Gradient Descent

We will assume mothing about the convexity of f. We will show that gradient descent reaches an ϵ-substationary point x, such that $|\nabla f(x)|_2 \leq \epsilon$, in $O(1/\epsilon^2)$ iterations.

Lets write Lipschitz parabolic upper bound:

[f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2} |y - x|_2^2, \quad \text{for all } x, y. \quad (1)]

Lets plug in $y = x^{k + 1} = x ^ k - \alpha \nabla f(x^k), x = x^k$ to equation (1)

[f(x^{k + 1}) \leq f(x^k) + \nabla f(x^k) (-\alpha \nabla f(x^k)) + \frac{L}{2}\alpha^2 | \nabla f(x^k) |_2^2]

[f(x^{k+1}) \leq f(x^k) - \alpha | \nabla f(x^k) |_2^2 + \frac{\alpha^2L}{2}|\nabla f(x^k) |_2^2]

[f(x^{k+1}) \leq f(x^k) + \alpha | \nabla f(x^k) |_2^2(\frac{\alpha L}{2} - 1)]

Lets use $\alpha \leq 1/L$, and rearrange the previous result:

[\alpha (1 - \frac{L\alpha}{2}) | \nabla f(x^k) |_2^2 \leq f(x^k) - f(x^{k + 1})]

[\frac{\alpha}{2} | \nabla f(x^k) |_2^2 \leq \alpha (1 - \frac{L\alpha}{2}) | \nabla f(x^k) |_2^2 \leq f(x^k) - f(x^{k + 1})]

[| \nabla f(x^k) |_2^2 \leq \frac{2}{\alpha} (f(x^k) - f(x^{k + 1}))]

Lets sum the previous result over all iterations from $1,\ldots,k+1$:

[\sum_{i=0}^{k} | \nabla f(x^i) |_2^2 \leq \frac{2}{\alpha} (f(x^0) - f(x^1) + f(x^1) - f(x^2) + f(x^2) + ...) = \frac{2}{\alpha} (f(x^0) - f(x^*))]

Lets lower bound the sum in the previous result to get:

[k \cdot \min_{i=0, \dots, k} |\nabla f(x^i)|*2 \leq \sqrt{\sum*{i=0}^{k} | \nabla f(x^i) |_2^2} \leq \sqrt{\frac{2}{\alpha (k+1)} (f(x^0) - f^*)}]

[\min_{i=0, \dots, k} |\nabla f(x^i)|_2 \leq \sqrt{\frac{2}{\alpha (k+1)} (f(x^0) - f^*)}]

Accelerated methods

1) To solve this task, we need to analyze the **local convergence** of the **Heavy Ball Method** applied to the given function. The function (f(x)) is piecewise quadratic, meaning its gradient (\nabla f(x)) is piecewise linear. Since the method is known to perform well for strongly convex quadratics using the optimal hyperparameters:

[\alpha^* = \frac{4}{(\sqrt{L}+\sqrt{\mu})^2}, \quad \beta^* = \left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}\right)^2]

Given:

[f(x) = \begin{cases} \frac{25}{2}x^2, & \text{if } x < 1 \ \frac{1}{2}x^2 + 24x - 12, & \text{if } 1 \leq x < 2 \ \frac{25}{2}x^2 - 24x +36, & \text{if } x \geq 2 \end{cases}]

and its derivative:

[\nabla f(x) = \begin{cases} 25x, & \text{if } x < 1 \ x + 24, & \text{if } 1 \leq x < 2 \ 25x - 24, & \text{if } x \geq 2 \end{cases}]

Lets prove, that the given function is convex, strongly convex, smooth.

Lets prove μ-strong convexity and find coefficient.

μ-strong means, that $\forall x, y, \forall \lambda \in [0, 1]:$

[f(y) \geq f(x) + \nabla f(x) (y - x) + \frac{\mu}{2} | y - x |^2]

[f(\lambda x_1 + (1 - \lambda) x_2) \leq \lambda f(x_1) + (1 - \lambda) f(x_1) - \frac{\mu}{2}\lambda (1-\lambda)| x_1 - x_2 |^2]

for function $g(x) = ax^2 + bx + c$: $\mu = 2a \Rightarrow$ for $f(x) ; \mu=2 \cdot min(\frac{25}{2}, \frac{1}{2}, \frac{25}{2}) = 2 \cdot \frac{1}{2} = 1$

Lets count L, that will prove smoothness of function $f = f(x)$

[\forall x, y: ; | \nabla f(x) - \nabla f(y) | \leq L| x - y |]

[L = max(\nabla^2f) = max(25, 1, 25) = 25]

Now, we are plotting the function value for $x \in [-4, 4]$ (look in the .ipynb report)

[\nabla f(x) = \frac{1}{m} \sum_{i=1}^{m} -b_i (1 - \sigma(b_i \langle a_i, x \rangle)) a_i + \lambda x]

[\nabla f(x) = -\frac{1}{m} \sum_{i=1}^{m} b_i (1 - \sigma(b_i \langle a_i, x \rangle)) a_i + \lambda x]

2.

Lets assume classification task:

Logistic regression is a standard model in classification tasks. For simplicity, consider only the case of binary classification. Informally, the problem is formulated as follows: There is a training sample ${(a_i, b_i)}*{i=1}^m$, consisting of m vectors $a_i \in \mathbb{R}^n$ (referred to as features) and corresponding numbers $b_i \in {-1, 1}$ (referred to as classes or labels). The goal is to construct an algorithm $b(\cdot)$, which for any new feature vector a automatically determines its class $b(a) \in {-1, 1}$. In the logistic regression model, the class determination is performed based on the sign of the linear combination of the components of the vector a with some fixed coefficients $x \in \mathbb{R}^n$: $$ b(a) := \text{sign}(\langle a, x \rangle). $$ The coefficients x are the parameters of the model and are adjusted by solving the following optimization problem: $$ \tag{LogReg} \min*{x \in \mathbb{R}^n} \left(\frac{1}{m} \sum_{i=1}^m \ln(1 + \exp(-b_i \langle a_i, x \rangle)) + \frac{\lambda}{2} |x|^2 \right), $$ where $\lambda \geq 0$ is the regularization coefficient (a model parameter).

Lets the LogReg problem be convex for $\lambda = 0$. What is the gradient of the objective function? Will it be strongly convex? What if you will add regularization with $\lambda > 0$?

[f(x) = \frac{1}{m} \sum_{i=1}^m \ln(1 + \exp(-b_i \langle a_i, x \rangle)) + \frac{\lambda}{2} |x|^2]

$\forall x_1, x_2, \forall t$

[\frac{1}{m}\sum_{i=1}^{m}\ln(1 + \exp(-b_i(t\langle a_i, x_1\rangle + (1 - t)\langle a_i, x_2 \rangle))) \leq]

[\leq \frac{1}{m}\sum_{i=1}^{m}t\ln(1 + \exp(-b_i\langle a_i, x_1 \rangle)) + (1-t)\ln (1 + \exp (-b_i \langle a_i, x_2 \rangle))]

[1 + \exp(-b_i (t \langle a_i, x_1 \rangle + (1 - t)\langle a_i, x_2 \rangle)) \leq]

[\leq (1 + \exp (-b_i \langle a_i, x_1 \rangle))^t (1 + \exp (-b_i \langle a_i, x_2 \rangle))^{1-t}]

[1 + \exp^t(-b_i \langle a_i, x_1 \rangle) \exp^{1-t}(-b_i \langle a_i, x_2 \rangle) \leq]

[\leq 1 + \exp^t(-b_i \langle a_i, x_1 \rangle) \exp^{1-t}(-b_i \langle a_i, x_2 \rangle) + ...]

Thus, we define that f(x) is convex by definition of convex functions. By the way, it's still convex with any $\lambda \geq 0$.

Now, we compute its gradient step by step.

Each individual term inside the summation in the loss function is:

[\ell_i(x) = \ln(1 + \exp(-b_i \langle a_i, x \rangle))]

Define:

[z_i = \langle a_i, x \rangle]

Then,

[\ell_i(x) = \ln(1 + \exp(-b_i z_i))]

Differentiate w.r.t. (x):

[\nabla_x \ell_i(x) = \frac{-b_i \exp(-b_i z_i)}{1 + \exp(-b_i z_i)} a_i]

Using the sigmoid function definition:

[\sigma(y) = \frac{1}{1 + \exp(-y)}]

We rewrite:

[\nabla_x \ell_i(x) = -b_i (1 - \sigma(b_i z_i)) a_i]

So the gradient of the summation term is:

[\nabla_x \left(\frac{1}{m} \sum_{i=1}^{m} \ln(1 + \exp(-b_i \langle a_i, x \rangle)) \right) = \frac{1}{m} \sum_{i=1}^{m} -b_i (1 - \sigma(b_i \langle a_i, x \rangle)) a_i]

The second term in the function is:

[\frac{\lambda}{2} |x|^2]

Since the gradient of (\frac{1}{2} |x|^2) is simply (x), we get:

[\nabla_x \left(\frac{\lambda}{2} |x|^2 \right) = \lambda x]

Thus, the full gradient is:

[\nabla f(x) = \frac{1}{m} \sum_{i=1}^{m} -b_i (1 - \sigma(b_i \langle a_i, x \rangle)) a_i + \lambda x]

Or more compactly:

[\nabla f(x) = -\frac{1}{m} \sum_{i=1}^{m} b_i (1 - \sigma(b_i \langle a_i, x \rangle)) a_i + \lambda x]

For the regularized logistic regression problem:

$$\min_{x \in \mathbb{R}^n} \left(\frac{1}{m} \sum_{i=1}^{m} \ln \left(1 + \exp(-b_i \langle a_i, x \rangle) \right) + \frac{\lambda}{2} |x|^2 \right)$$

I'll determine the smoothness parameter L and the strong convexity parameter μ.

The smoothness parameter L is the upper bound on the eigenvalues of the Hessian matrix of the objective function.

Let's compute the Hessian:

1. First, let's denote $f(x) = \frac{1}{m} \sum_{i=1}^{m} \ln(1 + \exp(-b_i \langle a_i, x \rangle)) + \frac{\lambda}{2}|x|^2$
2. The Hessian is: $$\nabla^2 f(x) = \frac{1}{m} \sum_{i=1}^{m} \frac{\exp(-b_i \langle a_i, x \rangle)}{(1 + \exp(-b_i \langle a_i, x \rangle))^2} \cdot a_i a_i^T + \lambda I$$
3. We can simplify the first term using the logistic function $\sigma(z) = \frac{1}{1+e^{-z}}$: $$\nabla^2 f(x) = \frac{1}{m} \sum_{i=1}^{m} \sigma(b_i \langle a_i, x \rangle) \cdot (1-\sigma(b_i \langle a_i, x \rangle)) \cdot a_i a_i^T + \lambda I$$
4. Note that $\sigma(z)(1-\sigma(z)) \leq \frac{1}{4}$ for all z (maximum at $z=0$)
5. Therefore: $$\nabla^2 f(x) \preceq \frac{1}{4m} \sum_{i=1}^{m} a_i a_i^T + \lambda I$$
6. The maximum eigenvalue of the Hessian is bounded by: $$L = \frac{1}{4m} \lambda_{max}\left(\sum_{i=1}^{m} a_i a_i^T\right) + \lambda$$

If we define $A = [a_1, a_2, \ldots, a_m]^T$, then: $$L = \frac{1}{4m} \lambda_{max}(A^T A) + \lambda = \frac{|A|_2^2}{4m} + \lambda$$

where $|A|_2$ is the spectral norm of matrix A.

The strong convexity parameter μ is the lower bound on the eigenvalues of the Hessian matrix.

1. From the Hessian expression: $$\nabla^2 f(x) = \frac{1}{m} \sum_{i=1}^{m} \sigma(b_i \langle a_i, x \rangle)(1-\sigma(b_i \langle a_i, x \rangle)) \cdot a_i a_i^T + \lambda I$$
2. Since $\sigma(z)(1-\sigma(z)) \geq 0$ for all z, we have: $$\nabla^2 f(x) \succeq \lambda I$$

3. Therefore, the strong convexity parameter is: $$\mu = \lambda$$

For the regularized logistic regression problem:

- **Strong smoothness parameter**: $L = \frac{|A|_2^2}{4m} + \lambda$
- **Strong convexity parameter**: $\mu = \lambda$

The condition number of this optimization problem is therefore: $$\kappa = \frac{L}{\mu} = \frac{|A|_2^2}{4m\lambda} + 1$$

This analysis shows that the regularization parameter λ directly influences both the smoothness and convexity of the problem, and larger values of λ improve the condition number of the problem.

Conclusion

I make realisation and experiments for heavy ball and Nesterovs methods. As a convergence criteria was chosen tol=10e-6 L2 norm weights not changing criteria. All methods were works. The best betta for heavy ball $\beta = 0.6$, for Nesterovs method $\beta = -1$, strategies, used changing from iteration to iteration betta works worst, than constant methods. For all methods I plotted graphics, showing method performance on the given dataset.

Chat AI Bot - Chat GPT | Midjourney | Deepseek | Perplexity, [3/22/25 5:13 PM]

Computational Complexity Analysis of Forming Matrices M^{-1} and M

Problem

We need to determine the number of floating-point operations (FLOP) required to compute matrices M^{-1} and M, where:

$$M^{-1} = \hat{A}^T \Phi^T \Phi \hat{A} \in \mathbb{R}^{n \times n}$$

Here $\Phi = R H^{(\text{norm})}_m S \in \mathbb{R}^{(n+p) \times m}$, where:

- $H^{(\text{norm})}_m = \frac{1}{\sqrt{m}} H_m$ - normalized Hadamard matrix of size $m \times m$
- R - row selection matrix of size $(n+p) \times m$
- S - some matrix (presumably a diagonal scaling matrix) of size $m \times m$
- $\hat{A}$ - some matrix of size $m \times n$

It is known that multiplying a Hadamard matrix by a vector $H_m v$ requires $O(m \log m)$ operations.

Solution

1. Determining the FLOP count for computing M^{-1}

Let's break down the computation of $M^{-1} = \hat{A}^T \Phi^T \Phi \hat{A}$ into stages.

A. Computing $\Phi = R H^{(\text{norm})}_m S$

1. Computing $H^{(\text{norm})}_m S$:

   If S is a diagonal matrix, then $H^{(\text{norm})}_m S$ can be calculated by multiplying each column of $H^{(\text{norm})}_m$ by the corresponding diagonal element of S. This would require m^2 operations.

   If S is an arbitrary matrix, then the product $H^{(\text{norm})}_m S$ can be computed as m multiplications of the matrix $H^{(\text{norm})}_m$ by the columns of S. According to the problem statement, each such multiplication requires $O(m \log m)$ operations, so the total complexity is $O(m^2 \log m)$.

   For further calculation, we'll use the second option - the general case where computing $H^{(\text{norm})}_m S$ requires $O(m^2 \log m)$ FLOP.

2. Computing $R(H^{(\text{norm})}_m S)$:

   Multiplying matrix R of size $(n+p) \times m$ by matrix $(H^{(\text{norm})}_m S)$ of size $m \times m$ gives a matrix of size $(n+p) \times m$. Each element of the result requires m multiplications and $m-1$ additions, giving approximately $2m$ FLOP. With a total of $(n+p) \times m$ elements, the total FLOP count is: $(n+p) \times m \times 2m = 2m^2(n+p)$.

   However, since R is a row selection matrix containing only one "1" in each row, multiplying R by any matrix reduces to selecting the corresponding rows. This can be implemented without arithmetic floating-point operations, so we'll consider this step to require $O(1)$ FLOP.

   In total, computing $\Phi = R H^{(\text{norm})}_m S$ requires $O(m^2 \log m)$ FLOP.

B. Computing $\Phi \hat{A}$

Multiplying matrix Φ of size $(n+p) \times m$ by matrix $\hat{A}$ of size $m \times n$ gives a matrix of size $(n+p) \times n$. This will typically require $(n+p) \times n \times (2m-1) \approx 2m(n+p)n$ FLOP.

C. Computing $\Phi^T (\Phi \hat{A})$

Multiplying matrix Φ^T of size $m \times (n+p)$ by matrix $(\Phi \hat{A})$ of size $(n+p) \times n$ gives a matrix of size $m \times n$. This will require $m \times n \times (2(n+p)-1) \approx 2m \times n \times (n+p)$ FLOP.

D. Computing $\hat{A}^T (\Phi^T \Phi \hat{A})$

Multiplying matrix $\hat{A}^T$ of size $n \times m$ by matrix $\Phi^T \Phi \hat{A}$ of size $m \times n$ gives a matrix of size $n \times n$. This will require $n \times n \times (2m-1) \approx 2mn^2$ FLOP.

Total FLOP count for computing M^{-1}

Summing all stages:

- Computing Φ: $O(m^2 \log m)$ FLOP
- Computing $\Phi \hat{A}$: $2m(n+p)n$ FLOP
- Computing $\Phi^T (\Phi \hat{A})$: $2m \times n \times (n+p)$ FLOP
- Computing $\hat{A}^T (\Phi^T \Phi \hat{A})$: $2mn^2$ FLOP

Since n and p are typically smaller than m, the dominant term in the total complexity will be either $O(m^2 \log m)$ (for computing Φ) or $2m \times n \times (n+p)$ (for computing $\Phi^T (\Phi \hat{A})$), depending on the relationship between the values of n, p, and $\log m$.

For the general case, the total complexity of computing M^{-1} is:

$$O(m^2 \log m + 2m(n+p)n + 2m \times n \times (n+p) + 2mn^2) = O(m^2 \log m + mn(n+p))$$

I need to compute the FLOPs (floating point operations) required to naively compute $\hat{A}^T \hat{A}$.

Given:

- $\hat{A}$ is a dense matrix
- We're using standard matrix multiplication algorithms

Let's assume $\hat{A}$ is of size $m \times n$ (has m rows and n columns).

Then $\hat{A}^T$ is of size $n \times m$, and the resulting product $\hat{A}^T \hat{A}$ will be of size $n \times n$.

For each element of the result matrix, we need to compute the dot product of a row of $\hat{A}^T$ (which is a column of $\hat{A}$) with a column of $\hat{A}$. Each dot product requires m multiplications and $m-1$ additions.

Since the result matrix has $n \times n$ elements, the total number of operations is:

- Multiplications: $n \times n \times m = n^2m$
- Additions: $n \times n \times (m-1) = n^2(m-1) = n^2m - n^2$

The total number of FLOPs is the sum of multiplications and additions: $n^2m + (n^2m - n^2) = 2n^2m - n^2$

However, in the standard definition of FLOPs for matrix multiplication, we typically count a multiplication followed by an addition as a single FLOP pair, and the total is just n^2m.

Therefore, the answer is n^2m FLOPs.

Step 1: Compute $u = \hat{A}v$

- The result u will be a vector in $\mathbb{R}^m$
- For each of the m elements in u, we compute a dot product between a row of $\hat{A}$ and v
- Each dot product requires n multiplications and $n-1$ additions
- Total FLOPs for this step: $m \times (2n-1) = 2mn-m$ (or mn if counting multiplication-addition pairs as single FLOPs)

Step 2: Compute $\hat{A}^T u$

- The result will be a vector in $\mathbb{R}^n$
- For each of the n elements in the result, we compute a dot product between a row of $\hat{A}^T$ (which is a column of $\hat{A}$) and u
- Each dot product requires m multiplications and $m-1$ additions
- Total FLOPs for this step: $n \times (2m-1) = 2mn-n$ (or mn if counting multiplication-addition pairs as single FLOPs)

Total FLOPs for both steps: $(2mn-m) + (2mn-n) = 4mn-(m+n)$ (or $2mn$ if counting multiplication-addition pairs as single FLOPs)

Since we're typically counting each multiplication-addition pair as a single FLOP in matrix operations, the total cost is $2mn$ FLOPs

FLOPs Analysis for Preconditioned Conjugate Gradient Method

Part 1: FLOPs for Preconditioned CG with k iterations

In preconditioned CG, each iteration requires:

- Matrix-vector product with $\hat{A}^T\hat{A}$ (computed as $\hat{A}^T(\hat{A}v)$): $2mn$ FLOPs
- Preconditioning step: applying $M = (\hat{A}^T \Phi^T \Phi \hat{A})^{-1}$ to a vector
- Vector operations (dot products, vector additions, scalar-vector multiplications): $O(n)$ FLOPs

Let's analyze the preconditioning step:

- Applying $M = (\hat{A}^T \Phi^T \Phi \hat{A})^{-1}$ directly would be expensive
- In practice, this would be implemented by solving the system $(\hat{A}^T \Phi^T \Phi \hat{A})z = r$ for each preconditioner application
- This would cost $O(n^3)$ if done directly, but typically more efficient methods are used
- Let's denote this cost as $C_{precond}$

Total FLOPs for k iterations: $k \times (2mn + C_{precond} + O(n)) = O(k(mn + C_{precond}))$

Part 2: FLOPs to directly solve $\hat{A}^T \hat{A} x = \hat{A}^T b$

To directly solve this system:

1. Form $\hat{A}^T\hat{A}$ explicitly: n^2m FLOPs
2. Form $\hat{A}^Tb$: $2nm$ FLOPs
3. Solve the n×n system using Gaussian elimination: $\frac{2n^3}{3}$ FLOPs

Total FLOPs for direct solution: $n^2m + 2nm + \frac{2n^3}{3} = O(n^2m + n^3)$

Part 3: When is CG slower than direct solution?

CG becomes slower when: $k(2mn + C_{precond} + O(n)) > n^2m + 2nm + \frac{2n^3}{3}$

If we assume $C_{precond}$ dominates the per-iteration cost and is of order $O(n^2)$ or higher (which is reasonable for many preconditioners), then:

$k \cdot O(C_{precond}) > O(n^2m + n^3)$

For CG to be slower, k would need to be approximately: $k > \frac{O(n^2m + n^3)}{O(C_{precond})}$

If $C_{precond} = O(n^2)$, then CG becomes slower when $k > O(m + n)$ If $C_{precond} = O(n^3)$, then CG becomes slower when $k > O(1 + \frac{m}{n})$

For many practical problems where m >> n, CG becomes slower when k is on the order of m/n or larger, depending on the exact implementation of the preconditioner.

Newton Method Convergence Analysis Report

Problem Description

We analyze the behavior of Newton's method for minimizing the function: $$f(x,y) = \frac{x^4}{4} - x^2 + 2x + (y-1)^2$$ starting from the initial point $(0,2)$.

Theoretical Analysis

Function Analysis

Let's examine the components needed for Newton's method:

1. **Gradient of f**: $$\nabla f(x,y) = \begin{pmatrix} x^3 - 2x + 2 \ 2(y-1) \end{pmatrix}$$
2. **Hessian matrix of f**: $$H(x,y) = \begin{pmatrix} 3x^2 - 2 & 0 \ 0 & 2 \end{pmatrix}$$
3. **At the starting point $(0,2)$**:
    - $\nabla f(0,2) = \begin{pmatrix} 2 \ 2 \end{pmatrix}$
    - $H(0,2) = \begin{pmatrix} -2 & 0 \ 0 & 2 \end{pmatrix}$

Newton's Method Behavior

The Newton direction is computed as $d = -H^{-1}\nabla f$. At $(0,2)$:

$d = -\begin{pmatrix} -2 & 0 \ 0 & 2 \end{pmatrix}^{-1} \begin{pmatrix} 2 \ 2 \end{pmatrix} = -\begin{pmatrix} -1/2 & 0 \ 0 & 1/2 \end{pmatrix} \begin{pmatrix} 2 \ 2 \end{pmatrix} = \begin{pmatrix} 1 \ -1 \end{pmatrix}$

Critical Issue: The Hessian at $(0,2)$ has a negative eigenvalue (-2), meaning the function is not locally convex. This causes Newton's method to move in a direction of *increasing* function value rather than decreasing it, as it moves toward a saddle point or maximum rather than a minimum.

Comparison with Gradient Methods

Gradient Descent (Fixed Step Size α = 0.01)

The update would be: $x_{k+1} = x_k - 0.01 \nabla f(x_k)$

At $(0,2)$, this gives the direction $(-0.02, -0.02)$, which properly moves toward decreasing function values, though slowly.

Steepest Descent (Line Search)

This method performs a line search in the negative gradient direction, finding an optimal step size. Since the gradient direction is correct, steepest descent will make progress toward a minimum, unlike Newton's method in this case.

Conclusion

Newton's method fails at $(0,2)$ because the Hessian has a negative eigenvalue. This causes the method to move toward a saddle point rather than a minimum. In contrast, gradient-based methods will make progress toward a minimum, though potentially more slowly.

This example highlights a key weakness of Newton's method: it requires positive definiteness of the Hessian to guarantee descent directions. When this condition is violated, the method may diverge or move toward non-minimum critical points.