Enoncé du projet pour le cours d'optimisation

Juillet 2021
Ecole d'Eté en Intelligence Artificielle
fondation Vallet
Cotonou, Bénin

Prise en main du code, intuition sur les fonctions

Aller dans le répertoire `Project`.

1. Ouvrir `3Dplots`. C'est un fichier pour dessiner des fonctions en `d<-2` dimensions ("contour plots" et "plot 3D"). Dessiner plusieurs des fonctions données dans `test_functions`[1]. Il suffit de changer le champ `fun<-` et mettre le nom de la fonction à dessiner (par exemple `fun<-quadratic` ou `fun<-rosen` ou `fun<-ackley`, ...).

   (a) Repérer quelles fonctions sont multimodales (i.e., ont plusieurs optima locaux différents).

   (b) Changer la position de l'optimum `glob_xstar` et le conditionnement `cond.no` de la fonction `quadratic`. Regarder l'effet sur les dessins. La fonction `quadratic` est une fonction test fondamentale pour le développement théorique et pratique des optimiseurs: c'est la fonction la plus simple qui possède un minimum, et on peut considérer que localement autour des optima locaux toutes les fonctions sont quadratiques (d'après le développement de Taylor à l'ordre 2).

[1] Les fonctions de `test_functions` marchent avec des dimensions arbitraires

2. Ouvrir `mainOptim`. Il s'agit du programme principal qui permet de

   - formuler le problème (liste `PbFormulation`): choisir une fonction (champ `fun`), son nombre de dimensions (champ `d`), ses bornes (champs `LB,UB`)
   - choisir les paramètres de l'algorithme d'optimisation (liste `optAlgoParam`). Ici ce sont les paramètres des variantes de la méthode de descente vues en cours: des critères d'arrêt (champs `budget`, `minGradNorm`, `minStepSize`), l'activation de la recherche en ligne (`linesearch_type <- "armijo"`) ou non (`linesearch_type <- "none"`), la méthode d'estimation de la direction de recherche (`direction_type <- "gradient"` ou `"momentum"` ou `"NAG"`), le facteur de taille de pas quand la recherche en ligne n'est pas active (`stepFactor`), le coefficient de mémoire `beta` des accélérations de gradient, ...

   (a) Avec la fonction `quadratic` et un point initial pas trop proche de l'optimum `glob_xstar`, observer l'effet de `stepFactor` sur les itérations quand il n'y a pas de recherche en ligne (`linesearch_type <- "none"`). Remarquer qu'en partant du même point avec la recherche en ligne (`linesearch_type <- "armijo"`), la convergence a lieu, au prix de quelques évaluations supplémentaires à chaque itération.

   (b) La fonction Rastrigin est multimodale. Sur la fonction `"rastrigin"` en `d<-2` dimensions, observer des convergences locales, i.e., des points de convergence qui sont des optima locaux mais pas globaux. Pour être sûr de bien observer le phénomène, il faut laisser l'optimiseur converger suffisamment longtemps, typiquement `budget<-10000` et le critère d'arrêt est autre que le budget.

Etude d'une fonction de type apprentissage supervisé

On se propose de créer une nouvelle fonction qui a un rapport étroit avec des applications en machine learning:

$$f(x) \;=\; \sum_{i=1}^{d}(x_i - c_i)^2 + \lambda \sum_{i=1}^{d}|x_i| \quad , \quad \lambda \geq 0 \tag{1}$$

Le premier terme correspond à une parabole sphérique centrée en c (fonction `sphere`). Le second terme est la norme L1 (`L1norm`) multipliée par le coefficient λ.
Analogie avec le machine learning: x sont les poids du réseau; la fonction sphère est un modèle (simpliste) de l'erreur quadratique moyenne d'un réseau de neurones, erreur qui serait minimisée en c; Pour fixer les idées, faisons l'hypothèse que $c_i = i$, $i = 1, \ldots, d$, ainsi les c_i ont des valeurs différentes (accessoirement croissantes). $\lambda\|x\|_1$ est le terme de régularisation, qui aide à avoir une bonne capacité de généralisation du réseau. Nous nous proposons d'explorer pourquoi.

3. Programmer la fonction de l'équation (1) à la façon des autres fonctions du fichier `test_functions`.

4. La minimiser en dimension $d < -10$ avec, par défaut, la descente `"momentum"` complétée par la recherche en ligne `"armijo"`. Essayez plusieurs valeurs de $\lambda \geq 0$. Que remarquez vous sur les $x^\star$ proposés? En quoi cela peut il expliquer une meilleure capacité de généralisation du réseau de neurones?

Rendre une méthode de descente un peu plus globale: redémarrages

Nous avons vu en cours et avec la question 4.(b) que les méthodes de descentes convergent localement, ce qui peut être un inconvénient dans les problèmes où il y a plusieurs solutions.
Une façon simple de remédier partiellement à ce problème est de redémarrer des descentes depuis des points différents.

5. Proposer un algorithme et le programmer qui réalise des redémarrages aléatoires de d'optimisation par descente.

Régularisation vs. nombre de points de données pour un réseau de neurones

Reprendre le code donné dans le fichier `main_neural_net`: il s'agit d'un réseau de neurone utilisé en régression pour apprendre une fonction à 2 di-

mensions (donc il y a 2 entrées dans le réseau). On se propose d'étudier le rapport entre le nombre de données d'apprentissage, `ntrain`, et le bon choix du paramètre de régularisation λ (`lambda` dans le code).

6. Essayer les valeurs : `ntrain`=5, 10 et 50, et toutes leurs combinaisons avec $\lambda = 0.01$, 0.1 et 1. Que remarquez vous sur l'erreur de test (`rmsetest`) ? Comment l'expliquez vous ? Que peut on dire sur l'évolution de l'erreur d'apprentissage (`rmsetrain`) dans cette campagne d'essais? Que remarque-t-on sur les caractéristiques de solutions $x^\star$?

Réponses et commentaires supplémentaires

1.(a).
Unimodales: `sphere,quadratic,rosen,L1norm,tunnel`. AN: `sphere,quadratic,L1norm` sont convexes, pas les autres. `L1norm` est non différentiable.
Multimodales: `ackley,rastrigin,schwefel,michalewicz,quad_wave`

1.(b).
Quand on augmente le conditionnement, la fonction `quadratic` ressemble de plus en plus à une vallée profonde, néanmoins rectiligne, contrairement à `rosen`.

2.(a). Le conditionnement de la fonction quadratique a un role, plus il est grand, plus `stepFactor` doit être petit. Si `stepFactor` est trop grand, la méthode diverge, en d'autres termes elle sort des bornes de $\mathcal{S}$. Vice versa, un petit `stepFactor` permet la convergence, mais à une vitesse inférieure (petits pas). Exemple de valeurs: `xinit<-c(-4.9,-4.9)`, `cond.no<-3`, `stepFactor <- 0.01,0.1,0.9`.

2.(b). Pour trouver plus facilement des points initiaux menant à une convergence locale, générer le point initial au hasard avec `optAlgoParam$xinit <- runif(n = d,min = pbFormulation$LB,max = pbFormulation$UB)`

3.

```
sphereL1 <- function(xx){
  lambda<-5
  y<-sphere(xx)+lambda*L1norm(xx)
  return(y)
}
```

4. Plus λ est grand, plus $x^\star$ s'éloigne de c et tend vers 0 mais d'une façon non rectiligne: certaines composantes de $x^\star$, en l'occurence celles associées aux composantes basses de c, sont mises à 0.
Ceci peut être compris en remarquant le lien entre le problème (1) et l'optimisation

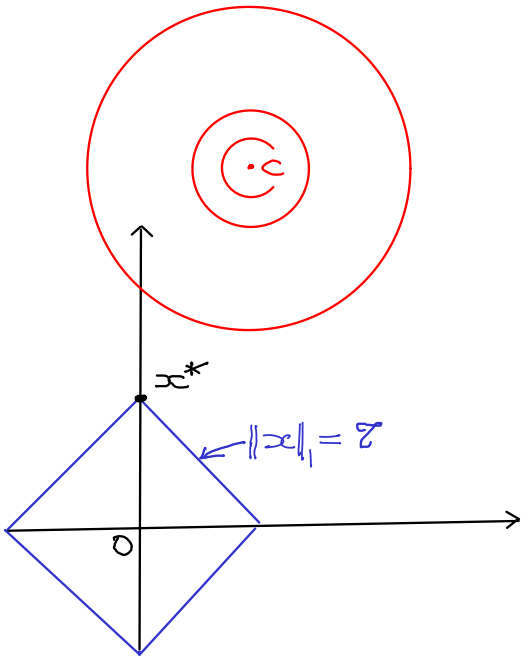
avec contraintes. Le problème

$$\begin{cases} \min_x f(x) = \|x - c\|^2 \\ \text{tel que} \quad g(x) = \|x\|_1 - \tau \leq 0 \end{cases} , \quad \tau > 0$$

a pour Lagrangien à minimiser sur x

$$\min_x f(x) + \lambda^\star g(x) = \|x - c\|^2 + \lambda^\star \|x\|_1 - \lambda^\star \tau$$

Le dernier terme ne dépend pas de x, et les deux précédents sont précisément la fonction régularisée (1). Le dessin ci-dessous représente la fonction sphère et la limite de la contrainte sur $\|x\|_1$. On voit que la solution tend à se trouver sur un sommet du domaine faisable. Or, en ces sommets, des composantes de x s'annulent. Notons que ce phénomène se produit de plus en plus quand on monte en dimensions et est moins frappant quand $d = 2$.

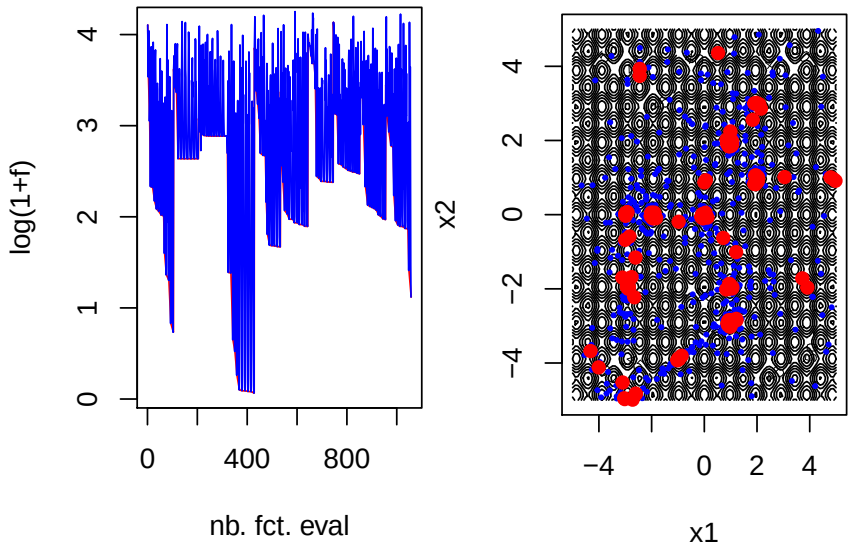


Analogie machine learning: si les composantes de x sont des poids de réseau de neurone, cela veut dire que certains liens dans le réseau de neurone sont supprimés. Ainsi le réseau a une plus faible capacité à surapprendre les données, il généralise mieux. Tout l'art est de trouver la bonne valeur de λ.

5.

Require: `budget`, `nb_restarts`
  for `i in 1` **to** `nb_restarts` **do**
    `xinit <- runif(n=d,min=LB,max=UB)`
    `res<-gradient_descent(xinit,budget=budget/nb_restarts)`
    update global search results
  end for

Plus de détails dans le fichier `restarted_descent`. Exemple d'exécution sur `rastrigin, d<-2, budget<-1000, nb_restart<-10`:



6. Régularisation vs. nombre de données: le plan d'expérience donne les résultats suivants:

	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 1$
`ntrain=5`	0.62266703	0.53166419	0.5217468
`ntrain=10`	0.66836290	0.21722179	0.4202079
`ntrain=50`	0.05191297	0.06292801	0.4229770

Table 1: Root Mean Square Error sur ensemble de test, `ntest=100`. En rouge: plus faible erreur par ligne (par nombre de données d'apprentissage).

On remarque que quand il y a peu de données, il est préférable de beaucoup régulariser ($\lambda = 1$ est mieux avec 5 points d'apprentissage) et vice

versa, avec beaucoup de données (50 points) il n'est pas vraiment nécessaire de régulariser. Intuitivement, cela peut être vu comme un rapport entre la flexibilité du réseau (qui augmente quand λ décroît) et le nombre de points d'apprentissage qui contraignent le réseau à passer par eux: un réseau flexible et peu contraint (maintenu) pour prédire n'importe quoi entre les points d'apprentissage, il généralise mal.
`rmsetrain` augmente avec λ car l'accent est de plus en plus mis sur la minimisation de $\|x\|_1$. D'ailleurs, quand λ est grand, de nombreuses composantes du réseau sont nulles, d'autant plus qu'il y a peu de points de données.