Descente de gradient

Bruno Bouzy

5 octobre 2005

Introduction

Ce chapitre présente ce qu'est une Descente de Gradient (DG). La DG s'applique lorsque l'on cherche le minimum d'une fonction dont on connaît l'expression analytique, qui est dérivable, mais dont le calcul direct du minimum est difficile. C'est un algorithme fondamental à connaître car utilisé partout sous des formes dérivées. Nous ne présentons ici que la version de base.

Exemples

dimension 1

Ce paragraphe présente la DG sur la minimisation de la fonction E de la figure 1:

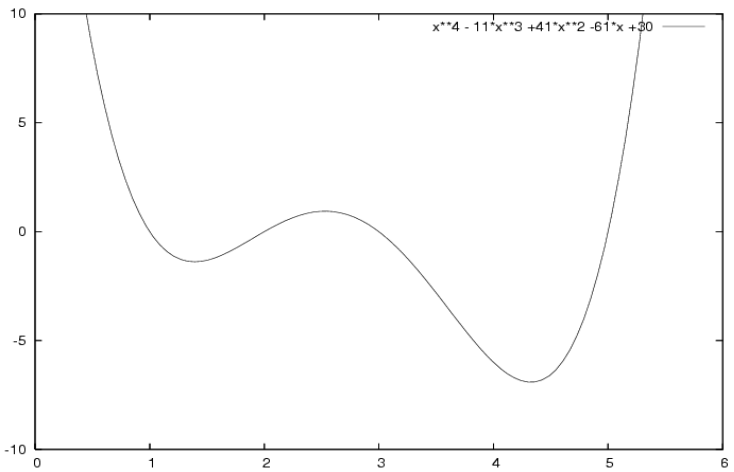

Figure 1: la fonction E(x) = (x-1)(x-2)(x-3)(x-5) dessinée sur [0,6]

Le problème est de trouver la valeur de x qui minimise E(x). Dans cette exemple, on connaît l'expression analytique de la fonction E :

$$E(x) = (x-1)(x-2)(x-3)(x-5)$$
$$= (x2 -3x +2)(x2 -8x +15)$$
$$= x^4 -11x^3 +41x^2 -61x +30$$

On connaît aussi sa dérivée:

$$E'(x) = 4x^3 - 33x^2 + 82x - 61$$

Pour trouver analytiquement le minimum de la fonction E, il faut trouver les racines de l'équation E'(x) = 0, donc trouver les racines d'un polynôme de degré 3, ce qui est « difficile ». Donc on va utiliser la DG. La DG consiste à construire une suite de valeurs x_i (avec x_0 fixé au hasard) de manière itérative:

$$x_{i+1} = x_i - \eta\, E'(x_i)$$

Remarque: cette formule se note aussi:

$$\Delta x = -\eta\, E'(x)$$

Δx représente la valeur que l'on ajoute à x à chaque itération.

On peut donner un critère de fin à la DG par exemple si $\Delta x < \varepsilon$ ou si i>nombre_max.

Pour comprendre ce que fait effectivement la DG, la suite donne des exemples d'exécutions avec différentes valeurs initiales de x_0 et η.

● si $x_0 = 5$ et $\eta = 0.001$, DG trouve x = 4.32 et E = -6.91 en 447 itérations.

Sur la figure 1, vérifier que ce résultat (x = 4.32 et E = -6.19) est plausible.

● si $x_0 = 5$ et $\eta = 0.01$, DG trouve x = 4.32 et E = -6.91 en 39 itérations.

Pourquoi le nombre d'itérations a diminué lorsque η a augmenté ? η s'appelle le « pas d'apprentissage ». La figure 2 montre la convergence de la série de valeurs x_i vers le minimum de E.

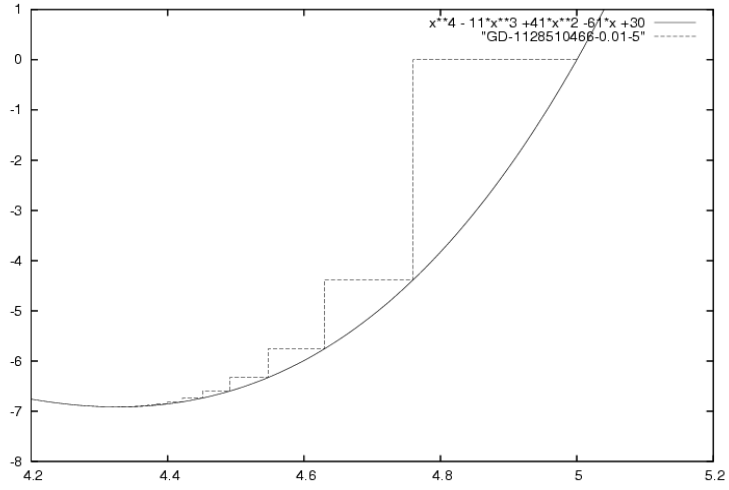

Figure 2: la série de valeurs x_i converge vers le point (4.32, -6,91).

● si $x_0 = 5$ et $\eta = 0.1$, que se passe-t-il ? La réponse est donnée par la figure 3: à partir de la 20ème itération, la série de valeurs oscille entre deux points: (4.48, -6.63) et (4.10, -6.46). La suite n'arrive pas à atteindre le minimum exactement car la pas d'apprentissage est trop grand.

● si $x_0 = 5$ et $\eta = 0.17$, la figure 4 montre ce qui se passe: n'importe quoi, le pas d'apprentissage est trop grand.

● si $x_0 = 5$ et $\eta = 1$, que se passe-t-il ?

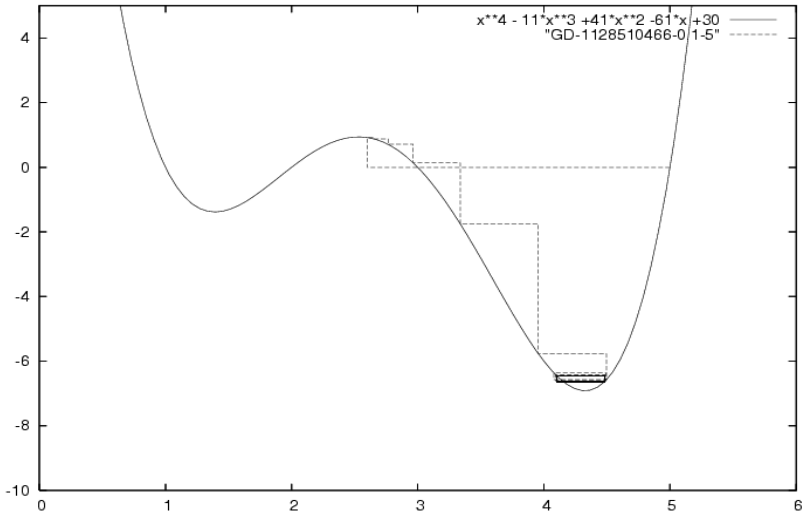

Figure 3: si $\eta = 0.10$, la série de valeurs x_i oscille entre les points (4.48, -6.63) et (4.10, -6.46).

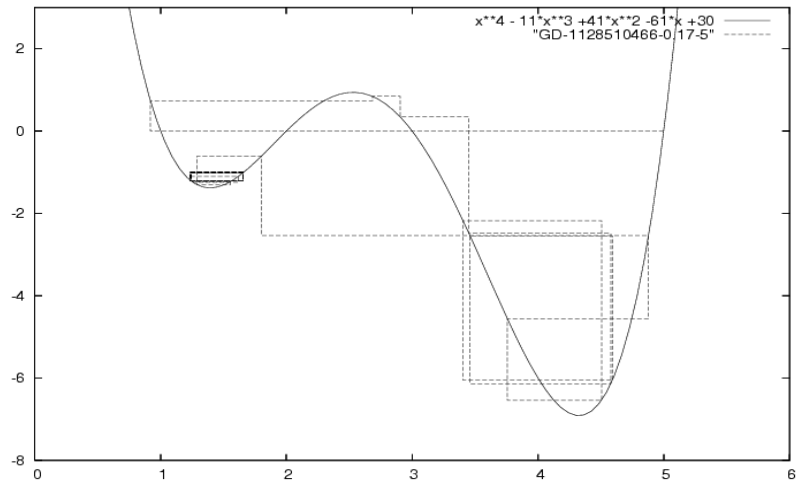

Figure 4: si $\eta = 0.17$, la série de valeurs x_i effectue un « joli parcours ».

On peut aussi voir l'effet de la valeur de départ x_0.

● si $x_0 = 0$ et $\eta = 0.01$, DG trouve x = 1.39 et E = -1.38 en 61 itérations.

Pourquoi la valeur finale est-elle différente de la valeur finale lorsque $x_0 = 5$? Réponse sur la figure 5. Ici le pas d'apprentissage est suffisamment petit, mais la valeur initiale fait que la série converge vers le minimum local (1.39, -1.38).

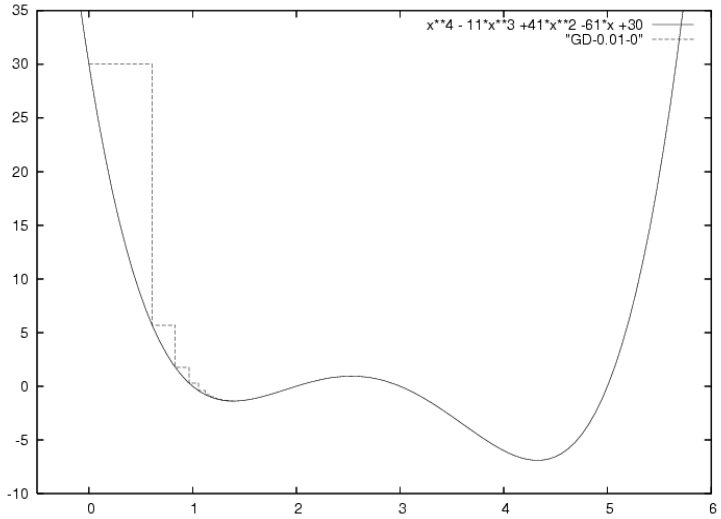

Figure 5: si $x_0 = 0$ et $\eta = 0.01$, la DG converge vers un minimum local.

En résumé, on voit que la DG trouve le bon résultat si la valeur de départ x_0 est plus proche du minimum global que d'un minimum local, et si le pas d'apprentissage est suffisamment petit. Si le pas d'apprentissage est trop grand, DG a un comportement chaotique ou diverge. Si la valeur de départ est mal choisie on trouve un minimum local au lieu de trouver le minimum global.

dimension 2

En dimension 2 avec une fonction E(x,y) à minimiser, l'algorithme DG utilise les formules suivantes:

$$x_{i+1} = x_i - \eta\, \partial E/\partial x(x_i, y_i)$$
$$y_{i+1} = y_i - \eta\, \partial E/\partial y(x_i, y_i)$$

Si on choisit:

$$E(x,y) = (x-1)(x-2) + (y+3)(y+4)$$

on a:

$$\partial E/\partial x(x, y) = 2x - 3$$
$$\partial E/\partial y(x, y) = 2y + 7$$

- si $x_0 = 1$, $y_0 = -5$, et $\eta = 0.01$, DG trouve x = 1.50, y = -3.50 et E = -5.0 en 284 itérations.
- si $x_0 = -1$, $y_0 = -3$, et $\eta = 0.01$, DG trouve x = 1.49, y = -3.50 et E = -5.0 en 309 itérations.
- si $x_0 = -1$, $y_0 = -5$, et $\eta = 0.01$, DG trouve x = 1.49, y = -3.50 et E = -5.0 en 309 itérations.
- si $x_0 = 2$, $y_0 = -4$, et $\eta = 0.01$, DG trouve x = 1.50, y = -3.50 et E = -5.0 en 229 itérations.

Tous ces calculs sont-ils nécessaires ? Aurait-on pu trouver la solution analytiquement ? Y a-t-il des minima locaux ?

dimension n

En dimension n, si on note le vecteur d'entrée $X = (x_1, x_2, ..., x_n)$ on cherche à minimiser la fonction E de n variables $E(X) = E(x_1, x_2, ..., x_n)$. La formule de mise à jour synthétique est:

$$\Delta X = -\eta \nabla E(X)$$

∇E désigne la fonction « gradient » de E, d'où le nom de la méthode. Alors que E(X) est un nombre (dimension 1) fonction de n variables, $\nabla E(X)$ est un *vecteur* avec n coordonnées. Dans la formule de la DG, la jième coordonnée de X, x_j, est mise à jour avec :

$$\Delta x_j = -\eta \, \partial E/\partial x_j(X) = -\eta \, \partial E/\partial x_j(x_1, x_2, ..., x_n)$$

Une implémentation en C

```
// descente de gradient pour z = E(x,y)

#include <stdio.h>

int main() {

  float nu = 0.01; printf("(nu = %f)\n", nu);
  float x = 1; printf("(x0 = %f)\n", x);
  float y = 6; printf("(y0 = %f)\n", y);

  float E = 1; int i; float erreurx = 1000; float erreury = 1000;
  float erreur = 1000; float dEdx = 1; float dEdy = 1;

  for (i=0; i<10000 && erreur>0.01; i++) {
     E = (x-1)*(x-2) + (y+3)*(y+4);
    dEdx = 2*x-3; dEdy = 2*y+7;
    x -= nu*dEdx; y -= nu*dEdy;
    erreurx = (dEdx>0) ? dEdx : -dEdx; erreury = (dEdy>0) ? dEdy : -dEdy;
    erreur = (erreurx>erreury) ? erreurx : erreury;
  }

  printf("i = %d, x = %f, y = %f, E = %f, ", i, x, y, E);
  printf("dE/dx = %f, dE/dy = %f.\n", dEdx, dEdy);
  return (0);

}
```

Références

Antoine Cornuéjols, Laurent Miclet, Apprentissage artificiel, concepts et algorithmes, Eyrolles, pages 57-64.

Christopher Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995, chapitre 1, pages 17-28.