Cours IFT6266,

Optimisation à base de gradient

1 Petit rappel sur le gradient

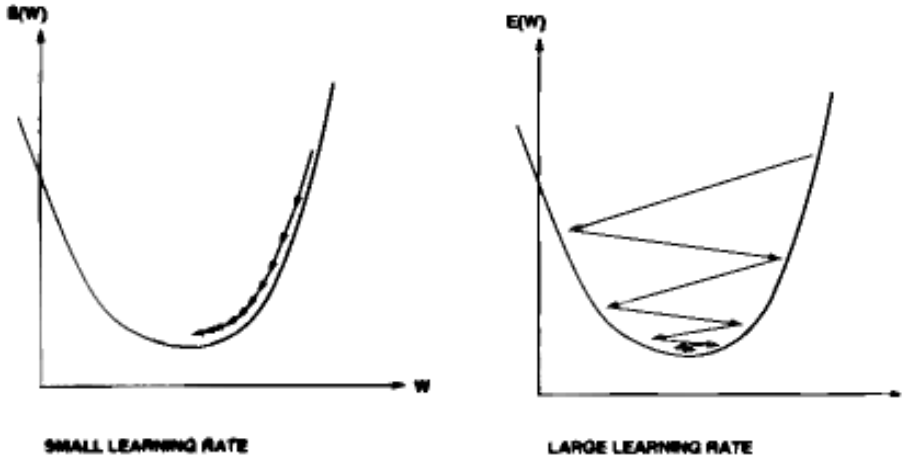
La dérivée partielle de C (qui peut dépendre de w et d'autres variables), est

$$\frac{\partial C(w)}{\partial w} = \lim_{\delta w \to 0} \frac{C(w + \delta w) - C(w)}{\delta w}$$

donc cela nous dit comment un changement de w affecte C. Supposons que w soit un vecteur de paramètres et C une fonction de coût à minimiser. Si $\frac{\partial C}{\partial w_i} < 0$ cela signifie qu'une augmentation (infinitésimale) du paramètre w_i ferait baisser l'erreur C. On voudrait donc changer w dans la direction du vecteur de gradient $\frac{\partial C}{\partial w} = (\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \ldots, \frac{\partial C}{\partial w_m})$. C'est le principe de la **descente de gradient**:

$$w_{t+1} \leftarrow w_t - \eta_t \frac{\partial C(w_t)}{\partial w}$$

Si η_t est suffisamment petit, cette procédure est garantie de réduire C à chaque fois. Si η_t n'est pas trop petit on aura convergeance vers un minimum de C, mais ça pourrait être un minimum local.



SMALL LEARNING RATE

LARGE LEARNING RATE

2 Calcul du gradient et graphe de flot

On applique les règles de dérivation, et en particulier la règle de **dérivation en chaîne** est la plus utile:

$$\frac{\partial C(u_1(w), u_2(w), \ldots, u_n(w))}{\partial w} = \sum_{i=1}^{n} \frac{\partial C}{\partial u_i} \frac{\partial u_i}{\partial w}$$

La puissance de cette règle de dérivation en chaîne devient apparente quand **on l'applique récursivement**. Supposons que l'on peut décompose $C(w)$ en une série d'opérations différentiables, telles que le calcul peut se représenter par une séquence de ces opérations, où les intrants de chaque opération s'obtiennent à partir des résultats des opérations précédentes (ou bien à partir de w ou à partir de quantités externes ou constantes). Notons v_i le résultat de la i-ème opération, obtenu en appliquant la fonction f_i à un sous-ensemble de $\{v_1, \ldots, v_{i-1}\}$. Pour simplifier la notation on va prendre $v_1 = w$, et on s'intéresse donc à obtenir $\frac{\partial C}{\partial v_1}$. On va le faire en calculant les résultats intermédiaires $\frac{\partial C}{\partial v_i}$. Le temps total du calcul de C est donc la somme des temps de calcul de chacun des $v_i = f_i(v_1, \ldots, v_{i-1})$. On peut montrer que l'on peut obtenir le gradient $\frac{\partial C}{\partial w}$ dans le même ordre de temps de calcul, en appliquant la règle de dérivation en chaîne récursivement. De même qu'on visite et calcule les v_i séquentiellement pour calculer C, on va faire la même chose pour le gradient, *mais en sens inverse* (en commençant par le dernier v_i, qui est C lui-même). Pour chacun on va calculer

$$\frac{\partial C}{\partial v_i} = \sum_{j>i} \frac{\partial C}{\partial v_j} \frac{\partial f_j}{\partial v_i}$$

où $\frac{\partial f_j}{\partial v_i}$ vaut 0 si v_i ne rentre pas comme argument direct de la fonction f_j. On initialise la récursion avec $\frac{\partial C}{\partial C} = 1$ et on continue en **rétro-propageant le gradient** dans le graphe de flot, dans le sens inverse du calcul de C. Pour chaque noeud du graphe on a donc v_i, calculé dans la **passe avant** du calcul de C en fonction de w, et $\frac{\partial C}{\partial v_i}$, calculé dans la **passe arrière** de **rétro-propagation** du gradient, dans l'ordre inverse.

Des implantations génériques de cette procédure récursive existent et utilisent justement une structure de données en graphe, avec pour chaque noeud les champs v_i et $\frac{\partial C}{\partial v_i}$. Notons que les v_i peuvent être scalaires, vecteurs, matrices, etc...

Quel est l'avantage de cette approche récursive? c'est qu'elle peut nous permettre d'obtenir le gradient en un temps exponentiellement plus petit qu'une approche naïve qui n'exploite pas le fait que l'expansion de $C(w)$ contient des termes

réutilisés à beaucoup d'endroits. C'est en fait une application du principe général de *programmation dynamique* en informatique.

3 Gradient stochastique

Dans le contexte de l'apprentissage, la fonction de coût est souvent exprimée comme une somme de termes indépendents associés à chaque exemple:

$$C(w) = \sum_i Q(f(x_i, w), y_i) = \sum_i C_i(w)$$

La descente de gradient ordinaire (appelée **"batch"** en anglais) utilise simplement $\frac{\partial C}{\partial w}$ pour calculer chaque modification de w. La descente de gradient **stochastique** utilise $\frac{\partial C_i}{\partial w}$ et les paramètres sont modifiés après chaque exemple:

$$w_{t+1} \leftarrow w_t - \eta \frac{\partial C_i(w_t)}{\partial w}.$$

Si on a un ensemble d'apprentissage fini (qui ne grossit pas plus vite que le temps que ça nous prend pour faire ces mise à jour), on va recycler les exemples déjà vus (i.e., itérer plusieurs fois sur l'ensemble d'apprentissage). Avec cette méthode, l'apprentissage se fait généralement de manière beaucoup **plus rapide** (à cause de la redondance présente dans les exemples), et de manière **plus robuste au problème des minima locaux**.

L'apprentissage par descente de gradient stochastique est un cas particulier de ce qu'on appelle l'apprentissage "on-line", ou exemple par exemple. Considérez un ensemble d'entraînement qui est l'union de deux copies d'un même sous-ensemble d'exemples. Dans ce cas il est clair que la descente de gradient "batch" va faire deux fois le calcul des gradients, pour rien, alors que la descente de gradient stochastique ira aussi vite que si on avait pas fait de copie des données. L'application de cette idée donne lieu au fait que la descente de gradient stochastique devient proportionnellement de plus en plus efficace par rapport aux méthodes "batch" au fur et à mesure que le nombre d'exemples augmente. On voit aussi clairement que quand ce nombre d'exemples tend vers l'infini, on ne peut pas utiliser les méthodes "batch" (puisque celles-ci requièrent plusieurs itérations sur l'ensemble d'apprentissage, et qu'on a seulement le temps d'en faire UNE), mais on peut utiliser une méthode "on-line" comme la descente de gradient stochastique, et obtenir la convergence de l'apprentissage. Dans le cas où nous avons non pas un ensemble fixe d'exemples mais une SOURCE d'exemples (par exemple les données arrivent en temps réel

de la télé), à un taux comparable ou plus élevé que le temps requis pour traiter (et faire une mise à jour des paramètres) chaque exemple, alors nous n'avons pas le choix d'utiliser une méthode "on-line". Au mieux nous pouvons stocker une quantité limitée d'exemples à la fois et faire des "mini-batch", mais nous ne pouvons engoufrer tout l'ensemble d'exemples et faire plusieurs itérations dessus. Cela élimine la grande majorité des méthodes d'optimisation numérique classiques, qui supposent que l'on peut calculer $C(w)$ un grand nombre de fois pour différentes valeurs de w.

Pour ce qui est des minima locaux et aussi pour mieux comprendre la nature du gradient stochastique, il faut voir $\frac{\partial C_i(w_t)}{\partial w}$ comme un *estimateur sans biais* de $\frac{\partial C(w_t)}{\partial w}$. Cela est clairement vrai puisque la moyenne des $\frac{\partial C_i(w_t)}{\partial w}$ est égal (à une constante multiplicative près) à $\frac{\partial C(w_t)}{\partial w}$. Donc

$$\frac{\partial C_i(w_t)}{\partial w} = \frac{\partial C(w_t)}{\partial w} + bruit$$

où le *bruit* est de moyenne nulle. Un autre facteur intéressant est que sous l'hypothèse i.i.d. des données, les $\frac{\partial C_i(w_t)}{\partial w}$ sont i.i.d. entre eux. Donc la moyenne d'un petit nombre d'entre eux (une "mini-batch") devient rapidement un bon estimateur de leur espérance, et ce à un coût de calcul considérablement moindre (et le *bruit* diminue en 1 sur la racine carrée de la taille de la "mini-batch"). Avec le gradient stochastique, on devrait penser non pas en terme d'une valeur unique pour w_t mais plutôt en fonction d'un nuage de valeurs, car w_t va "s'agiter" de manière aléatoire à cause du *bruit*, et c'est le nuage dans son ensemble qui se déplace au fur et à mesure que l'on descend vers des valeurs plus faibles de C. La taille de ce nuage est proportionnelle au *bruit* et au pas de gradient η. **Le nuage ne pourra pas rentrer dans un "trou", ou un bol, qui est plus mince que le nuage lui-même**. On va donc éviter les minima locaux "pointus", qui sont ceux que l'on veut justement éviter le plus (ils donnent généralement lieu à une mauvaise généralisation car ils correspondent à des solutions qui requièrent une grande précision en w, donc une classe de fonctions plus grande, et correspondent généralement à des solutions peu stables par rapport au tirage particulier des exemples d'apprentissage). Avec cette vision des choses, on voit aussi pourquoi il sera important de graduellement réduire η si on veut descendre au fond du bol de notre minimum final.

On peut évidemment utiliser le même truc du calcul efficace du gradient par rétropropagation dans un graphe de flot, mais avec C_i comme noeud terminal du graphe.

4 Convergence de la descente de gradient

Dans le cas de la descente de gradient ordinaire comme celui de la descente de gradient stochastique, on peut garantir asymptotiquement la convergence vers un minimum (qui peut être local) si les pas de gradients η sont positifs et sont graduellement réduits:

$$\sum_{t=1}^{\infty} \eta_t = \infty$$

et

$$\sum_{t=1}^{\infty} \eta_t^2 < \infty$$

Voir la thèse de Léon Bottou sur le site du cours pour des preuves de la convergence.

En pratique on peut prendre un pas de la forme

$$\eta_t = \frac{\eta_1}{1 + t/c}$$

et on se retrouve avec deux hyper-paramètres η_1 (le pas de gradient initial) et c (la vitesse de décroissance), qu'il faudra estimer en essayant différentes valeurs. On peut commencer avec $c = \infty$, donc un pas constant, pour choisir η_1, en se basant sur la vitesse de convergence initiale. Ensuite, on choisira c en se basant sur la convergence finale. Ce que l'on peut espérer c'est qu'une fois ce choix fait, il sera relativement stable par rapport à des variations mineures de nos données et de notre classe de fonctions.