Optimisation à finalité statistique

Salim Lardjane

Université de Bretagne-Sud

Cours 6 - Algorithmes de Gradient

Algorithmes de gradient

On va considérer dans la suite divers algorithmes de minimisation qui ont pour point commun de *requérir* l'évaluation des dérivées de la fonction en plus de ses valeurs en certains points.

L'essentiel des résultats *théorique* ne s'appliquent *au sens strict* qu'à des fonctions *quadratique* ou *convexes*.

Toutefois, de nombreuses fonctions qu'on souhaite optimiser peuvent être bien approximées par des fonctions quadratiques au voisinage du minimum.

Algorithme de la plus grande pente

Les dérivées partielles d'une fonction par rapport aux m coordonnées du vecteur des paramètres sont appelées globalement *gradient* de la fonction.

Formellement, le vecteur gradient s'écrit

$$g = \begin{pmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_m} \end{pmatrix}$$

Algorithme de la plus grande pente

Le vecteur gradient possède les propriétés suivantes :

— Si on se déplace *dans la direction* de $g(\theta)$ à partir d'un point quelconque θ de $\mathbb{R}^m$, la valeur de la fonction *croît le plus rapidement possible*. On dit que g *pointe* dans la direction du plus grand accroissement de la fonction.

— Si on se déplace dans *la direction opposée* à celle de $g(\theta)$ (direction de l'*anti-gradient*) *la fonction décroît le plus rapidement possible*.

Ces propriétés sont à la base de l'algorithme de la plus grande pente, également appelé algorithme de descente de gradient.

Algorithme de la plus grande pente

Etant donné un point courant θ^i, le point θ^{i+1} est obtenu par *recherche directe unidimensionnelle* dans la direction $-g(\theta^i)$.

Le *processus itératif* de détermination d'un minimum est donc défini par la formule de mise à jour

$$\theta^{i+1} = \theta^i - \lambda_i g(\theta^i)$$

que l'on peut également noter

$$\theta \leftarrow \theta - \lambda g(\theta)$$

Le point initial θ^i est donné et λ_i est déterminé par la procédure de recherche directe unidimensionnelle.

Algorithme de la plus grande pente

A première vue, il peut sembler que l'algorithme de la plus grande pente soit "optimal" puisque chaque recherche unidimensionnelle a lieu dans la meilleure *direction*.

Toutefois, la direction de la plus grande pente est une *propriété locale* plutôt que globale de la fonction.

Par conséquent, de nombreux *changements de direction* peuvent être requis, *ce qui rend la méthode inefficace dans de nombreux problèmes*.

Algorithme de la plus grande pente

La convergence de l'algorithme de la plus grande pente peut être *extrêmement lente* et de nombreuses modifications ont été proposées pour améliorer sa convergence.

Par exemple, une possibilité est d'utiliser comme direction de recherche

$$\theta^i - \theta^{i-2}$$

de temps en temps, à la place de l'antigradient. Cette modificiation a pour effet, en général, d'accélerer la convergence.

Algorithme de la plus grande pente

Exercice 2. Afin d'illustrer l'utilisation de l'algorithme de la plus grande pente, on pourra l'appliquer, sous R et Python, à la fonction de Rosenbrock

$$f(\theta_1, \theta_2) = 100(\theta_2 - \theta_1^2)^2 + (1 - \theta_1)^2$$

On constatera que la convergence est extrêmement lente.

Algorithme de la plus grande pente

En fait, le défaut majeur de l'algorithme de descente de gradient est qu'il ne tient pas compte des *dérivées secondes* de $f(\theta)$, alors que la *courbure* de la fonction, qui détermine son comportement au voisinage du minimum, dépend, elle, de ces dérivées.

Une méthode corrigeant ce défaut est la méthode de Newton-Raphson, implémentée sous R par la fonction **nlm()**.

Algorithme de Newton-Raphson

Une fonction *quadratique* est définie par

$$f(\theta) = a + \theta' b + \frac{1}{2}\theta' B \theta$$

où a est un scalaire, b un vecteur de constantes et B une matrice symétrique réelle qui sera supposée *définie positive*.

Une telle fonction admet un minimum au point θ^* donné par

$$\theta^* = -B^{-1} b$$

Algorithme de Newton-Raphson

En utilisant le développement de Taylor, on peut montrer que, sous certaines conditions de différentiabilité, une fonction $f(\theta)$ peut être approximée, au voisinage d'un point quelconque θ^0 par une fonction $h(\theta)$ définie par

$$h(\theta) = f(\theta^0) + (\theta - \theta^0)' g(\theta^0) + \frac{1}{2}(\theta - \theta^0)' H(\theta^0)(\theta - \theta^0)$$

où g est le vecteur gradient et où H est la matrice hessienne d'élements

$$h_{ij} = \frac{\partial f}{\partial \theta_i \partial \theta_j}$$

Algorithme de Newton-Raphson

On considère alors qu'une *approximation raisonnable* du point où $f(\theta)$ atteint son minimum est fournie par le le point θ^1 où $h(\theta)$ atteint son minimum.

Celui-ci se met sous la forme

$$\theta^1 = \theta^0 - H^{-1}(\theta^0)g(\theta^0)$$

Algorithme de Newton-Raphson

L'*algorithme de Newton-Raphson* utilise θ^1 comme approximation du point où f atteint son minimum et itère la procédure précédente à partir de celui-ci, ce qui donne la formule de mise à jour

$$\theta^{i+1} = \theta^i - H^{-1}(\theta^i)g(\theta^i)$$

Plus généralement, on aura

$$\theta^{i+1} = \theta^i - \lambda_i H^{-1}(\theta^i)g(\theta^i)$$

où λ_i est déterminé par recherche directe *unidimensionnelle* en partant de θ^i dans la *direction* de $-H^{-1}g$.

La convergence de l'algorithme de Newton-Raphson est *très rapide lorsque θ^0 est proche du minimum* puisqu'en général $h(\theta)$ fourni une bonne approximation de $f(\theta)$ dans un tel voisinage.

Algorithme de Newton-Raphson

Cette méthode présente toutefois deux inconvénients :

1. Elle nécessite l'inversion de la hessienne, ce qui peut être lourd du point de vue calculatoire lorsque le nombre de paramètres est élevé.

2. Si θ^i n'est pas proche du minimum, H peut devenir *définie négative*, auquel cas l'algorithme peut ne pas converger.

Algorithme de Newton-Raphson

Exercice 3. Appliquer la méthode de Newton-Raphson à la fonction de Rosenbrock, sous R et Python, et comparer le nombre d'itérations requises pour converger avec le nombre d'itérations requises par l'algorithme de la plus grande pente, en partant des mêmes conditions initiales et pour le même critère d'arrêt.

Algorithme de Newton-Raphson

Plusieurs autres méthodes sont basées sur une direction de recherche de la forme

$$-A(\theta^i)g(\theta^i)$$

l'idée étant de s'affranchir des inconvénients de l'algorithme de Newton-Raphson tout en conservant ses bonnes propriétés de convergence.

A cet effet, A est une matrice symétrique réelle définie positive mise à jour à chaque itération.

La formule de mise à jour doit permettre d'éviter l'inversion d'une matrice $m \times m$ à chaque itération et garantir que A reste définie positive.

Ces méthodes sont appelées *méthodes de Quasi-Newton* et diffèrent l'une de l'autre par la façon dont A est choisi.

Algorithmes de Quasi-Newton

L'algorithme DFP de Davidon, Fletcher et Powell consiste essentiellement en une méthode de la plus grande pente *au début des itérations* pour *se transformer graduellement* en méthode de Newton-Raphson.

Au fil des itérations, des *approximations* A_i de l'inverse de la matrice hessienne au minimum sont mises à jour de façon à ce que les matrices $A_i = A(\theta^i)$ soient toutes définies positives.

Algorithmes de Quasi-Newton

Pour θ, la formule de mise à jour correspondante s'écrit

$$\theta^{i+1} = \theta^i - \lambda_i A_i g_i$$

où $g_i = g(\theta^i)$ et A_i est la i-ème approximation de l'inverse de la hessienne au minimum.

Le pas λ_i correspond au minimum dans la direction de recherche et peut être déterminé par une méthode de recherche unidimensionnelle.

Algorithmes de Quasi-Newton

Le choix de A_0 est *arbitraire*, pourvu qu'elle soit définie positive.

On choisit en général la matrice identité I_m, ce qui a pour conséquence que le premier déplacement a lieu dans la direction de la plus grande pente.

L'approximation de l'inverse de la hessienne au minimum est mise à jour à l'aide de la formule dite BFGS (Broyden, Fletcher, Goldfarb et Shanno) :

$$A_{i+1} = A_i + B_i + C_i$$

où

$$B_i = \frac{zz'}{z'u}$$

et

$$C_i = \frac{A_i u u' A_i}{u' A_i u}$$

avec

$$z = -\lambda_i A_i g_i, \quad u = g_{i+1} - g_i$$

Algorithmes de Quasi-Newton

Exercice 4. Appliquer l'algorithme de Quasi-Newton BFGS à la fonction de Rosenbrock sous R et Python et comparer les résultats obtenus à ceux obtenus par l'algorithme de descente de gradient et l'algorithme de Newton-Raphson.

Algorithme des gradients conjugués

Un facteur critique des algorithmes de quasi-Newton est la direction de recherche obtenue à chaque itération.

Dans ce contexte, un concept important est celui de *directions conjuguées*.

On dit que deux directions p et q sont conjuguées relativement à la matrice symétrique réelle définie positive A si

$$p'Aq = 0$$

On peut démontrer que si les recherches successives sont effectuées dans des directions *mutuellement conjuguées*, alors le minimum d'un fonction quadratique de m paramètres peut être déterminé par un algorithme de type quasi-Newton en m itérations au plus.

Algorithme des gradients conjugués

L'*algorithme de Fletcher-Reeves* tire parti de ce résultat en utilisant une formule de mise à jour simple qui produit une suite de directions mutuellement conjugués.

Le déplacement initial est dans la direction de la plus grande pente et les déplacements suivants sont obtenus à l'aide de la formule

$$d_i = -g_i + \frac{||g_i||^2}{||g_{i-1}||^2} d_{i-1}$$

Les directions ainsi obtenues sont mutuellement conjuguées et l'algorithme détermine le minimum de toute fonction quadratique de m paramètres en m itérations *au plus*.

Algorithme des gradients conjugués

Lorsque l'algorithme est appliqué à des fonctions non quadratiques, il se comporte plus ou moins de la même façon que pour une fonction quadratique dès qu'on est proche du minimum.

En général, on reprend toutes les m itérations la direction de la plus grande pente et on reprend par la suite la construction des directions conjuguées.

Algorithme des gradients conjugués

Exercice 5. Appliquer l'algorithme des gradients conjugués à la fonction de Rosenbrock sous R et Python et comparer les résultats obtenus à ceux obtenus par l'algorithme de descente de gradient, l'algorithme de Newton-Raphson et l'algorithme de quasi-Newton BFGS.