Regularized Nonlinear Acceleration

Damien Scieur, Alexandre d'Aspremont, Francis Bach

Acknowledgements

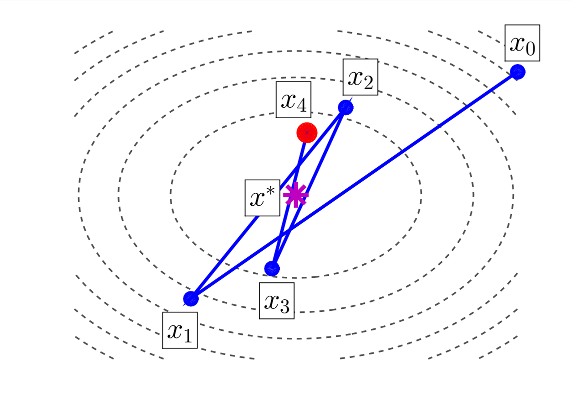
SpaRTaN

Sparse Representations and Compressed
Sensing Training Network

INVENTEURS DU MONDE NUMÉRIQUE

cnrs dépasser les frontières

Algorithms produce a sequence of iterates. We usually keep the last/best one, or their mean.



Can we do better?

Idea of Extrapolation

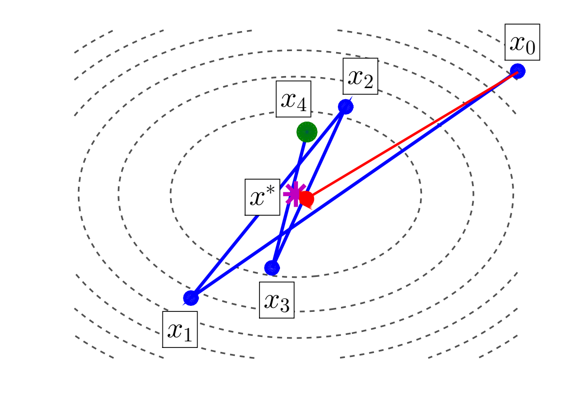
1. Run a simple algorithm, e.g. gradient descent

Idea of Extrapolation

1. Run a simple algorithm, e.g. gradient descent
2. “Guess” the solution using an extrapolation algorithm

Idea of Extrapolation

1. Run a simple algorithm, e.g. gradient descent
2. "Guess" the solution using an extrapolation algorithm
3. Enjoy! ☺



Extrapolation in Real Life

Extrapolation in Real Life

An infinite number of mathematicians walk into a bar...

Extrapolation in Real Life

An infinite number of mathematicians walk into a bar...

The first one orders a beer,
the second one orders half a beer,
the third one orders a fourth of a beer, ...

Extrapolation in Real Life

An infinite number of mathematicians walk into a bar...

The first one orders a beer,
the second one orders half a beer,
the third one orders a fourth of a beer, ...

The bartender stops them, pours two beers and says “You guys should know your limits!” - **Extrapolation step!**

Example in One Dimension

An autoregressive process generates scalars x_t, converging to x^*:

$$x_{t+1} = ax_t + b \qquad \Leftrightarrow \qquad (x_{t+1} - x^*) = \alpha(x_t - x^*)$$

Example: the "beer" series $\left\{1\;;1+\frac{1}{2}\;;1+\frac{1}{2}+\frac{1}{4}\;;\ \ldots\right\}$

$$2 - x_{t+1} = \frac{1}{2}(2 - x_t) \quad \Rightarrow \quad \begin{cases} \alpha & = 1/2 \\ x^* & = 2 \end{cases}$$

Like the bartender, we can estimate x^* using only **three** iterates

Aitken's Δ^2 Formula

Three points $\{x_{t-1}, x_t, x_{t+1}\}$ from autoregressive process:

$$(x_{t+1} - x^*) = \alpha(x_t - x^*)$$

Aitken's Δ^2 Formula

Three points $\{x_{t-1}, x_t, x_{t+1}\}$ from autoregressive process:

$$(x_{t+1} - x^*) = \alpha(x_t - x^*)$$

1) Estimate α: Take the difference between two successive x_t,

$$\left.\begin{aligned}(x_{t+1} - x^*) &= \alpha(x_t - x^*) \\ (x_t - x^*) &= \alpha(x_{t-1} - x^*)\end{aligned}\right\} \quad \Rightarrow \quad (x_{t+1} - x_t) = \alpha(x_t - x_{t-1})$$

Simple estimation: $\alpha_{est} = \frac{(x_{t+1} - x_t)}{(x_t - x_{t-1})}$.

Aitken's Δ^2 Formula

Three points $\{x_{t-1}, x_t, x_{t+1}\}$ from autoregressive process:

$$(x_{t+1} - x^*) = \alpha(x_t - x^*)$$

1) Estimate α: Take the difference between two successive x_t,

$$\left.\begin{aligned} (x_{t+1} - x^*) &= \alpha(x_t - x^*) \\ (x_t - x^*) &= \alpha(x_{t-1} - x^*) \end{aligned}\right\} \quad \Rightarrow \quad (x_{t+1} - x_t) = \alpha(x_t - x_{t-1})$$

Simple estimation: $\alpha_{est} = \frac{(x_{t+1} - x_t)}{(x_t - x_{t-1})}$.

2) Extrapolate x^*:

$$(x_{t+1} - x^*) = \alpha_{est}(x_t - x^*) \qquad \Rightarrow \quad x_{extr} = \frac{x_{t+1} - \alpha_{est} x_t}{1 - \alpha_{est}}$$

Require only three iterates!

Examples

When applied to the beer series,

$$\left\{1,\, 1+\frac{1}{2},\, 1+\frac{1}{2}+\frac{1}{4},\, \ldots\right\},$$

recover **exactly** $x_{extr} = 2 = x^*$ using only three iterates!

Examples

When applied to the beer series,

$$\left\{1,\, 1+\frac{1}{2},\, 1+\frac{1}{2}+\frac{1}{4},\, \dots\right\},$$

recover **exactly** $x_{extr} = 2 = x^*$ using only three iterates!

Performs well on other kind of sequences, e.g.

$$x_t = \sum_{k=0}^{t} \frac{(-1)^k}{2k+1} \;\rightarrow\; x^* = \frac{\pi}{4}.$$

Without extrapolation: $x_9 - x^* \approx 0.03$.
Using Δ^2 formula on x_7, x_8, x_9: $x_{extr} - x^* \approx 0.00001$!

Extension to $\mathbb{R}^d$

$$\text{Scalar autoregressive process} \quad \longrightarrow \quad \text{Vector autoregressive process}$$

$$(x_{t+1} - x^*) = \alpha(x_t - x^*) \quad \longrightarrow \quad (x_{t+1} - x^*) = A(x_t - x^*)$$

We assume A symmetric.

Extension to $\mathbb{R}^d$

$$\text{Scalar autoregressive process} \quad \longrightarrow \quad \text{Vector autoregressive process}$$

$$(x_{t+1} - x^*) = \alpha(x_t - x^*) \quad \longrightarrow \quad (x_{t+1} - x^*) = A(x_t - x^*)$$

We assume A symmetric.

Example: **Gradient method** with step size h,

$$(x_{t+1} - x^*) = (x_t - x^*) - h\nabla f(x_t)$$

Linearizing $\nabla f(x)$ around x^* gives

$$\boxed{(x_{t+1} - x^*) = \underbrace{\left(I - h\nabla^2 f(x^*)\right)}_{=A}(x_t - x^*)} + \textit{Perturbations}$$

Outline

1. Extension of Δ^2 formula in $\mathbb{R}^d$? **Anderson acceleration**
2. Performances? **Optimal on quadratics**
3. Impact of perturbations? **Huge** $\rightarrow$ **Regularization** (our work)
4. Rate of convergence? **Asymptotically optimal**

Acceleration and Weighted Average

Vector autoregressive process with $\|A\| = (1 - \kappa) < 1$,

$$(x_{t+1} - x^*) = A(x_t - x^*) = A^{t+1}(x_0 - x^*)$$

Error at iteration k:

$$\|x_k - x^*\| \leq (1 - \kappa)^k \|x_0 - x^*\| \qquad \textit{(Slow)}$$

We *literally* waste information contained in $x_0, ..., x_{k-1}$!

Acceleration and Weighted Average

Vector autoregressive process with $\|A\| = (1-\kappa) < 1$,

$$(x_{t+1} - x^*) = A(x_t - x^*) = A^{t+1}(x_0 - x^*)$$

Error at iteration k:

$$\|x_k - x^*\| \leq (1-\kappa)^k \|x_0 - x^*\| \qquad \textit{(Slow)}$$

We *literally* waste information contained in $x_0, \ldots, x_{k-1}$!

Proposition

There exists vector $c \in \mathbb{R}^k$ s.t. $\sum_{i=0}^k c_i = 1$ and

$$\|\sum_{i=0}^{k} c_i x_i - x^*\| \leq (1-\sqrt{\kappa})^k \|x_0 - x^*\| \qquad \text{(Optimal)}$$

Proof: There exist accelerated methods (e.g. Nesterov)

Goal of extrapolation

Find the best coefficients c such that

$$\|\sum_{i=0}^{k} c_i x_i - x^*\| = \|x_{extr} - x^*\|$$

is as small as possible.

Approximation of Extrapolation Error

Goal: Approximate (then minimize) the extrapolation error

$$x_{extr} - x^* = \sum_{i=0}^{k} c_i x_i - x^* \quad \text{(unknown)}$$

Approximation of Extrapolation Error

Goal: Approximate (then minimize) the extrapolation error

$$x_{extr} - x^* = \sum_{i=0}^{k} c_i x_i - x^* \quad \text{(unknown)}$$

Definition: Residual at iteration i, $r_i \triangleq x_{i+1} - x_i$ (known)

Approximation of Extrapolation Error

Goal: Approximate (then minimize) the extrapolation error

$$x_{extr} - x^* = \sum_{i=0}^{k} c_i x_i - x^* \quad \text{(unknown)}$$

Definition: Residual at iteration i, $\quad r_i \triangleq x_{i+1} - x_i$ (known)

Proposition

The combination of residuals approximates the extrapolation error

$$\sum_{i=0}^{k} c_i r_i = (A - I)(x_{extr} - x^*)$$

Proof:

$$\begin{aligned} x_{i+1} - x_i &= (x_{i+1} - x^*) - (x_i - x^*) \\ &= A(x_i - x^*) - (x_i - x^*) = (A - I)(x_i - x^*) \end{aligned}$$

Acceleration Algorithm (Anderson's Acceleration)

1. Compute the residuals $r_i = x_{i+1} - x_i, \quad i = 1...k$

Acceleration Algorithm (Anderson's Acceleration)

1. Compute the residuals $r_i = x_{i+1} - x_i, \quad i = 1...k$
2. Under the constraint $\sum c_i = 1$, solve

$$\min_c \| \sum_{i=0}^{k} c_i r_i \| \;\approx\; \min_c \| x_{extr} - x^* \|$$

Acceleration Algorithm (Anderson's Acceleration)

1. Compute the residuals $r_i = x_{i+1} - x_i, \quad i = 1...k$
2. Under the constraint $\sum c_i = 1$, solve

$$\min_c \| \sum_{i=0}^{k} c_i r_i \| \;\approx\; \min_c \|x_{extr} - x^*\|$$

2'. Let $R = [r_0, r_1, ..., r_k]$. The closed-form formula is

$$c^* = \frac{(R^T R)^{-1}\mathbf{1}}{\mathbf{1}^T (R^T R)^{-1}\mathbf{1}}$$

Acceleration Algorithm (Anderson's Acceleration)

1. Compute the residuals $r_i = x_{i+1} - x_i, \quad i = 1...k$
2. Under the constraint $\sum c_i = 1$, solve

$$\min_c \| \sum_{i=0}^{k} c_i r_i \| \;\; \approx \;\; \min_c \| x_{extr} - x^* \|$$

2'. Let $R = [r_0, r_1, ..., r_k]$. The closed-form formula is

$$c^* = \frac{(R^T R)^{-1} \mathbf{1}}{\mathbf{1}^T (R^T R)^{-1} \mathbf{1}}$$

3. Return

$$x_{extr} = \textstyle\sum_{i=0}^{k} c_i^* x_i \quad \approx x^*$$

Performances

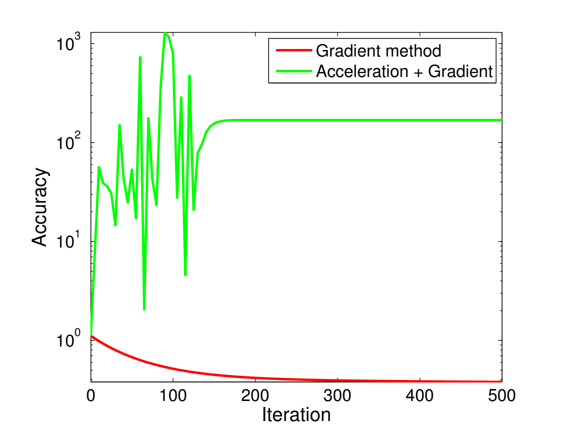
Convergence rate for minimizing quadratics:

$$\|x_{extr} - x^*\| \leq (1 - \sqrt{\kappa})^k \|x_0 - x^*\| \qquad \text{(Optimal)}$$

Performances

Convergence rate for minimizing quadratics:

$$\|x_{extr} - x^*\| \leq (1 - \sqrt{\kappa})^k \|x_0 - x^*\| \qquad \text{(Optimal)}$$



Does not work for minimizing nonlinear functions...

Reminder

Gradient method with step size h,

$$(x_{t+1} - x^*) = (x_t - x^*) - h\nabla f(x_t)$$

Linearizing $\nabla f(x)$ around x^* gives

$$\boxed{(x_{t+1} - x^*) = A(x_t - x^*)} + \textit{Perturbations}$$

Anderson's Acceleration is unstable! - why?

Impact of perturbations?

The computation of c^* involves $(R^T R)^{-1}$

Proposition

If $R^T R$ is perturbed by a matrix P (e.g. Taylor remainder), then

$$\text{Error on } c^* \leq \|(R^T R)^{-1}\| \|P\| \|c^*\|$$

Impact of perturbations?

The computation of c^* involves $(R^T R)^{-1}$

Proposition

If $R^T R$ is perturbed by a matrix P (e.g. Taylor remainder), then

$$\text{Error on } c^* \leq \|(R^T R)^{-1}\| \|P\| \|c^*\|$$

Bad conditionning (Tyrtyshnikov, 1994)

Krylov matrix. $\|(R^T R)^{-1}\|$ grows exponentially with k.

The error on c^* is virtually **unbounded**.

Regularized Nonlinear Acceleration (RNA)

Perturbations are controlled by **Tikhonov Regularization**

Input: Sequence $\{x_0, ..., x_{k+1}\}$, parameter $\lambda > 0$

1: Form $R = [r_0, \ ..., \ r_k]$, where $r_i = x_{i+1} - x_i$ — $O(dk)$

2: Compute $R^T R$ — $O(dk^2)$

3: Compute $c^* = \frac{(R^T R + \lambda \mathbf{I})^{-1} \mathbf{1}}{\mathbf{1}^T (R^T R + \lambda \mathbf{I})^{-1} \mathbf{1}}$ — $O(k^3)$

Output: Return $x_{extr} = \sum_{i=0}^{k} c_i^* x_i \approx x^*$

Paper: *Regularized Nonlinear Acceleration* (NIPS 2016)

Performances of RNA

Algorithmic complexity. In practice, $k \ll d$. Complexity is $O(d)$!

Sparse input. Complexity $O(k^2 s)$. Sparse output: $\|x_{extr}\|_0 \leq ks$.

Matlab/Python complexity. Only 5 lines of code!

Theorem (Scieur, d'Aspremont and Bach, 2016)

Asymptotic Acceleration Let $\|x_0 - x^*\| \to 0$ and λ well chosen,

$$\|x_{extr} - x^*\| \leq O\left((1-\sqrt{\kappa})^k \|x_0 - x^*\|\right) \qquad \text{(Optimal)}$$

(Non-asymptotic bounds hold as well)

The gradient method on smooth and strongly convex functions meets the assumptions

Practical Usage (from most to less important)

A. Restart Strategy

1. Generate $\{x_0, ..., x_{k+1}\}$ using your Algorithm, starting at x_0
2. Let $x_{extr} = \textbf{RNA}(\{x_0, ..., x_{k+1}\}, \lambda)$ and restart with $x_0 = x_{extr}$

Practical Usage (from most to less important)

A. Restart Strategy

1. Generate $\{x_0, ..., x_{k+1}\}$ using your Algorithm, starting at x_0
2. Let $x_{extr} = \textbf{RNA}(\{x_0, ..., x_{k+1}\}, \lambda)$ and restart with $x_0 = x_{extr}$

B. Grid search (as expensive as backtracking line-search)

Choose several λ_j and compute x_{extr}^j, then choose the best

$\Rightarrow$ Makes the algorithm **parameter-free**!

Practical Usage (from most to less important)

A. Restart Strategy

1. Generate $\{x_0, ..., x_{k+1}\}$ using your Algorithm, starting at x_0
2. Let $x_{extr} = \textbf{RNA}(\{x_0, ..., x_{k+1}\}, \lambda)$ and restart with $x_0 = x_{extr}$

B. Grid search (as expensive as backtracking line-search)

Choose several λ_j and compute x_{extr}^j, then choose the best

$\Rightarrow$ Makes the algorithm **parameter-free**!

C. Line-search

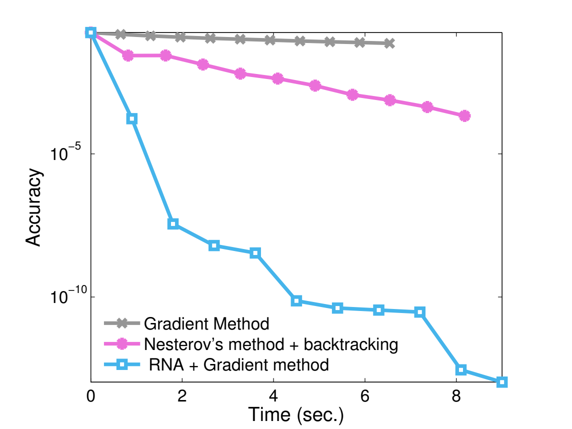
Solve approximatively

$$\min_h f\left(x_0 + h(x_{extr} - x_0)\right)$$

Numerical experiment: Logistic regression

Dataset: Madelon (2000 data points, 500 features, $\kappa = 10^{-6}$),

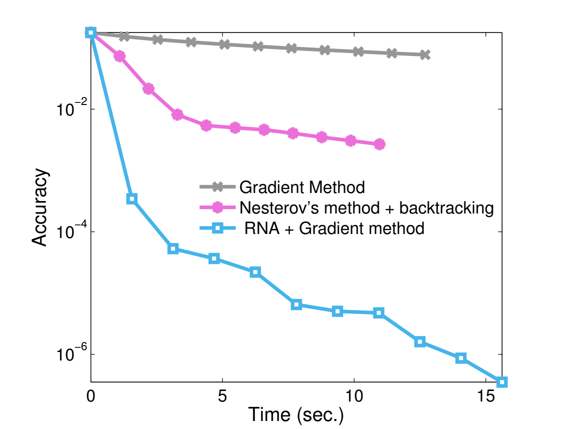
$$f(w) = \tau\|w\|_2^2 + \sum_{i=1}^{N} \log(1 + \exp(y_i X_i^T w)).$$



Numerical experiment: Logistic regression

Dataset: Madelon (2000 data points, 500 features, $\kappa = 10^{-9}$),

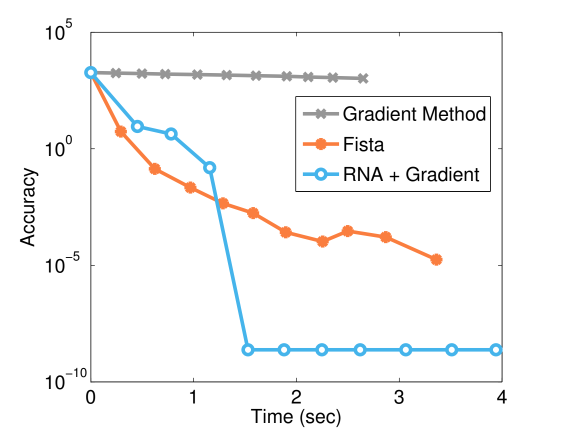
$$f(w) = \tau \|w\|_2^2 + \sum_{i=1}^{N} \log(1 + \exp(y_i X_i^T w)).$$



Numerical experiment: Dual SVM

Dataset: Madelon (2000 data points, 500 features),

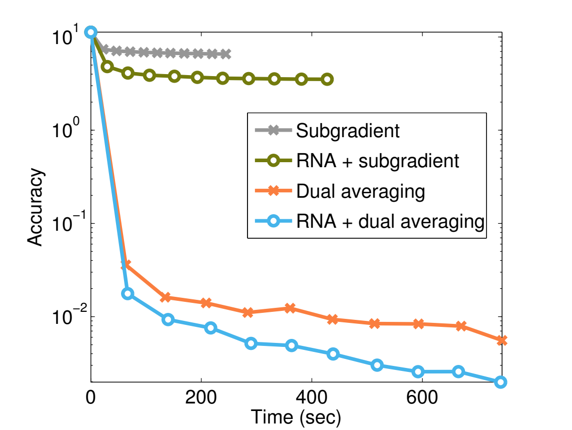
$$f(w) = \frac{1}{2}\|X^T \mathrm{diag}(y) w\|^2 - \mathbf{1}^T w, \quad 0 \le w \le 1.$$



Numerical experiment: Max-cut (Non-smooth optimization)

Dataset: Random graph (200 nodes, 2000 edges),

$$f(w) = \lambda_{\max}\Big(\text{Laplacian}(G) + \text{diag}(w)\Big) - \mathbf{1}^T w$$



Conclusion

- Simple, generic acceleration algorithm
- Highly adaptive
- Negligible additional computation cost
- Significant convergence speedup over optimal methods

Work in progress...

- Acceleration of accelerated methods?
- Proximal version?
- Non-smooth acceleration?

Thank you!