Descente de gradient parallélisée

Matthieu Vegreville
Guillaume Wenzek

4 mai 2014

Table des matières

Chapitre 1

Introduction du problème

1.1 Descente de gradient stochastique

La régression linéaire et logistique, les réseaux de neurones, utilisent ce que l'on appelle la descente de gradient. Cette méthode à notamment l'avantage d'avoir une convergence assurée dans le cas de l'optimisation d'une fonction convexe. Mais elle devient très lente dès que l'on a un jeu de donnée de taille significative. La descente de gradient stochastique permet d'accélérer la convergence, mais ne permet plus de garantir la convergence exact. En pratique la descente de gradient stochastique est largement utilisée. Formalisons un peu le problème. On cherche à minimiser une erreur de prédiction assortie d'une régularisation :

$$S(\theta) = \sum_i (\theta.x_i - y_i)^2/M + \|\theta\| \tag{1.1}$$

La valeur du gradient de cette fonction est :

$$G_S(\theta) = 2 * \sum_i \theta.x_i/M + 2 * \theta \tag{1.2}$$

La fonction S étant convexe, nous sommes assurés de converger. Pour calculer le vecteur, nous avons besoin de l'ensemble des x_i. La descente stochastique fait le pari qu'en utilisant seulement une petite partie des x_i on peut avoir une relativement bonne approximation du gradient, et que les erreurs de chaque itérations se compensant on convergera vers le minimum global. Cette descente de gradient dite stochastique car on choisit à chaque étape des x_i aléatoirement pour calculer une approximation du gradient a l'avantage d'être plus rapide à calculer. La descente de gradient (classique ou stochastique) peut facilement se paralléliser. En effet le gradient étant obtenu à partir d'une somme, on peut répartir les exemples de training entre différents cœurs, chaque cœurs s'occupant de calculer le gradient sur ses exemples (ou une partie). Il faut ensuite rassembler les sommes partielles afin d'obtenir le gradient puis updater le vecteur θ.

1.2 Système de recommandation collaborative

Un problème classique en machine learning est celui de recommandation collaborative. Ce problème est par exemple celui de Netflix : ayant un ensemble d'utilisateurs, qui ont regardé et noté des film comment faire des prédictions à tout les utilisateurs même à ceux ayant notés peu de films ? L'approche classique est la suivante, on essaye de résoudre simultanément deux problèmes : pour un film à quels utilisateurs va-t-il plaire et pour un utilisateur, quels film vont l'intéresser ? Cette fois on n'a plus un vecteur θ pour l'ensemble des utilisateurs mais un vecteur θ_j pour chaque utilisateur j. Chaque film i reçoit une note y_{ij} de la part de j. Bien que la matrice Y soit très grosse (pour Netflix plus de 10 millions d'utilisateurs et plus de 100 mille films), elle est essentiellement vide. En effet la plupart des utilisateurs ne notent que très peu de films. Notons : Ω l'ensemble des couples (i, j) correspondant aux films i notés par l'utilisateur j. Pour ce ramener à un problème classique on note Θ_j le vecteur de recommandation de l'utilisateur j, et X_i le vecteur de "features" du film i. Il s'agit donc maintenant de minimiser la fonction :

$$S(\Theta, X) = \frac{1}{|\Omega|} \sum_{i,j\in\Omega} (\Theta_j.X_i - y_{ij})^2 + \sum_j \|\Theta_j\| + \sum_i \|X_i\| \tag{1.3}$$

Cette fonction de coup étant toujours convexe, on peut envisager d'utiliser une descente de gradient. La question est de savoir quelle taille affecter au vecteur de features des films. Celui-ci est critique car le nombre d'opérations dépend linéairement de ce rang. La première remarque est que la matrice Y des notes données a un rang très faible. En effet si deux utilisateurs aime tout les deux les films d'actions il est probable que tout les films d'action qu'ils ont vu aura probablement des notes proches. Si l'on suppose que les nombres de catégories de film et d'utilisateurs sont relativement limités, on voit que les matrice X et Θ n'ont pas besoin d'avoir beaucoup de lignes. L'évaluation de ce nombre de ligne r sera abordé plus loin.

1.3 Approches pour la recommandation collaboratives

L'expression de la fonction de coup peut se réécrire de la façon plus abstraite suivante :

$$S(W) = fW + \|W\| \tag{1.4}$$

Outre la descente de gradient, différentes méthodes mathématiques existe pour minimiser cette fonction. Celles-ci font un usage extensif du calcul des valeurs singulières de la matrice W, ce qui rend le processus long et inapplicable pour des problèmes avec des dimensions comme celles des bases de données de Netflix.

...

Chapitre 2

Méthode proposée

Le papier "Parallel Stochastic Gradient Algorithms for Large-Scale Matrix Completion" de Benjamin Recht & Christopher Ré propose une méthode, Jellyfish, pour implémenter une descente de gradient stochastique parallèle de façon efficace.

La principale source d'inefficacité dans les programmes parallélisés vient souvent de l'utilisation extensive de verrou sur des données qui doivent pouvoir être modifiés par différents "threads". Cette méthode fait donc attention à ne pas mettre de verrou sur les matrices Θ et X calculées par le programme.

A chaque étape la descente de gradient stochastique améliore simultanément les vecteursΘ_j et X_i de la façon suivante :

$$X_i = X_i(1 - \mu_i \alpha) - 2\alpha\Theta_j(\Theta_j.X_j - Y_{ij}) \tag{2.1}$$

$$\Theta_j = \Theta_j(1 - \mu_j \alpha) - 2\alpha X_i(\Theta_j.X_i - Y_{ij}) \tag{2.2}$$

Il est donc possible de modifier simultanément les vecteurs (Θ_j, X_i) d'une part et les vecteurs $(\Theta_{j'}, X_{i'})$ d'autre part. Pour une efficacité optimale, Jellyfish, réparti donc les indices (i, j) entre les différents cœurs. Le principe est simple à chaque "epoch" un des cœurs découpe de façon aléatoire les i en $(P - 1)$ parties :$I_0, I_1, ..., I_{P-2}$, où P est le nombre de coeurs disponible. Le dernier coeur est réservé pour cette répartition des indices. De même les j sont répartis en $J_0, J_1, ..., J_{P-2}$. L'epoch va être divisée en $P - 1$ rounds. Au premier round le processeur 0 reçoit les indices $I_0 x J_0$, le processeur 1 les indices $I_1 x J_1$... Au second round le processeur 0 reçoit les indices $I_0 x J_1$, le processeur 1 les indices $I_1 x J_2$... Ceci garanti que à la fin de l'epoch, tout les couples (i, j) auront été traités, par conséquent toute l'information contenue dans la matrice de notation Y aura été utilisée. De plus durant chaque round les différents processeur traitent chacun des portions différentes de Θ et de X, ce qui nous évite de mettre des verrous sur ces tableaux constamment accédés en lecture et écriture. Les $P-1$ threads qui modifie les matrices Θ et X, et le thread qui distribue les indices sont synchronisé à chaque fin de round. Il est important que ce dernier ne ralentisse pas l'exécution des autres.

Table des figures

Liste des tableaux