Algorithmes d'optimisation basés sur la descente du gradient

Apprentissage et optimisation

Ange Tato

25 mars 2024

Plan

Plan

Descente du gradient

- Apprentissage dans les réseaux de neuronnes = trouver un minimum (le meilleur ?) de la fonction de perte.
- Trouver le minimum d'une fonction $f(x)$ veut dire résoudre $\nabla f(x) = 0$.
- Équation à plusieurs inconnues et la résolution n'est pas triviale !
- Méthodes permettant de faire une approximation du minimum, à l'instar de la descente du gradient.

Descente du gradient

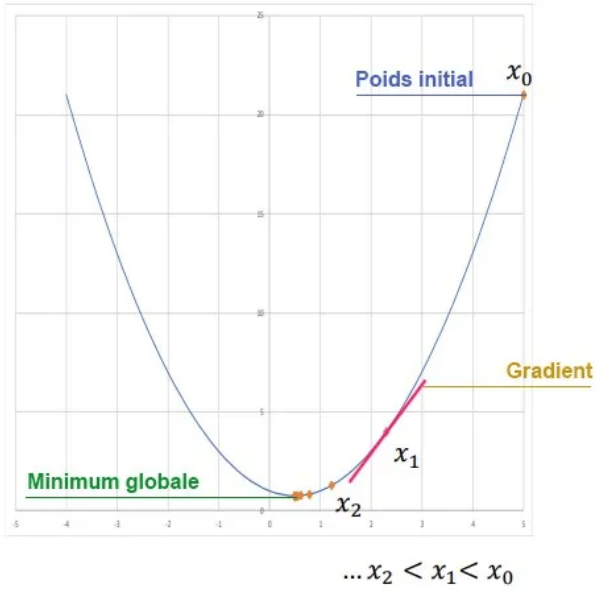
- Apprentissage par descente du gradient : Algorithmes d'optimisation de premier ordre, qui utilisent la dérivée première de la fonction à minimiser.
- Minimisation de la fonction de perte : minimisation de l'écart entre le prédit et le réel.
- Plusieurs algorithmes d'optimisation existants : SGD, Adam, AMSGrad, etc. Tous de premier ordre.
- Principe de la descente du gradient : $w \in \mathbb{R}^n$

$$w_{n+1} = w_n - \delta \cdot \nabla_{w_n} f(w) \tag{1}$$

- $f(w)$ représente la fonction d'erreur. $\nabla_{w_n} f(w)$ représente le gradient.

Descente du gradient

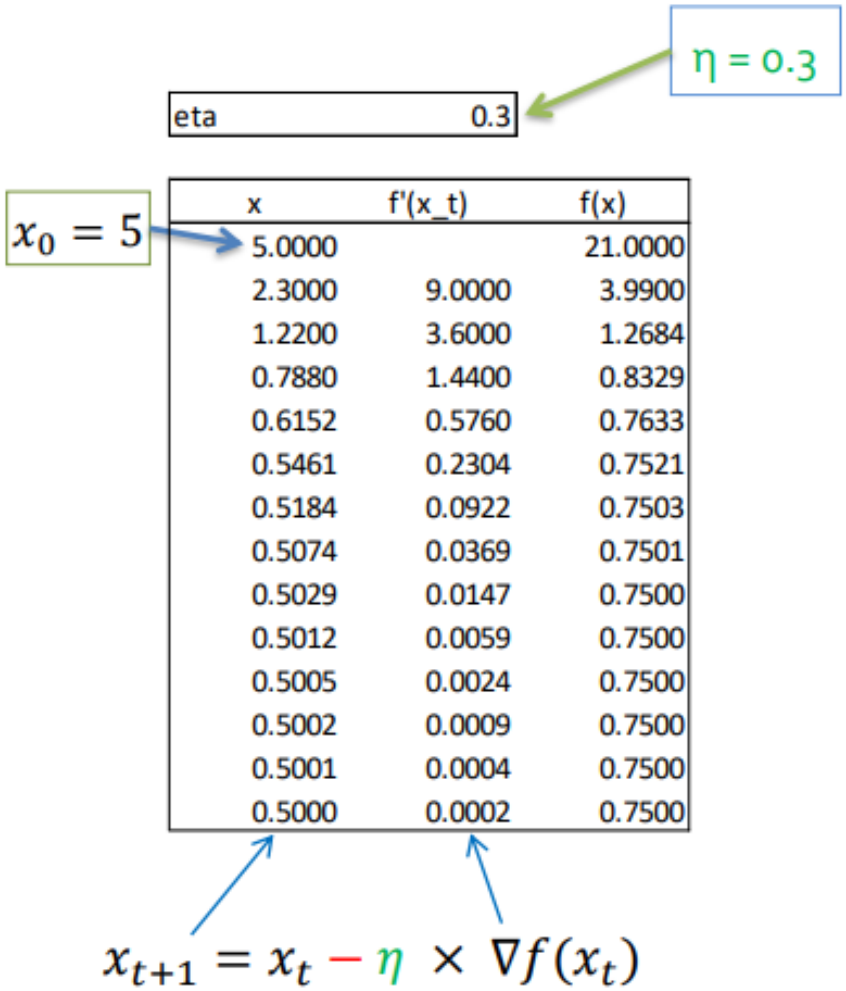
Fonctionnement : **Problème simple à 1 dimension**



- Si on prend la fonction $f(x) = x^2 - x + 1$ comme exemple et que l'on souhaite minimiser par rapport à x, la solution est de résoudre l'équation $f'(x) = 0$ .Ce qui donne $f'(x) = 2x - 1 = 0 \Rightarrow x = \frac{1}{2}$

Descente du gradient

Fonctionnement : **Problème simple à 1 dimension**



Descente du gradient

- Le cas précèdent ne comporte qu'un seul paramètre.
- Les réseaux de neurones comportant des millions de paramètres, le schéma suivant représente d'une façon plus fidèle le problème :

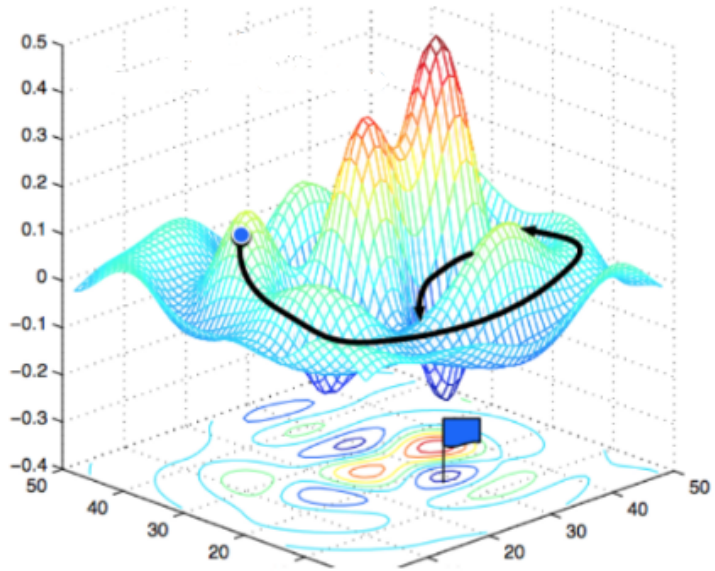


Figure – source

Descente du gradient

Fonctionnement : **Problème à 2 dimensions**

- Prenons l'exemple de $f(x, y) = x^2 + 3y^2$ dont le minimum est atteint en (0, 0) et appliquons la méthode du gradient.
- $\nabla f(x, y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right) = (2x, 6y)$
- On part d'un point $(x_0, y_0) = (2, 1)$ par exemple. On calcule $\nabla f(x_0, y_0) = (4, 6)$. On fixe $\delta = 0.2$.
- On se déplace dans la direction opposée à ce gradient : $(x_1, y_1) = (x_0, y_0) - \delta \nabla f(x_0, y_0) = (2, 1) - 0.2(4, 6) = (2, 1) - (0.8, 1.2) = (1.2, -0.2)$.

Descente du gradient

Fonctionnement : **Problème à 2 dimensions**

- On note que $f(x_1, y_1) = 1.56$ est bien plus petit que $f(x_0, y_0) = 7$.
- On recommence ensuite depuis (x_1, y_1). En quelques étapes les valeurs de f tendent vers la valeur minimale et, dans notre cas, la suite converge vers $(0, 0)$.
- Que se passe-t-il si l'on part d'un autre point ? $(x_0, y_0) = (-1, -1)$ et fixons le pas à $\delta = 0.1$.
- Alors $(x_1, y_1) = (-0.8, -0.4), (x_2, y_2) = (-0.64, -0.16)$... La suite converge également vers $(0, 0)$.

Descente du gradient

Le choix de δ est important :

- Prenons la fonction f définie par $f(x) = x^2 + 1$ et testons différentes 'mauvaises' valeurs du pas δ (avec $x_0 = 2$).

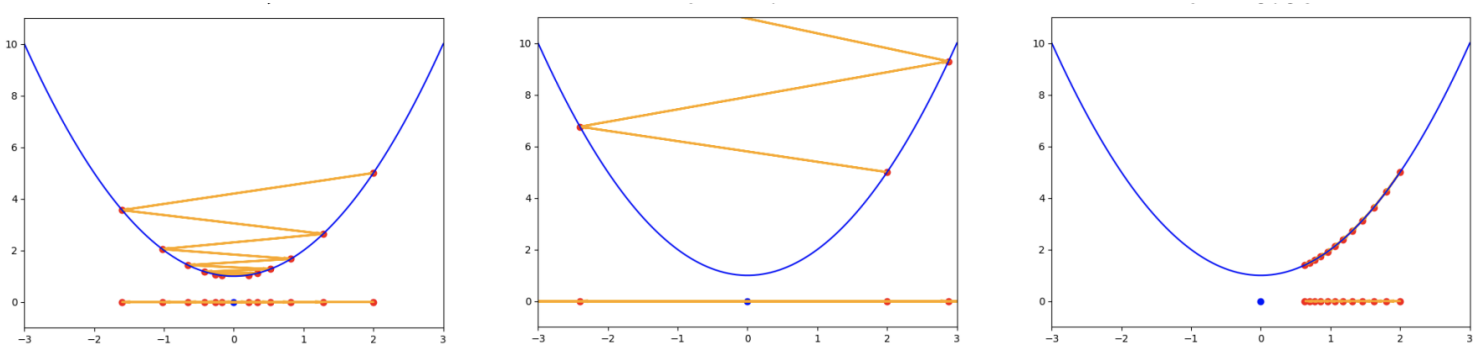


Figure – $\delta = 0.9$, $\delta = 1.1$, $\delta = 0.05$

Descente du gradient

Le choix de δ est important :

- Pour $\delta = 0.9$, la suite x_k tend bien vers $x_{min} = 0$. Les ordonnées sont bien décroissantes mais comme δ est trop grand, la suite des points oscille de part et d'autre du minimum.
- Pour $\delta = 1.1$, la suite x_k diverge. Les ordonnées augmentent, la suite des points oscille et s'échappe. Cette valeur de δ ne donne pas de convergence vers un minimum.
- Pour $\delta = 0.05$, la suite x_k tend bien vers x_{min} mais, comme δ est trop petit, il faudrait beaucoup d'itérations pour arriver à une approximation raisonnable.

Descente du gradient

Le choix du point de départ est important :

- Le choix du point de départ est également important surtout lorsqu'il existe plusieurs minimums locaux.
- Soit la fonction $f(x) = x^4 - 5x^2 + x + 10$.
- Cette fonction admet deux minimums locaux. La suite x_k de la descente de gradient converge vers l'un de ces deux minimums selon le choix du point initial x_0 (ici $\delta = 0.2$).

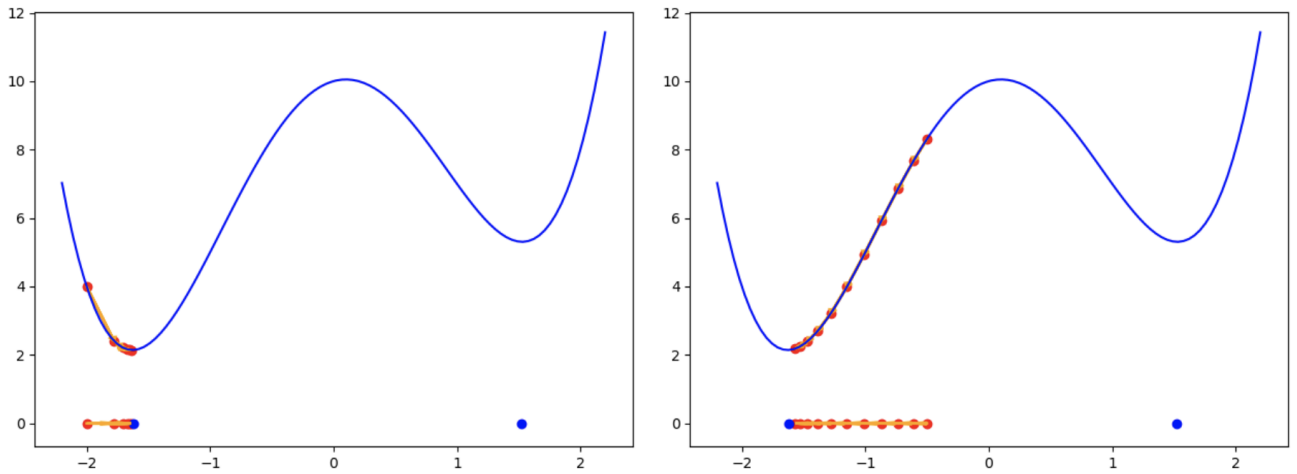


Figure – $x_0 = -2$, $x_0 = -0.5$

Descente du gradient

Le choix du point de départ est important :

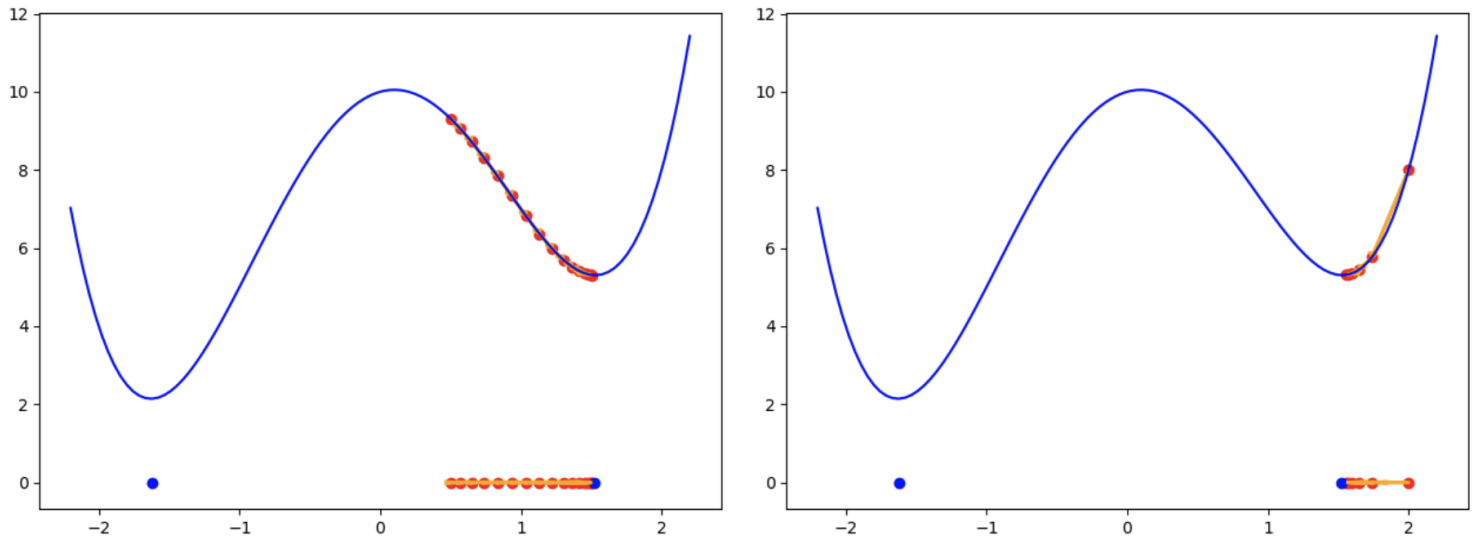


Figure – $x_0 = 0.5$, $x_0 = 2$

Descente du gradient

Algorithme du gradient :

- Soit une fonction $f : \mathbb{R}^n \to \mathbb{R}$, $P \mapsto f(P)$ de plusieurs variables, avec $P = (x_1, ..., x_n)$, dont on sait calculer le gradient $\nabla f(P)$.
- **Données** :
  - Un point initial $P_0 \in \mathbb{R}^n$
  - Un niveau d'erreur $\epsilon > 0$.
- **Itération**. On calcule une suite de points $P_1, P_2, ... \in \mathbb{R}^n$ par récurrence de la façon suivante. Supposons que l'on ait déjà obtenu le point P_k :
  - on calcule $\nabla f(P_k)$,
  - on choisit un pas δ_k et on calcule $P_{k+1} = P_k - \delta_k \nabla f(Pk)$.
- **Arrêt** : $\|\nabla f(P_k)\| \leq \epsilon$

Algorithme du gradient :

- **Remarques** :
  - Plus on choisit le point initial P_0 proche d'un minimum local, plus l'algorithme va aboutir rapidement. Mais comme on ne sait pas où est ce minimum local (c'est ce que l'on cherche), le plus simple est de choisir un P_0 au hasard.
  - Le choix du pas δ_k est crucial. On sait que l'on peut choisir δ_k assez petit de façon à avoir $f(P_k + 1) \leq f(P_k)$ car dans la direction de $-\nabla f(P_k)$ la fonction f décroît.
  - On peut fixer à l'avance un pas δ commun à toutes les itérations. On pourrait également tester à chaque itération plusieurs valeurs de δ par balayage et choisir pour δ_k celui en lequel f prend la plus petite valeur.

Descente du gradient

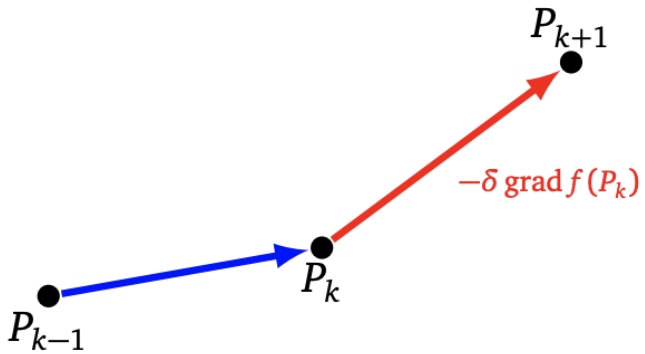
Optimisation de la descente du gradient :

- SGD (descente du gradient stochastique).
- Comment améliorer/accélérer la descente ?

Plan

Accélération de la descente du gradient : Momentum

- Dynamique Newtonniene : Balle lancée du haut d'une montage, gagne de l'inertie (moment) !
- L'inertie de la balle est en quelque sorte la mémoire de la trajectoire passée.
- **Moment** : le vecteur $\overrightarrow{P_{k-1}P_k}$ correspond à la vitesse et est appelé le moment au point P_k.

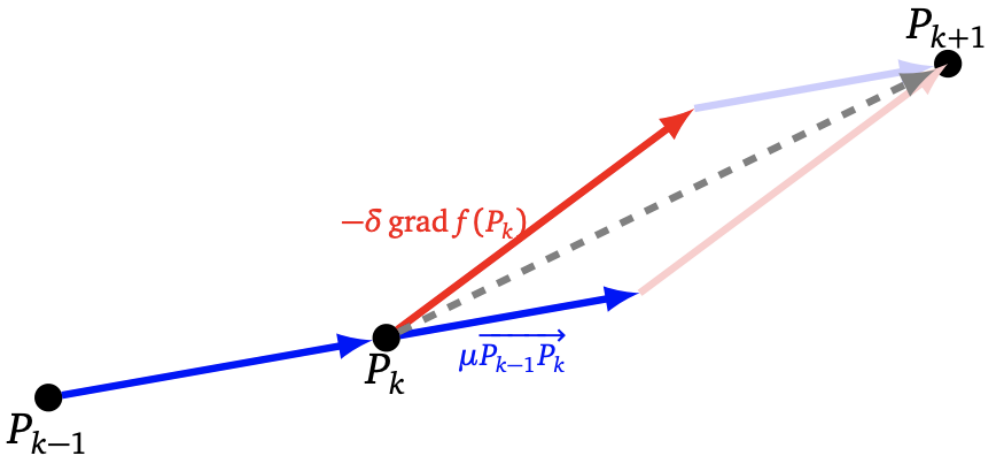


Accélération de la descente du gradient : Momentum

- La formule de la descente de gradient avec **moment** est :

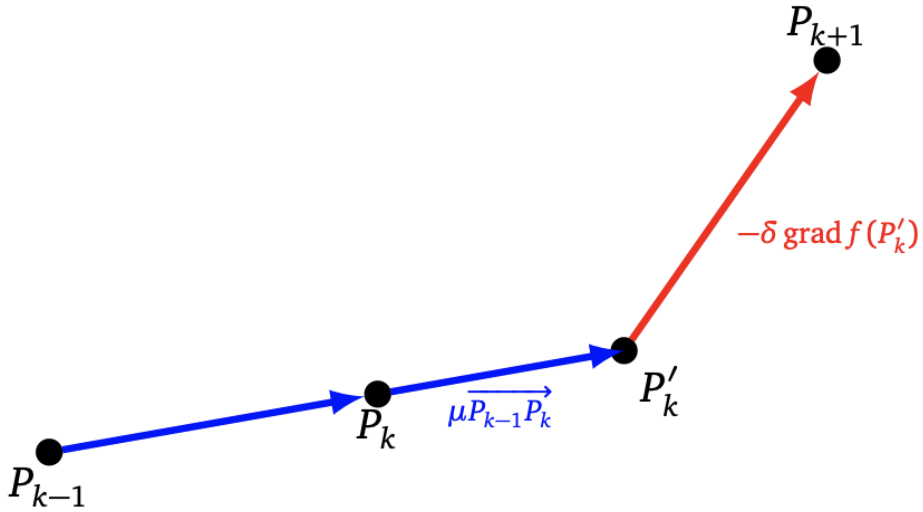
$$P_{k+1} = P_k + \mu\overrightarrow{P_{k-1}P_k} - \delta\nabla f(P_k).$$

- où μ, $\delta \in \mathbb{R}$. On peut prendre par exemple $\mu \in [0.5, 0.9]$ et $\delta = 0.01$.



Accélération de la descente du gradient : *Nesterov Accelerated Gradient*

- La méthode NAG consiste à appliquer d'abord le moment, pour obtenir un point P'_k, puis de calculer le gradient en ce point (et non en P_k).
- La formule est : $P_{k+1} = P_k + \mu\overrightarrow{P_{k-1}P_k} - \delta\nabla f(P_k + \mu\overrightarrow{P_{k-1}P_k})$.
- Si $P'_k = P_k + \mu\overrightarrow{P_{k-1}P_k}$ alors la formule devient : $P_{k+1} = P'_k - \delta\nabla f(P'_k)$.



Algorithmes basés sur la descente du gradient

Accélération de la descente du gradient : *Adaptive Gradient*

- **Problème** : Le taux d'apprentissage qui est utilisé est **constant** pour tous les paramètres même si chacun d'eux ne se trouve pas à la même distance de leurs valeurs optimale.
- **Adagrad** : adapter le taux d'apprentissage en fonction des paramètres.
- δ en plus d'être manuellement spécifié par l'utilisateur avant l'entraînement (hyper-paramètre) il est modifié pour en trouver la valeur optimale au fur et à mesure de l'apprentissage.

Accélération de la descente du gradient : *Adagrad*

- Des mises à jour plus importantes pour les paramètres peu fréquents et des mises à jour plus petites pour les paramètres fréquents.
- Utile dans les problèmes avec données manquantes (*sparse data*).

$$P_{k+1} = P_k - \frac{\delta}{\sqrt{\sum_{i=1}^{k}(\nabla f(P_i))^2 + \epsilon}} \nabla f(P_k)$$

- δ est la valeur initiale du taux d'apprentissage et ϵ est une petite valeur (*smoothing term*) qui permet d'éviter la division par zéro.

Algorithmes basés sur la descente du gradient

Accélération de la descente du gradient : *Adagrad* [1]

- AdaGrad accumule la somme au carré de tous les gradients et l'utilise pour normaliser le taux d'apprentissage afin qu'il puisse être plus ou moins grand selon le comportement des gradients passés.
- **Problème** : le taux d'apprentissage diminue au point que l'apprentissage s'arrête complètement à cause du très faible taux d'apprentissage.
- Le taux d'apprentissage aura tendance à s'approcher de 0 après un certain nombre d'itérations.

Accélération de la descente du gradient : *Root Mean Square*

- RMSprop [1] : version améliorée de AdaGrad où l'accumulation agressive des gradients au carrés a été résolue.
- RMSprop décompose le gradient accumulé de telle sorte que seule une partie des gradients passés est considérée.
- Moyenne mobile (*moving average*) des gradients passés.

$$\begin{aligned} P_{k+1} &= P_k - \frac{\delta}{\sqrt{G_k + \epsilon}} \nabla f(P_k) \\ G_k &= \gamma \cdot G_{k-1} + (1-\gamma) \cdot (\nabla f(P_k))^2. \end{aligned}$$

1. https ://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Algorithmes basés sur la descente du gradient

Accélération de la descente du gradient : *Root Mean Square*

- La mise à jour prend la portion γ de la somme cumulée passée du gradient au carré et prend la portion $(1-\gamma)$ du gradient au carré courant.
- **Inconvénient** : Problème d'initialisation de G_k. Valeurs très loin des gradients.

Plan

Adam et AAdam

Adam : Adaptive Moment Estimation[2]

- Basé sur des estimations adaptatives de moments d'ordre inférieur. Un moment est une quantité qui mesure la forme d'une fonction.
- Le premier moment (moyenne), une fois normalisé par le second moment (variance non centrée), donne la direction de la mise à jour.
- Combinaison de RMSprop et des techniques de momentum comme NAG.

$$
\begin{aligned}
P_{k+1} &= P_k - \frac{\delta}{\sqrt{\hat{v}_k + \epsilon}} \hat{m}_k \\
m_k &= \beta_1 \cdot m_{k-1} + (1 - \beta_1) \cdot \nabla f(P_k) \\
v_k &= \beta_2 \cdot v_{k-1} + (1 - \beta_2) \cdot \nabla f(P_k)^2 \\
\hat{m}_k &= \frac{m_k}{1 - \beta_1^k} \\
\hat{v}_k &= \frac{v_k}{1 - \beta_2^k}.
\end{aligned}
$$

AMSGrad[3]

- Mémoire à long terme : maintient le maximum de tous les v_k qui est une moyenne exponentiellement décroissante des gradients carrés passés, jusqu'à l'instant courant et utilise cette valeur maximale pour normaliser la moyenne courante du gradient.

$$
\begin{aligned}
P_{k+1} &= P_k - \frac{\delta}{\sqrt{\hat{v}_k + \epsilon}} \hat{m}_k \\
m_k &= \beta_1 \cdot m_{k-1} + (1 - \beta_1) \cdot \nabla f(P_k) \\
v_k &= \beta_2 \cdot v_{k-1} + (1 - \beta_2) \cdot \nabla f(P_k)^2 \\
\hat{m}_k &= \frac{m_k}{1 - \beta_1^k} \\
\hat{v}_k &= max(\hat{v}_{k-1}, \frac{v_k}{1 - \beta_2^k}).
\end{aligned}
$$

AdamW [4]

- Variante de Adam qui intègre une régularisation L2 (aussi appelée décroissance du poids) directement dans la fonction de perte pendant l'entraînement.

$$
\begin{aligned}
P_{k+1} &= P_k - \left(\frac{\delta}{\sqrt{\hat{v}_k + \epsilon}}\hat{m}_k + \lambda \cdot P_k\right) \\
m_k &= \beta_1 \cdot m_{k-1} + (1-\beta_1) \cdot \nabla f(P_k) \\
v_k &= \beta_2 \cdot v_{k-1} + (1-\beta_2) \cdot \nabla f(P_k)^2 \\
\hat{m}_k &= \frac{m_k}{1-\beta_1^k} \\
\hat{v}_k &= \frac{v_k}{1-\beta_2^k}.
\end{aligned}
$$

AdamW

- Stabilité de l'entraînement : La régularisation L2 aide à stabiliser l'entraînement en évitant le surajustement (overfitting) en réduisant la variance des paramètres du modèle.
- Particulièrement bénéfique pour les modèles profonds avec de nombreux paramètres, car cela aide à contrôler la complexité du modèle.
- Meilleure généralisation : En réduisant la magnitude des poids, AdamW peut conduire à une meilleure généralisation du modèle.
- Convergence plus rapide : La régularisation L2 peut favoriser une convergence plus rapide de l'algorithme d'optimisation, ce qui peut réduire le temps nécessaire pour entraîner le modèle.

Accélération des méthodes existantes : fonctionnement

- Si la direction de la pente ne change pas, alors on peut prendre de plus grands pas pour se diriger plus rapidement vers le minimum.
- Par contre, si la direction change, alors on doit continuer la descente du gradient normalement.

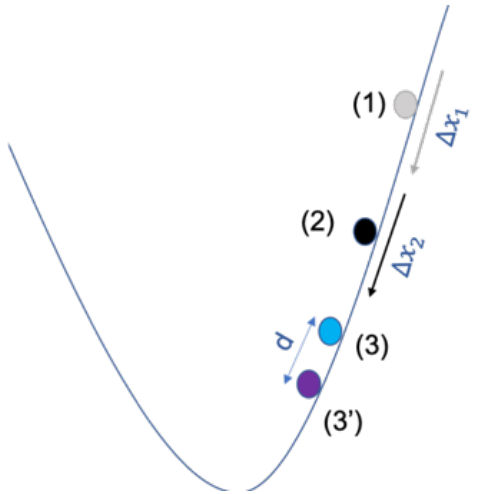


Figure – (1) Position initiale ; (2) Position après la première itération du GD (Gradient Descent) ; (3) Position après la 2eme itération du GD ; (3') Positionaprès la 2eme itération du AGD (Accelerated GD).

Accélération des méthodes existantes : fonctionnement

- Position initiale : x_1 ;
- Position à la prochaine itération : $x_2 = x_1 - \delta \cdot \nabla f(x_1)$ en utilisant la version la plus basique de l'algorithme de la descente du gradient ;
- Position à la 2eme itération : $x_3 = x_2 - \delta \cdot \nabla f(x_2)$;
- (3') Position à la 2eme itération avec accélération vu que on a $\nabla f(x_2) \cdot \nabla f(x_1) > 0$: $x_3 = x_2 - \eta \cdot (\nabla f(x_2) + \nabla f(x_1))$;
- gain de $d = -\delta \cdot \nabla f(x_1)$;
- Dans le cas de Adam et AMSGrad, la direction du pas passée est conservée dans le moment d'ordre 1 (m) ;
- Le test de la variation de la direction sera donc fait entre m et le gradient actuel.
- Il faut fixer un seuil $S <| \nabla f(x_2) + \nabla(x_1) |$ au-delà duquel on arrête l'accélération pour éviter d'osciller autour du minimum.

Accélération des méthodes existantes : AAdam

- En pratique : version python Tensorflow, version python Keras.

$$
\begin{aligned}
P_{k+1} &= P_k - (\delta \frac{\beta_1}{\sqrt{\hat{v}_k + \epsilon}} \hat{m}_k + d) \\
d &= \Delta P_{k-1} * sign(\nabla_{P_k} f(P_k)) * (1 - \beta_1) \\
\hat{m}_k, \hat{v}_k &= \text{Paramètres de Adam} \\
\Delta\theta_{k-1} &= \text{Dernière mise à jour.}
\end{aligned}
$$

Plan

Quel algorithme d'optimisation choisir ?

- Le choix de l'optimiseur dépend du type de problème, de données et des caractéristiques spécifiques du modèle.
- Descente de gradient stochastique (SGD) :
  - utilisé comme optimiseur de base.
  - Simple et efficace pour les problèmes simples.
  - Peut être amélioré avec des techniques telles que la descente de gradient stochastique avec momentum.
- SGD avec momentum :
  - Ajoute un terme de momentum pour accélérer la convergence.
  - Utile pour les fonctions de coût avec de nombreux minima locaux.

Quel algorithme d'optimisation choisir ?

- Adam (Adaptive Moment Estimation) :
  - Combinaison de la méthode du gradient adaptatif et du moment d'ordre un.
  - Bien adapté aux réseaux de neurones profonds et aux problèmes avec des données volumineuses.
  - Gère automatiquement le taux d'apprentissage pour chaque paramètre.
- AAdam
  - Variation d'Adam qui vise à accéléré la convergence de la version originale d'Adam. Utilisé dans les mêmes cas que Adam.
- RMSprop (Root Mean Square Propagation) :
  - Adapte le taux d'apprentissage de chaque paramètre en fonction de la moyenne des gradients récents pour ce paramètre.
  - Utile pour atténuer les problèmes de taux d'apprentissage trop élevé ou trop faible.

Quel algorithme d'optimisation choisir ?

- Adagrad (Adaptive Gradient Algorithm) :
  - Adapte le taux d'apprentissage de chaque paramètre en fonction de la somme des gradients précédents pour ce paramètre.
  - Convient aux données rares ou peu fréquentes.
- AMSGrad :
  - Variation d'Adam qui vise à résoudre un problème de convergence identifié dans la version originale d'Adam.
- AdamW :
  - Variation de Adam avec une régularisation L2 sur les poids.
  - Peut aider à stabiliser l'entraînement et à améliorer les performances générales du modèle.

Quel algorithme d'optimisation choisir ?

- Un tableau récapitulatif intéressant des avantages et inconvénients de chaque algorithme : Summary of popular optimizers highlighting their strengths and weaknesses.
- Tests des algotithmes sur une fonction simple.
- Tests des algorithmes sur mnist.

References

[1] John Duchi, Elad Hazan et Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In : *Journal of Machine Learning Research* 12.Jul (2011), p. 2121-2159.

[2] Diederik P Kingma et Jimmy Ba. "Adam: A method for stochastic optimization". In : *arXiv preprint arXiv:1412.6980* (2014).

[3] Sashank J Reddi, Satyen Kale et Sanjiv Kumar. "On the convergence of adam and beyond". In : *arXiv preprint arXiv:1904.09237* (2019).

[4] Ilya Loshchilov et Frank Hutter. "Decoupled weight decay regularization". In : *arXiv preprint arXiv:1711.05101* (2017).