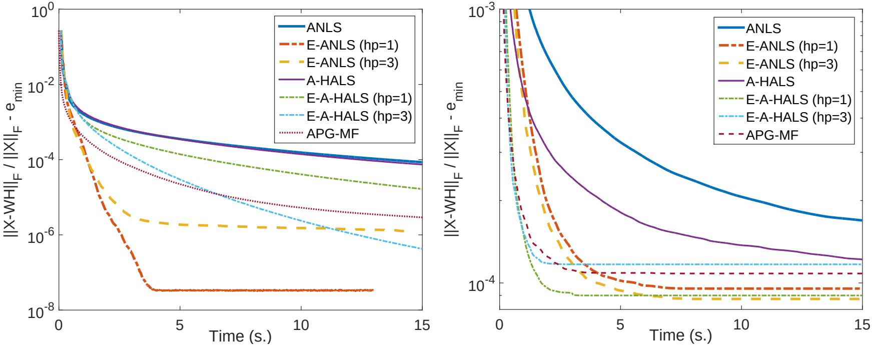
Image data



Text data

Fast conclusion : E wins.

Compare with other method on speed (time)



Average err. of ANLS, A-HALS and extrapolated variants, on low-rank (left) and full-rank (right) synthetic data.
APG-MF† = an extrapolated proximal type algorithm, with convergence proof.

Fast conclusion : E wins and beats APG-MF†.

† Xu-Yin 2013 "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion". SIAM J. Img Sci.

Overall results : E wins!

Method	Data	Ex wins?
A-HALS	Low/full rank synthetic data	YES
	Dense Image data†	YES
	Sparse text data$^\#$	YES
ANLS	Low/full rank synthetic data	YES
	Dense Image data†	YES
	Sparse text data$^\#$	YES

† ORL, Umist, CBCL, Frey.

$^\#$ Zhong-Ghosh 2005. Generative model-based document clustering: a comparative study

Conclusions

- No matter what method XXX, E-XXX $>$ XXX.
- E-XXX $>$ APG-MF (an extrapolated proximal-type method).
- Between E-ANLS vs E-A-HALS : no clear winner
  - Low rank synthetic data : E-ANLS $\gg$ everything
  - Dense data : E-A-HALS $\approx$ E-ANLS, although A-HALS $>$ ANLS
  - Sparse data : E-A-HALS $\gg$ everything
- Between different hp
  - Up-Ex-Up-Ex ($hp = 2$) seems worst
  - Up-Up-Ex-Ex ($hp = 1$) or Up-Ex-Pro-Up-Ex ($hp = 3$) are better

Don't trust me ? Go https://arxiv.org/abs/1805.06604, try the code!

Outline

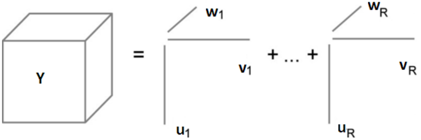
1. Introduction - Non-negative Matrix Factorization
2. Computing NMF
   - Variations on BCD
   - A-HALS
   - Matrix-wise Projected Gradient Update and the Multiplicative update
3. Find $(\mathbf{W}, \mathbf{H})$ numerically fast : acceleration via extrapolation
   - Recall : acceleration in single variable problem
   - Accelerating NMF algorithms using extrapolation
4. Computing NTF
5. Computing NNLS

Tensor extension

(Joint-work with Jeremy E. Cohen of IRISA, Rennes, France)

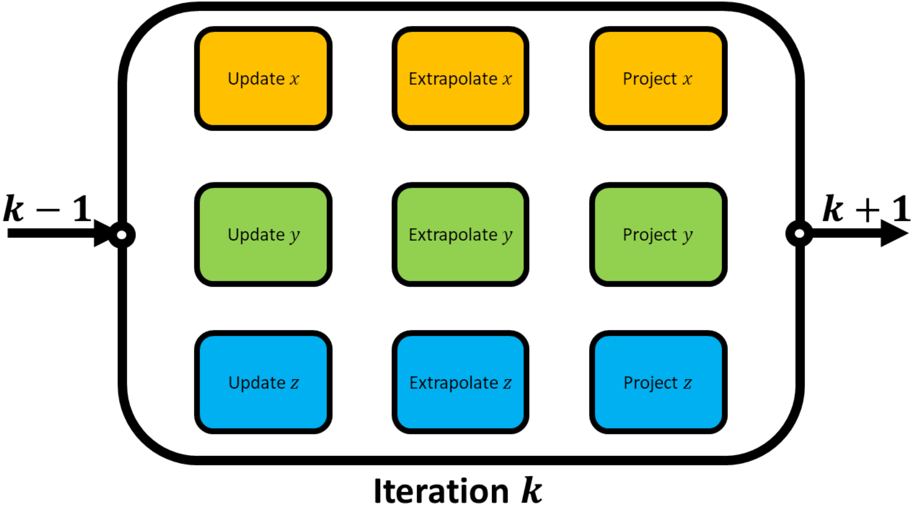
Extend the idea of extrapolation to the tensor cases; more precisely to the **Non-negative Canonical Polyadic Decomposition** (NNCPD).

$$\begin{aligned}\min_{\mathbf{U},\mathbf{V},\mathbf{W}} \Phi(\mathbf{U},\mathbf{V},\mathbf{W}) &= \|\mathbf{Y} - \mathbf{U} * \mathbf{V} * \mathbf{W}\| \text{ s.t. } \mathbf{U} \geq 0, \mathbf{V} \geq 0, \mathbf{W} \geq 0 \\ &= \|\mathbf{Y} - \sum_i^r \mathbf{u}_i * \mathbf{v}_i * \mathbf{w}_i\|\end{aligned}$$



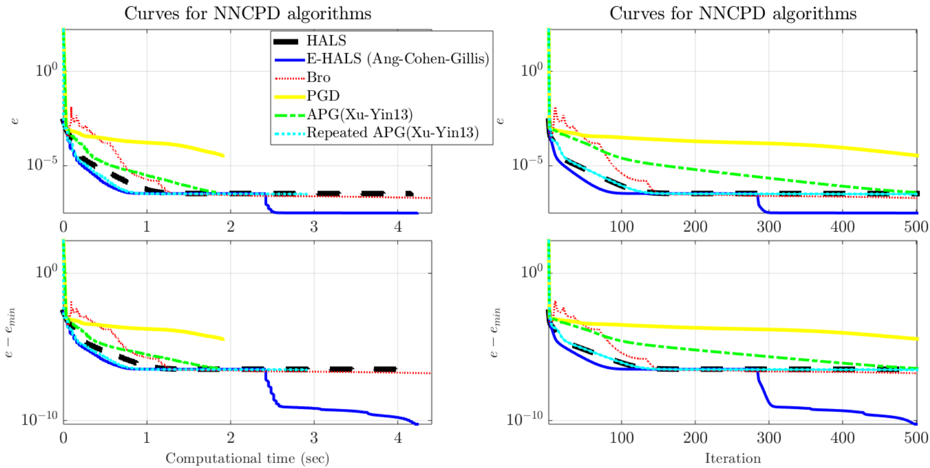
Experiments showed that the approach is very promising and is able to significantly accelerate the NNCPD algorithms.

Unsolved problem : NNCPD has even higher variability on the chain structure.



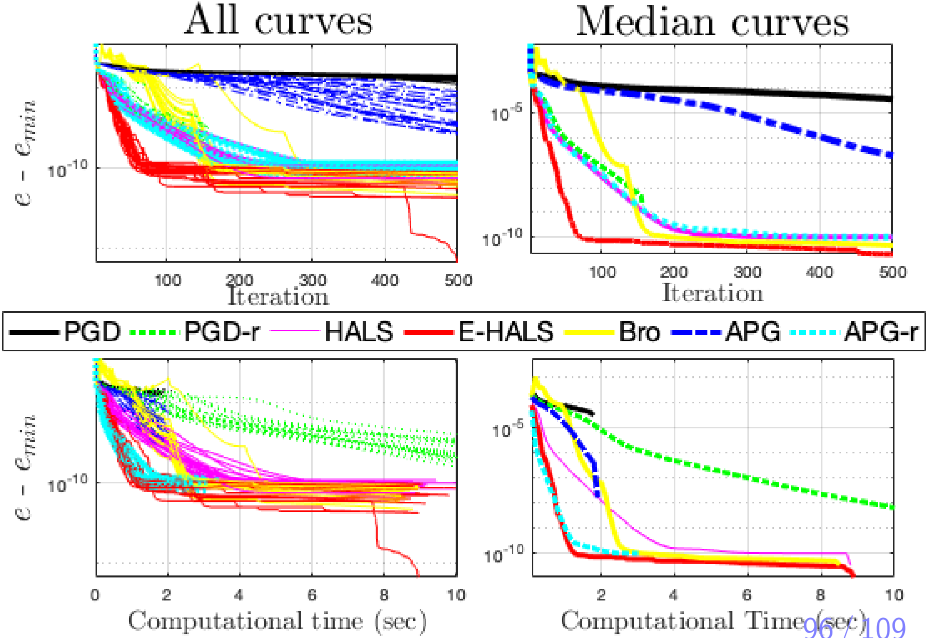
Understanding the relationship between the data structure (rank size, size of each mode) and the chain structure will be crucial.

Results : Toy example

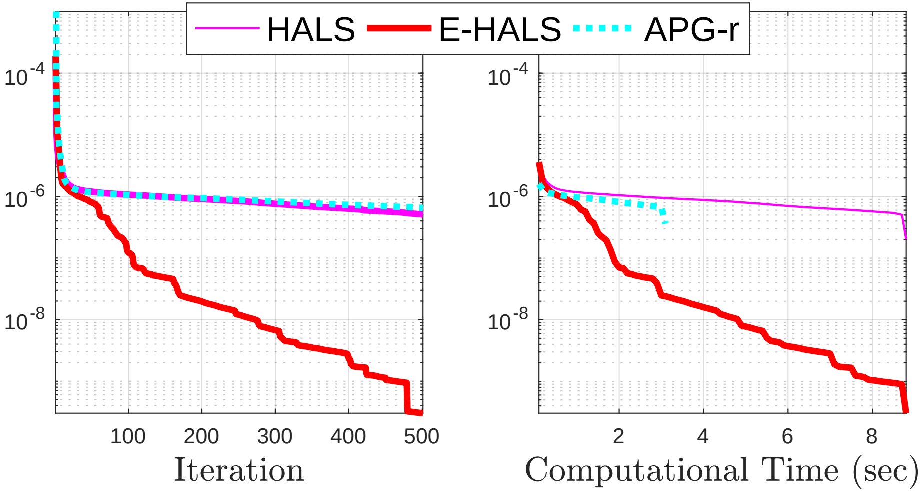


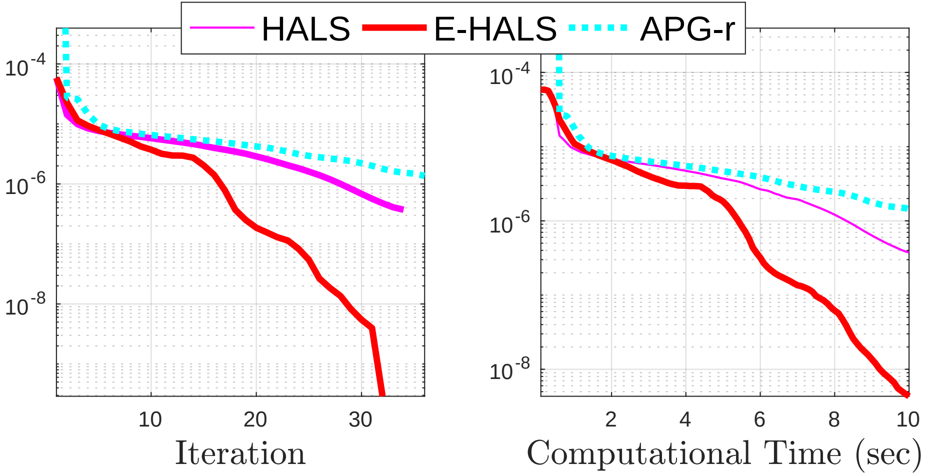
What about MU : too slow, not qualified.

Results : low rank, balanced sizes



Results : low rank, balanced sizes, ill-conditioned




HALS
E-HALS
APG-r
10^-4
10^-6
10^-8
10
20
30
Iteration
10^-4
10^-6
10^-8
2
4
6
8
10
Computational Time (sec)

On fitting the ground truth

TABLE 1 – Median $\mathrm{RE}_{\text{final}}$ in % of U, V, W, (* means ≥ 40)

Algo	Test 1	Test 2	Test 3
PGD	15, 1.8, 1.8	4.3, *, *	*, *, *
PGD-r	0.2, 1.8, 1.6	4.1, *, *	*, *, *
HALS	0.2, 1.6, 1.6	2.2, 22, 23	4.7, 5.2, 5.2
E-HALS	0.2, 1.8, 1.8	0.04,0.3, 0.3	0.4,0.8,0.8
Bro	0.2, 1.5, 1.5	0.2, 2.4, 2.4	*, *, *
APG	0.5, 3.0, 2.9	4.3, *, *	*, *, *
APG-r	0.2, 1.3, 1.2	2.7, *, 28	10,11,11

1. Introduction - Non-negative Matrix Factorization
2. Computing NMF
   - Variations on BCD
   - A-HALS
   - Matrix-wise Projected Gradient Update and the Multiplicative update
3. Find $(\mathbf{W}, \mathbf{H})$ numerically fast : acceleration via extrapolation
   - Recall : acceleration in single variable problem
   - Accelerating NMF algorithms using extrapolation
4. Computing NTF
5. Computing NNLS

Problem $(\mathcal{P})$: given $(\mathbf{A}, \mathbf{b})$, solve

$$\text{(NNLS) } \mathbf{x} = \operatorname*{argmin}_{\mathbf{x} \geq \mathbf{0}} \Phi(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2.$$

Let $\mathbf{Q} = \mathbf{A}^\top \mathbf{A}$, $\mathbf{p} = \mathbf{A}^\top \mathbf{b}$, we have an equivalent expression

$$\mathbf{x} = \operatorname*{argmin}_{\mathbf{x} \geq \mathbf{0}} \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} - \mathbf{p}^\top \mathbf{x} + c$$

Φ is $\|\mathbf{Q}\|_2$-smooth : the Lipschitz constant of $\nabla\Phi$ is $\|\mathbf{Q}\|_2$

PGD update : $\mathbf{x}^+ = \mathbf{x} - t(\mathbf{Q}\mathbf{x} - \mathbf{p})$ with $t = L^{-1}$

Multiplicative update

Using the component-wise step size $t_i = \dfrac{x_i}{[\mathbf{Qx}]_i}$, the vector update $\mathbf{x}^+ = \mathbf{x} - t(\mathbf{Qx} - \mathbf{p})$ becomes

$$
\begin{aligned}
x_i^+ &= x_i - t_i([\mathbf{Qx}]_i - p_i) \\
&= x_i - \frac{x_i}{[\mathbf{Qx}]_i}([\mathbf{Qx}]_i - p_i) \\
&= \frac{[\mathbf{Qx}]_i}{[\mathbf{Qx}]_i} x_i - \frac{[\mathbf{Qx}]_i - p_i}{[\mathbf{Qx}]_i} x_i \\
&= \frac{p_i x_i}{[\mathbf{Qx}]_i}
\end{aligned}
$$

In vector form, we have

$$\mathbf{x}^+ = \mathbf{x} \otimes \frac{\mathbf{p}}{\mathbf{Qx}},$$

where the multiplication $\otimes$ and division $\frac{[\,]}{[\,]}$ are element-wise.

As $\mathbf{p}$, $\mathbf{Q}$ and $\mathbf{x}_0$ are all non-negative, thus the iteration produce a non-negative output.

Solving NNLS by MU algorithm

Problem :

$$\mathbf{x}_{\mathsf{NNLS}} := \text{argmin}_{\mathbf{x}\geq 0} f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

The Multiplicative Update algorithm for NNLS

Algorithm MU for NNLS

Input: $\mathbf{A} \in \mathbb{R}_+^{m\times n}$, $b \in \mathbb{R}^m$, an initialization $\mathbf{x} \in \mathbb{R}_+^n$
Output: $\mathbf{x}$

1: **for** $k = 1, 2, \ldots$ **do**
2: $\quad \mathbf{x}_{k+1} = \mathbf{x}_k \otimes \dfrac{\mathbf{p}}{\mathbf{Q}\mathbf{x}_k}$
3: **end for**

It can be proved that, the objective function $f(\mathbf{x})$ is non-increasing under MU iteration $\mathbf{x}_{k+1} = \mathbf{x}_k \otimes \dfrac{\mathbf{p}}{\mathbf{Q}\mathbf{x}_k}$.

Solving NNLS by PGD algorithm

Problem :

$$\mathbf{x}_{\text{NNLS}} := \text{argmin}_{\mathbf{x}\geq 0} f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

The PGD algorithm for NNLS

Algorithm PGD for NNLS

Input: $\mathbf{A} \in \mathbb{R}_+^{m\times n}$, $b \in \mathbb{R}^m$, an initialization $\mathbf{x} \in \mathbb{R}_+^n$
Output: $\mathbf{x}$

1: **for** $k = 1, 2, \ldots$ **do**
2: $\quad \mathbf{x}_{k+1} = \left[\mathbf{x}_k - \frac{1}{L}(\mathbf{Q}\mathbf{x}_k - \mathbf{p})\right]_+$
3: **end for**

It can be proved that, the objective function $f(\mathbf{x})$ is strictly decreasing under PGD iteration when sufficient descent condition holds.

Problem :

$$\mathbf{x}_{\mathsf{NNLS}} := \operatorname{argmin}_{\mathbf{x}\geq 0} f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2$$

The A-PGD algorithm for NNLS

Algorithm A-PGD for NNLS

Input: $\mathbf{A} \in \mathbb{R}_+^{m\times n}$, $b \in \mathbb{R}^m$, an initialization $\mathbf{x} \in \mathbb{R}_+^n$
Output: $\mathbf{x}$

1: **for** $k = 1, 2, \ldots$ **do**
2: Compute β_k
3: $\mathbf{y}_{k+1} = \left[\mathbf{x}_k - \frac{1}{L}(\mathbf{Q}\mathbf{x}_k - \mathbf{p})\right]_+$
4: $\mathbf{x}_{k+1} = \mathbf{y}_{k+1} + \beta_k(\mathbf{y}_{k+1} - \mathbf{y}_k)$
5: **end for**

Solving NNLS by Accelerated-PGD algorithm, with restart

Problem :

$$\mathbf{x}_{\mathsf{NNLS}} := \operatorname{argmin}_{\mathbf{x}\geq 0} f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2$$

The A-PGD algorithm for NNLS

Algorithm A-PGD for NNLS

Input: $\mathbf{A} \in \mathbb{R}_+^{m\times n}$, $b \in \mathbb{R}^m$, an initialization $\mathbf{x} \in \mathbb{R}_+^n$

Output: $\mathbf{x}$

for $k = 1, 2, \dots$ **do**

  Compute β_k

  $\mathbf{y}_{k+1} = \left[\mathbf{x}_k - \frac{1}{L}(\mathbf{Q}\mathbf{x}_k - \mathbf{p})\right]_+$

  $\mathbf{x}_{k+1} = \mathbf{y}_{k+1} + \beta_k(\mathbf{y}_{k+1} - \mathbf{y}_k)$

  IF error increase **do**

    $\mathbf{x}_{k+1} = \mathbf{y}_{k+1}$ (take no extrapolation)

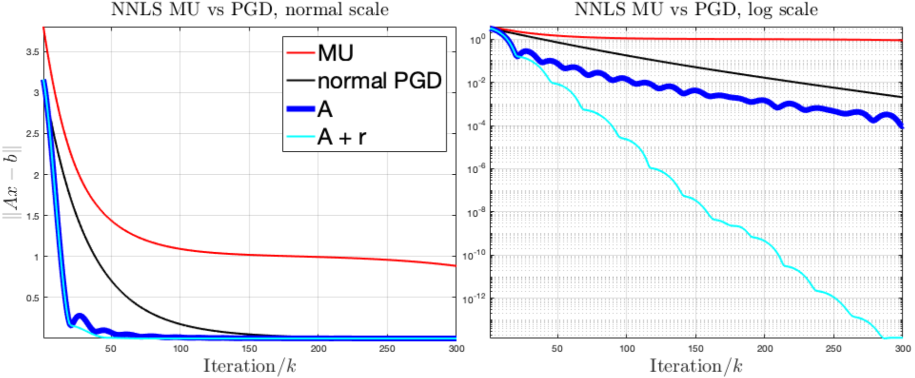
    reset β

  ENDIF

end for

Toy example

PGD without any acceleration is already much faster than MU. Not to mention those with acceleration and restart



$$(m,n) = 100, 10.$$

What about my scheme? : With a "good" parameter, the scheme is even faster than Nesterov's type acceleration algorithm. However, all of them are still in linear convergence rate.

You sure want to read it ?
(show the long proof)

Last page – summary

- What is Non-negative Matrix Factorization, Why NMF
- How to solve NMF *fast* with extrapolation

  A.-Gillis, "Accelerating Non-negative matrix factorization by extrapolation", *Neural Computation*, Feb, 2019.
- How to solve NTF *fast* with extrapolation

  A.-Cohen-Gillis, "Accelerating Approximate Nonnegative Canonical Polyadic Decomposition using Extrapolation", 2019.
- How to solve NNLS *fast* with extrapolation

  work in progress
- Some open problems

END OF PRESENTATION.

Slide, code, preprint in *angms.science*