Optimisation et parallélisation des réseaux de neurones

Pather Stevenson et Benedictus Kent Rachmat

Enseignant : M. Fortin Pierre

L3 ICHP - Semestre VI - 2022

Table des matières

Introduction

La nature numérique des ordinateurs a pour conséquences sur les méthodes de calcul traditionnelles qu'elles soient hautement structurées et linéaires. Ces méthodes sont généralement très efficaces pour résoudres certains types de problèmes que l'on retrouve notamment dans des modèles mathématiques, simulations physiques ou certaines analyses en microbiologie comme les protéines. Les méthodes classiques sont quant à elles pas très efficaces dans la résolution de problèmes du type reconnaissance d'images, filtrage du spam ou encore dans l'apprentissage adaptatif.

Tout comme les réseaux de neurones biologiques, les réseaux de neurones artificiels sont également des systèmes connexionniste largement parallèles, capable d'apprendre et de généraliser. Le parallèlisme naturel des réseaux permet une implémentation distrubée des réseaux de neurones artificiels et de fonctionner en parallèle. Ce qui résulte théoriquement en une amélioration des performances.

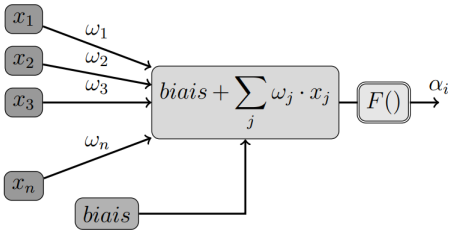
Dans ce document nous présenterons les réseaux de neurones artificiels et l'intérêt de la parallélisation et distribution de ceux-ci. Puis nous décrirons les différents niveaux et échelles de parallélisation. Ensuite nous discuterons du choix d'architecture entre GPU et CPU et leurs avantages en fonction du contexte.

Les réseaux de neurones artificiels

les réseaux de neurones artificiels reprennent la modélisation des neurones biologiques. Ils imitent le fonctionnement de ceux-ci dans le but de résoudre des problématiques d'apprentissage machine. Ce qui s'avère être plus performant que l'utilisation des techniques de régressions pour des tâches d'apprentissage automatique.

2.1 Le neurone, synapse et l'activation

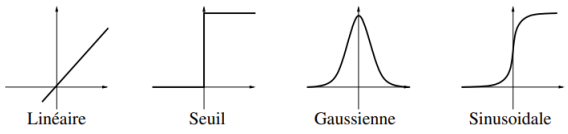
Introduit par les travaux de Warren S. McCulloch et Walter Pitts en 1943, le neurone artificiel est la composante de base d'un réseau de neurones. Il définit un modèle mathématique simplifié d'un neurone biologique.



Comme on peut le voir sur le neurone formel ci-dessus, celui-ci comprend une série d'entrées et d'une sortie. Les entrées x_j sont, de manière indépendante, pondérées (w_{ij} puis leur somme et une valeur de seuillage noté θ_i sont utilisées pour la valeur de sortie par le biais de la fonction d'activation noté F. Ainsi l'on peut déterminer cette sortie de la façon suivante :

$$a_i = F\Big(\sum_{j=1}^{N} w_{ij} x_j - \theta_i\Big) \tag{2.1}$$

Le choix de la fonction d'activation doit être fait en fonction du domaine d'application mais aussi de la position du neurone dans le réseau. Le choix de celle-ci influe sur les résultats. Voici quelques exemples de fonctions d'activation :



2.2 Les types

La caractéristique qui permet de classifier en deux grandes familles les types de réseaux de neurones est l'interconnexion. Ainsi l'on retrouve les réseaux récurrents qui sont des graphes de connexion avec boucles (réseau de Hopfield, réseau de Elman, ...) et les réseaux à propagation feed-forward qui sont des graphes de connexion sans boucles (réseau linéaire, perceptron multicouches, ...). Nous povuons spécifier nos types de la façon suivante :

— Multicouches feedforward : réseau en couches où chaque couches de neurones est l'entrée de la couche suivante

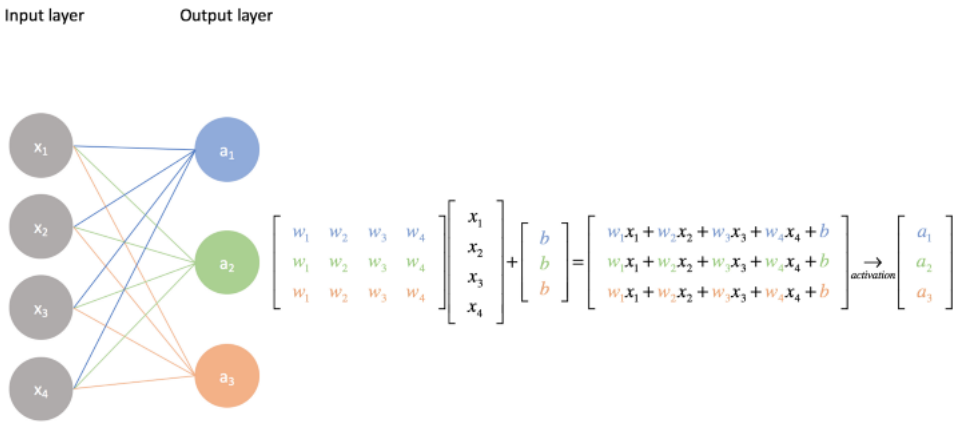
— feedback : Une seule couche de neurones avec des sorties qui bouclent sur les entrées

— Cartes auto-organisées (SOM) : forme des mappages d'un espace de grande dimension vers un espace de dimension inférieur

— Memoire distribuée fragmentée (SDM) : forme spéciale d'un réseau feed-forward à deux couches qui fonctionnent comme une mémoire associative

2.3 Considération

La topologie des réseaux de neurones décrit comment les neurones s'organisent les uns par rapport aux autres. La plupart des réseaux de neurones ont une topologie très parallélisable. Voici un exemple simple de topologie de réseau de neurones :



L'opération principalement effectuée dans les réseaux de neurones est la multiplication de matrices-vecteurs. Ce qui est donc très intéressant pour permettre leur parallélisation. Par exemple pour une couche de $N - node$ il y a N^2 multiplications scalaires et N sommes de N nombres.

Parallélisation dans les réseaux de neurones

Dans les réseaux de neurones, il existe des millions de paramètres qui définissent les modèles et qui nécessiste de grandes quantités de données pour pouvoir connaître ces paramtères à partir des données. C'est un processus lourd et coûteux en temps qui peut prendre plusieurs jours pour entraîner un réseau de neurones sur un seul CPU single core. Le jeu de données peut également être trop grand pour être stocker dans une seule machine.

C'est pourquoi il est important et intéressant de chercher à paralléliser et distruber les algorithmes des réseaux de neurones qui pourront s'exécuter beaucoup plus rapidement et permettre de réduire de manière significative le temps de d'apprentissage de ceux-ci.

On dit que les réseaux de neurones sont *embarrassingly parallel* car les calculs pour chaque noeuds sont généralement indépendants de tous les autres noeuds du même niveau.

3.1 Structure

Voici la structure typique de manipulation d'un réseau de neurones :

- Pour chaque session d'apprentissage :
  - Pour chaque exemples d'apprentissage de la session :
    - Pour chaque couche (forward & backward) :
      - Pour chaque neurone de la couche :
        - Pour chaque poids du neurone :

- Pour chaque bits de la valeur du poids

3.2 Les niveaux de paralléisation

Ce qui nous permet de conclure sur le fait qu'il y est 6 façons de paralléliser, en partant du grain le plus gros au grain le plus fin :

- Parallélisme des sessions d'apprentissages :

- Parallélisme des exemples d'apprentissage :

- Parallélisme des couches/Forward-Backward :

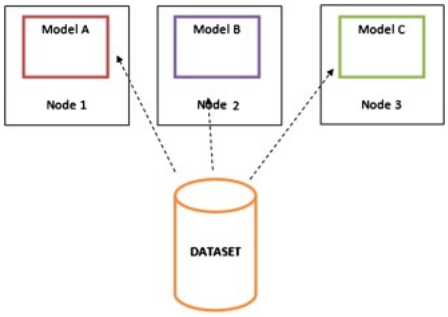
- Parallélisme des neurones :

- Parallélisme des poids :

- Parallélisme des bits

3.2.1 Inter-modèle

Généralement, lorsque le parallélisme inter-modèle est utilisé, il manipule alors différents modèles. Et chaque modèle peut avoir différents paramètres tels que la fonction d'équation, les types de couches, le nombre de neurones par couche, etc. Par exemple dans l'illustraion suivanten, les trois cas de modèles différents sont formés avec le même ensemble de données :

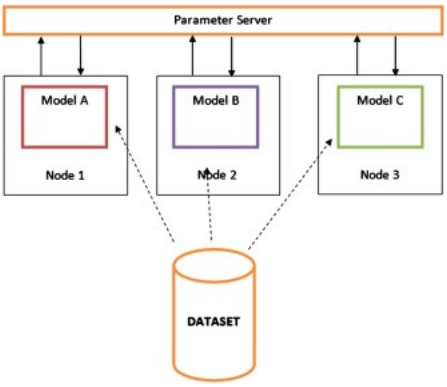


3.2.2 Intra-modèle

L'apprentissage automatique peut tirer parti des plates-formes modernes de traitement de données parallèles telles que Hadoop et Spark de plusieurs

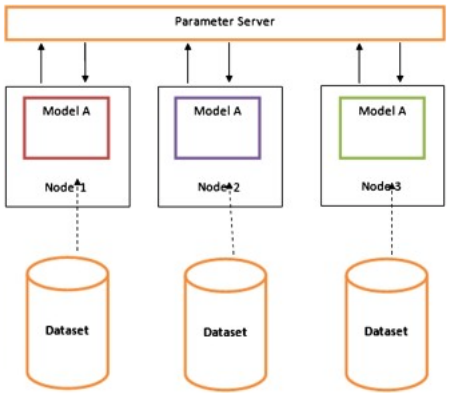
manières. Nous verrons comment mettre à l'échelle l'apprentissage automatique avec Hadoop ou Spark.

Trois méthodes différentes de traitement parallèle peuvent bénéficier à l'apprentissage automatique. Lorsque l'on pense au traitement parallèle dans le contexte de l'apprentissage automatique, ce qui nous vient immédiatement à l'esprit, c'est le partitionnement des données ainsi que les algorithmes d'apprentissage diviser pour mieux régner. Cependant, comme nous allons le découvrir, le partitionnement des données n'est pas la meilleure méthode. Le meilleur moyen est d'exploiter le traitement parallèle :



3.2.3 Data

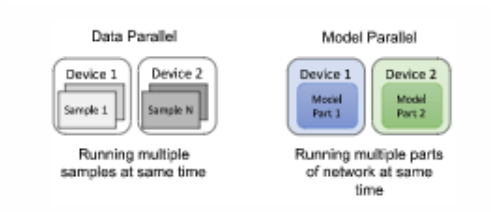
L'idée du parallélisme des données a été évoquée par Jeff Dean en tant que moyenne des paramètres. Si nous avons trois exemplaires du même modèle. Nous déployons le même modèle A sur trois noeuds différents, et un sous-ensemble des données est alimenté sur les trois modèles identiques. Les valeurs des paramètres sont envoyées au serveur de paramètres et, après avoir collecté tous les paramètres, la moyenne est effectuée. À l'aide du serveur de paramètres, les données sont synchronisées. Les réseaux de neurones peuvent être formés en parallèle de deux manières, c'est-à-dire de manière synchrone (en attendant une itération complète et en mettant à jour les données) et de manière asynchrone (en envoyant des paramètres obsolètes hors du réseau). Mais le temps nécessaire pour les deux méthodes est le même, et le choix de la méthode n'est pas problème ici.



Pour surmonter les problèmes de parallélisme des données, le parallélisme au niveau des tâches a été introduit. Les tâches de calcul indépendantes sont traitées en parallèle à l'aide des instructions conditionnelles dans les GPU. Le parallélisme au niveau des tâches ne peut agir sans l'aide du parallélisme des données que dans une certaine mesure, au-delà de laquelle le GPU a besoin du parallélisme des données pour une meilleure efficacité. Mais le parallélisme au niveau des tâches donne plus de flexibilité et d'accélération des calculs aux GPU.

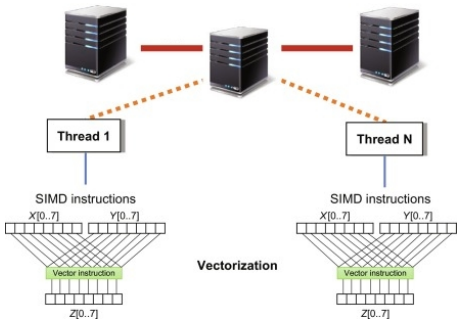
3.2.4 Data et modèle

Plusieurs restrictions existent à la fois dans le parallélisme des données et le parallélisme des modèles. Pour le parallélisme des données, nous devons réduire le taux d'apprentissage pour maintenir un processus de formation fluide s'il y a trop de nœuds de calcul. Pour le parallélisme des modèles, les performances du réseau seront considérablement réduites pour des raisons de dépenses de communication si nous avons trop de nœuds.



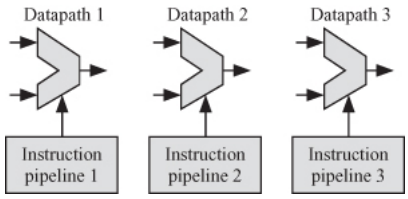
3.2.5 Tâche

un problème spécifique peut être décomposé en tâches plus petites et résolu simultanément par différents éléments de processus (processeurs, threads, etc.).



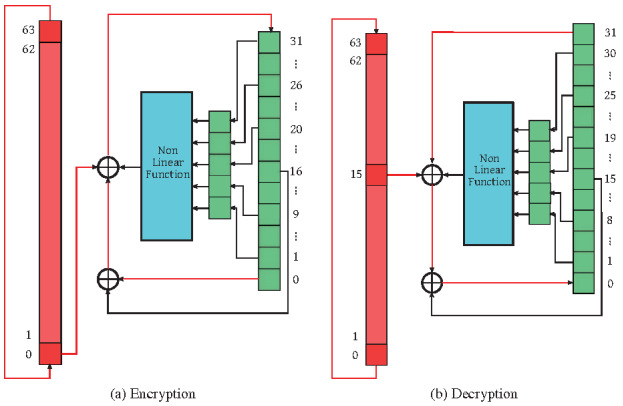
3.2.6 Instruction

les instructions d'un programme sont réordonnées et regroupées pour une exécution parallèle. Les processeurs modernes incluent des structures appelées pipelines qui permettent de diviser l'exécution des instructions en segments. Cela permet à différentes étapes de plusieurs instructions d'être exécutées en même temps.



3.2.7 Bits

S'organise pour se référencer à la taille des données avec lesquelles le processeur peut travailler. Par exemple, un processeur avec une taille de mot de 32 bits peut effectuer simultanément 4 additions indépendantes de 1 octet. Si la taille du processeur est de 1 octet, il devra effectuer 4 opérations.



3.3 Choix du degré de parallélisation en fonction du réseau

Il est ainsi important de mesurer le niveau de parallélisation possible en fonction du type de réseaux de neurones utilisés.

La structure multicouche de nombreux réseaux feedfoward et leur technique d'apprentissage avec rétroprogration limite l'accélération apportée par une parallélisation des calculs au niveau des neurones. Une parallélisation au niveau des réseaux ou des ensembles d'apprentissage sera alors préférable.

La paralléisation au niveau des neurones est, elle, plus efficace pour des réseaux à une seule couche comme par exemple les SOM ou certains réseaux récurrents.

3.4 Communication

Les réseaux de neurones ont un degré élevé de connectivité de par leur structure et ils manipulent des flux de données importants. De fait les structures de communications et bandes passantes sont très importantes au niveau matériel.

Choix d'architecture GPU/CPU

Un GPU est un processeur monopuce utilisé pour des calculs graphiques et mathématiques de grandes tailles. Au sein d'une machine celui-ci permet ainsi de libérer des cycles CPU pour d'autres tâches.

La différence notable entre les GPU et les CPU est la proportion de transistors dédiés aux unités de logiques arithmétique qui est plus élevées dans les GPU mais qui est donc moins élevés pour le cache et pour le contrôle de flux. Ainsi les GPU ont tendances à avoir plus de coeurs logiques. Théoriquement les GPU peuvent effectuer un certains nombres d'opérations en plus que les CPU.

Les GPU sont très intéressant dans l'apprentissage des réseaux de neurones car ils peuvent traiter plusieurs calculs simultanément. Le nombre de coeurs de ceux-ci permet un meilleur calcul de plusieurs processus parallèles. Et le grand nombre de données manipulées rendent la bande passante mémoire d'un GPU plus intéressante.

Les paramètres qui rendre donc en compte dans le choix entre GPU et CPU sont la bande passante mémoire, la taille du jeu de données ou encore l'optimisation des tâches.

4.1 Optimisation

L'optimisation des tâches est plus simple au sein d'un CPU que d'un GPU. De plus les coeurs de CPU sont bien plus puissants que ceux des GPU malgré la différence du nombreux de coeurs.

L'architecture MIMD des CPU permet l'exécution d'instructions différentes contraitrement aux GPU qui ont une architecture SIMD et qui donc exécutent la même instruction à un instant t en parallèle.

Permettre la parallélisation de réseaux de neurones volumineux est compliqué de fait les manières d'optimisations complexes sont également difficiles à mettre

en pratique au sein d'un GPU.

Ainsi dans le cas de réseaux de neurones de petites tailles l'utilisation d'un GPU n'est pas forcément nécessaire. Mais dans le cas où cela implique un grand nombre de calculs et donc un très grand nombre de paramètres alors l'apport en performance d'un GPU est intéressante.

Bibliographie

— https://www.sciencedirect.com/topics/computer-science/task-level-parallelism

— https://cerebras.net/blog/data-model-pipeline-parallel-training-neural-networks/

— https://www.teldat.com/blog/parallel-computing-bit-instruction-task-level-parallelism

— 2018 Swiss HPC Conference, Torsten Hoefler from (ETH) Zürich :
https://www.youtube.com/watch?v=xtxxLWZznBI

— http://documents.irevues.inist.fr/bitstream/handle/2042/12370/
AR19_8.pdf?sequence=1

— https://web.stanford.edu/~rezab/classes/cme323/S16/projects_reports/
hedge_usmani.pdf

— https://www.researchgate.net/publication/322149433_Parallel_Implementation_
of_a_Machine_Learning_Algorithm_on_GPU

— https://www.researchgate.net/publication/221165763_Artificial_
Neural_Networks_on_Massively_Parallel_Computer_Hardware

— https://tel.archives-ouvertes.fr/tel-00337399/document