Éléments logiciels pour le traitement de données massives
Rapport final
5 février 2017

Theano : kmeans & color quantization

Mehdi Miah
Duc-Vinh Tran

Enseignants :
Xavier Dupré
Matthieu Durut

Résumé : La color quantization consiste en la représentation d'une image à travers une petite palette de couleurs. Cela peut se faire à travers l'algorithme Kmeans, dont une version a été implémentée avec le framework Theano, utilisant la carte graphique. Les résultats obtenues montrent que l'utilisation de la GPU permet de réduire le temps d'exécution par dix. Cependant, un autre problème survient : la gestion mémoire, qui apparaît lors de la reconstruction de l'image. **Mots clés : Theano, kmeans, color quantization, big data**

1 Introduction

Le clustering est un apprentissage non supervisé : ceci consiste en la création de groupes d'objets ayant une similarité. Ce problème a une complexité exponentielle : il n'est pas possible de le résoudre de manière optimale en un temps raisonnable. C'est pour cela qu'il est d'usage d'avoir recours à des heuristiques permettant l'obtention d'un résultat approché.

Un des algorithmes les plus connus pour effectuer cela est le k-means.

Appliqué à une image (chaque pixel de l'image est représenté par un vecteur de dimension 3, correspondant aux 3 couleurs RGB), cela permet de construire une nouvelle image contenant uniquement K couleurs. L'obtention de ces K couleurs se fera automatiquement et différera d'une image à l'autre.

2 Application

Notre objectif étant d'appliquer un color quantization sur une image, et donc sur possiblement des millions de points, il est important d'utiliser des implémentations compatibles avec de telles données. Pour cela, nous avons premièrement travaillé sur un petit dataset en ayant recours à un algorithme combinatoire sous `numpy`, puis à la version distribuée par `scikit-learn`. Et enfin, dans le contexte des big data, nous avons implémenté une version à travers le framework `Theano` qui sollicite aussi bien le processeur (CPU) que la carte graphique (GPU).

2.1 Application à de petits datasets

Il s'est avéré rapidement que la version combinatoire, et donc naïve, implémentée sous `numpy` était bien plus lente par rapport à la version de `scikit-learn`. Ces résultats étaient, à vrai dire, attendus.

Ensuite, l'implémentation sous `Theano`, sous CPU et sous GPU nous ont montré que l'utilisation de la GPU rend un résultat 10 fois plus rapidement que la CPU.

2.2 Application à de grands datasets

Une fois ces premiers résultats obtenus, l'adaptation à une image a été implémentée : une image possédant évidemment plus de points et se "décrivant" par bien plus de couleurs, cela a complexifié la tâche par :

— un plus grand nombre de points ;
— un plus grand nombre de clusters

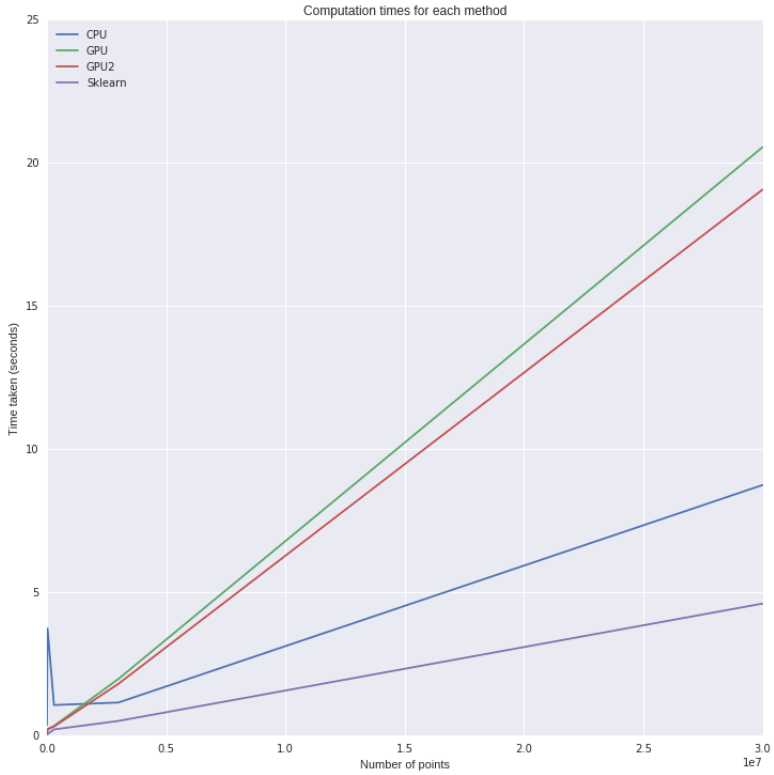
2.3 Application à de petits datasets avec un nombre de points croissant

En comparant les modèles sur un dataset contenant un nombre croissant de points, avec toujours trois clusters, il a été montré que la complexité était linéaire. De plus, l'utilisation de la GPU ralentit le processus. Notre hypothèse est les fonctions de mise à jour sous Theano tournent sur le CPU. Cela a pour conséquence un passage incessant entre la CPU et la GPU.

2.4 Application à des images avec un nombre de clusters croissants

En comparant les modèles sur la color quantization d'une image à travers un nombre croissant de clusters, il a été montré que l'utilisation de la GPU face à la CPU était d'autant plus visible que le nombre de clusters était grand.

FIGURE 1 – Temps d'exécution en fonction du nombre de points



3 Difficultés rencontrées

Dans le cadre du cours *Éléments logiciels pour le traitement de données massives*, nous avions tout d'abord souhaité utiliser le langage `Cuda` pour implémenter une version sous GPU. Cependant la maintenance du maintenance du package `numba` et `numbapro` nous a contraint a développé sous un autre framework.

C'est ainsi que nous avons fait le choix de `Theano`. En effet, ce langage permet une implémentation intuitive de la descente de gradient. Toutefois, disposant d'un ordinateur sous Linux et d'un autre sous Windows, les installations ne furent pas aussi simple l'une que l'autre. Tandis que l'installation de `Theano` et de `Cuda` pour l'implémentation sous GPU fut rapide sous Linux, le groupe a éprouvé des difficultés à utiliser le GPU avec `Theano` sous Windows. En fin de compte, le code a été developpé sur les deux terminaux, dont l'un disposait d'un accès à la GPU et l'autre non.

La version GPU2 a été faite pour optimiser la gestion de la mémoire.

4 Conclusion

L'utilisation sur un petit dataset a montré que les algorithmes pouvant concurrencer `scikit-learn`, exécutant du `C`, devait faire appel à la GPU. Toutefois, lors de l'application à de grandes images, il s'est avéré que la GPU ne montrait pas l'ensemble de ces capacités. Faute à une mauvaise implémentation sous `Theano` lors des updates ?

De plus, l'appel à de grandes images nécessite davantage de mémoire RAM. Or, autant l'algorithme Kmeans peut se faire sur du GPU, autant la reconstruction de l'image est longue.

FIGURE 2 – Temps d'exécution en fonction du nombre de clusters