TP 2 : Descente de gradient stochastique et méthode de Newton

Le but de ce TP est d'implémenter différentes améliorations à la méthode de la descente du gradient. Ce TP n'abordera pas les conditions qui font que ces méthodes convergent.

Ce TP est le prolongement du TP1 d'**Optimisation et méthode numériques**. Le découpage en classes s'occupant des descentes et en classes représentant les fonctions est donc toujours d'actualité. Vous allez reprendre le code que vous avez réalisé lors du premier TP ou celui disponible sur Moodle si vous n'êtes pas allés jusqu'au bout du TP1 la semaine dernière. Les objectifs aujourd'hui sont :

- Implémenter la méthode de la descente du gradient stochastique appliquée au problème de la régression linéaire et comparer sa vitesse de convergence avec la méthode classique ;
- Implémenter la méthode de la descente du gradient avec moment ;
- Implémenter la méthode RMSprop qui permet de faire évoluer la taille des pas de la descente de manière différente pour les différentes dimensions ;
- Implémenter la méthode de Newton ;
- Implémenter la méthode BFGS ;

Note : la méthode de la descente du gradient stochastique et RMSprop sont des méthodes utilisées pour accélérer l'entrainement d'un réseau de neurones.

1 Descente du gradient stochastique

La descente de gradient stochastique définit une suite de points $\mathbf{x}_k$ avec une règle de mise à jour semblable à celle de la descente de gradient standard :

$$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + t_k.\mathbf{d}_k.$$

Dans le cas de la régression linéaire, le paramètre d'intérêt s'appelant β changeons la notation en

$$\beta_{k+1} \leftarrow \beta_k + t_k.\mathbf{d}_k.$$

La différence porte sur la direction $\mathbf{d}_k$, qui est dorénavant une variable aléatoire d'espérance égale à l'opposé du gradient de la fonction f à optimiser :

$$\mathbb{E}[\mathbf{d}_k] = -\nabla f(\beta_k).$$

Sous certaines conditions sur f, sur t_k, et sur la distribution de $\mathbf{d}_k$, cette suite converge vers un minimum local de f.

L'idée pour une fonction f définie par la moyenne de fonctions $(f_i)_{i\in 1,\dots,m}$ est de tirer aléatoirement et uniformément un point $i_k \in 1,\dots,m$ à chaque pas de temps k et de choisir $\mathbf{d}_k = -\nabla f_{i_k}(\beta_k)$.

Dans notre cas, $f_i(\beta) = \left(\langle \mathbf{x}^{(i)}, \beta\rangle - y^{(i)}\right)^2$, et donc $\mathbf{d}_k$ prend la forme $-2.\mathbf{x}^{(i_k)}{\mathbf{x}^{(i_k)}}^T.\beta_{k-1} + 2.\mathbf{x}^{(i_k)}.y^{(i_k)}$.

Question 1 : Dans le fichier `least_square_error_grad_sotchastic.py` implémentez la méthode `stochastique_grad()` qui prend en paramètre un point et la taille de l'échantillon e et retourne le gradient stochastique en ce point.

Question 2 : Créez une classe `StochasticGradientDescent` qui permet de réaliser la descente de gradient stochastique.

Astuce 1 : Vous avez déjà une architecture de code qui permet de faire beaucoup de choses. Réfléchissez pour limiter la réécriture de code en vous basant sur l'existant. Si vous êtes bloqué, vous pouvez copier/coller le code et modifier uniquement ce qui doit l'être.

Astuce 2 : Votre classe a besoin d'un paramètre qui représente la taille de l'échantillon.

Question 3 : Testez votre code.

2 Méthode de la descente du gradient avec moment

La méthode de la descente du gradient a des difficultés à converger rapidement quand elle rencontre une crevasse qui n'est pas identique dans toutes les directions. En effet elle va avoir tendance à ricocher sur les pentes. Une solution est d'utiliser le gradient de l'itération précédente pour contrebalancer cela. C'est ce que l'on appelle le **moment**

Figure 1 – Descente de gradient sans moment

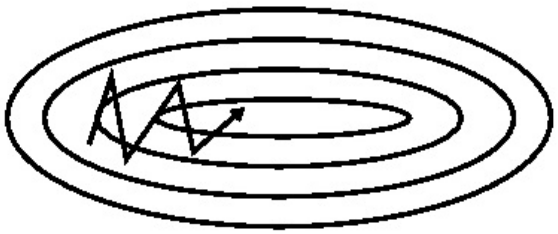

Figure 2 – Descente de gradient avec moment

Ainsi la formule de notre descente devient :

$$v_{k+1} = \gamma * v_k + (1 - \gamma) * grad(f, x_k)$$

$$x_{k+1} = x_k - t * v_{k+1}$$

Au lieu de

$$x_{k+1} = x_k - t * grad(f, x_k)$$

Question 1 : Créez un fichier `gradient_descent_with_momentum.py` et implémentez la classe `GradientDescentWithMomentum` qui permet de réaliser la descente de gradient avec moment. Elle devra hériter de la classe `AbstractGradientDescent`.

Question 2 : Comparez sa vitesse de convergence avec les autres méthodes que vous avez implémenté.

3 Méthode RMSprop

La méthode RMSprop est une amélioration de la méthode des moments qui permet un pas différent pour chaque dimension. Maintenant le pas du gradient est t divisé par la racine carré de la somme pondéré exponentielle des gradients au carré.

$$v_{k+1} = \gamma * v_k + (1-\gamma) * grad(f, x_k)^2$$
$$x_{k+1} = x_k - \frac{t * grad(f, x_k)}{\sqrt{\epsilon + v_{k+1}}}$$

Le facteur ϵ est là pour éviter les divisions par o

Proche d'un plateau ou d'une selle, le gradient devient très petit dans ses dimensions, et va entrainer un déplacement moindre. RMSprop permet des pas plus grands dans ses directions pour s'en échapper facilement.

Question 1 : Créez un fichier `gradient_descent_RMSprop.py` et implémentez la classe `GradientRMSprop` qui permet de réaliser la descente de gradient avec moment. Elle devra hériter de la classe `AbstractGradientDescent`.

Question 2 : Comparez sa vitesse de convergence avec les autres méthodes que vous avez implémentées.

4 La méthode de Newton

La méthode de Newton est une autre méthode numérique pour trouver le minimum d'une fonction. Là où la descente de gradient ne regarde que l'information qu'apporte le gradient, la méthode de Newton va utiliser les informations de la matrice hessienne de la fonction étudiée.

Question 1 : Rappelez la formule de la descente de Newton qui permet de calculer x_{k+1} à partir de x_k, de la hessienne de H_f et du gradient ∇f

Question 2 : Déterminez le pseudo code associé à la méthode Newton et implémentez-là en vous inspirant du code déjà fourni.

Astuce 1 : Vos pouvez créer une nouvelle classe qui hérite de la bonne classe

Astuce 2 : Réfléchissez pour écrire le moins de code possible.

Question 3 : Testez-là sur la fonction de Rosenbrock et l'estimateur des moindres carrés. Cela va vous demander d'implémenter des fonctions qui retournent les hessiennes des fonctions.