Filière informatique
Enseirb-Matmeca

Environnement pour l'apprentissage Keras Model Pruning

Zghari Hicham
Boudali Mohamed

1 Contexte

De nos jours, l'utilisation de certaines technologies comme *Machine learning* est inévitable. Ces technologies sont basées sur ce que nous appelons "réseau de neurone". Cependant, ces réseaux sont de plus en plus lourds, vu la complexité de certains systèmes informatique (Face ID sur Apple par exemple). Le but de notre projet est de chercher un moyen d'optimisation efficace pour réduire la taille de ces réseaux, tout en gardant une bonne précision et stabilité.

La première intuition face à un réseau lourd que l'on doit embarquer consiste tout simplement à l'alléger "Pruning".

2 Introduction

Il existe plusieurs méthodes pour optimiser un réseau de neurone. Une méthode connue est d'élargir le réseau en supprimant certaines connexions entre les neurones et les couches, ainsi que de réduire le nombre global de paramètres afin d'accélérer le calcul.

La figure suivante explique le principe d'élagage "Pruning". Chaque neurone du réseau possède une connexion avec la couche suivante, ce qui rend le calcul complexe. L'optimisation consiste à connecter chaque neurone, à quelque autres neurones pour éviter le nombre énorme de multiplication. Le réseau résultant s'appelle "Sparse Network".

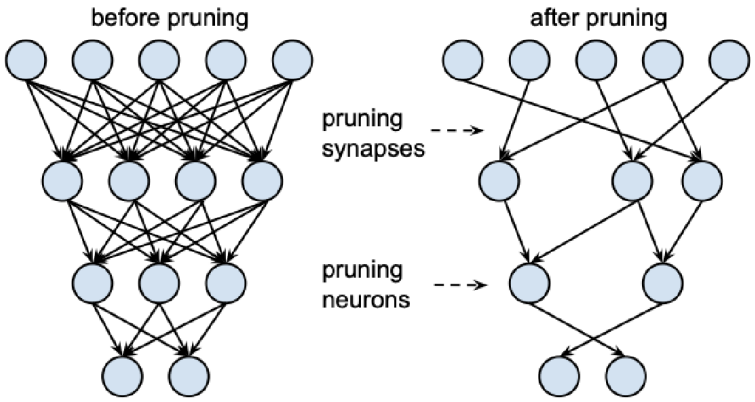


Figure 1 : Exemple d'élagage

Généralement, il existe deux méthodes d'élagage "Pruning" des réseaux de neurones :
-La première méthode s'appelle "Weight Pruning". Cette méthode change la valeur du poids individuel dans la matrice des poids à 0. Cela se traduit par la suppression des connexions dans la figure ci-dessus. Pour arriver à une parcimonie (Sparcity) d'ordre K, nous évaluons les poids individuels dans la matrice des poids, et nous mettons à zéro les K premières petites valeurs.
-La deuxième méthode s'appelle "Unit/Neuron Pruning". Cette méthode remplace une colonne entière par des zéros dans la matrice des poids, cela signifie la suppression d'un neurone.
Pour arriver à une parcimonie d'ordre K, nous supprimons les K premières petites colonnes en calculant leur norme 2 : $|M| = \sqrt{\sum_{i=1}^{N}(p_i)^2}$.

3 Base de données

Nous avons utilisé **MNIST** qui représente une base de données contenant des chiffres écrits à la main.

Figure 2 : Exemple de chiffres manuscrits

Le but est d'entraîner un modèle de reconnaissance de l'écriture manuscrite basé sur un réseau de neurone, afin de tester par la suite ces performances en utilisant d'une part la totalité du réseau, et d'une autre part un réseau optimiser "**Pruned**".

Finalement, nous avons découpé notre ensemble de données en deux parties, une première partie d'apprentissage (de taille 60000) et une autre de test(de taille 10000).

4 Apprentissage

Dans cette partie, nous avons utilisé le modèle *Keras* dans la bibliothèque *Tensorflow* pour entraîner le modèle de *Keras*. Premièrement, nous avons créé un réseau de neurone *ReLU* à 4 couches denses et complètement connectées avec les tailles suivantes : 1000, 1000, 500 et 200. Une cinquième couche pour les logs résultants (de taille 10).
Ensuite, nous avons entraîné le modèle que nous avons créé avec les données de la base **MNIST** que nous avons chargé.

Les résultats de cet apprentissage sur 10 époques sont les suivants :

Précision	Perte	Temps
0.9835	0.0673	70s 55ms

5 Élagage (Pruning)

Après l'apprentissage de notre modèle, nous allons maintenant l'optimiser en utilisant les deux méthodes vues en introduction (Weight pruning/Neuron pruning).

Nous avons commencé par la création d'une fonction qui prend en paramètres deux matrices de poids, et retourne une version creuse de ces matrices (en indiquant la méthode weight/neuron pruning) avec une parcimonie (sparsity) K. Puis, cette fonction nous a permis de prendre un réseau de neurone dense, une sparsity et une méthode (weight/neuron), et retourner une copie creuse du réseau.

5.1 Résultats

Nous avons tracé un premier graphe pour regarder l'effet du paramètre K (sparsity) sur les pertes du modèle en utilisant les deux méthodes de pruning, le résultats est le suivant :

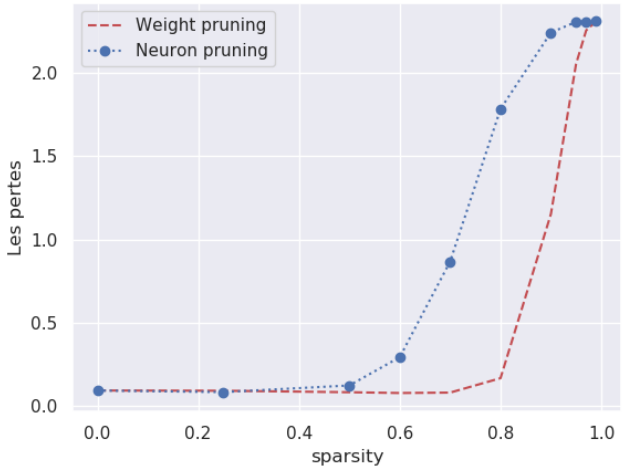


Figure 3 : Graphe de pertes pour les deux méthodes de pruning

Nous remarquons l'augmentation de pertes avec le paramètres K (pertes=2.3, k=0.99, méthode=weight), ce qui est totalement logique parce que nous réduisons le réseau. Nous remarquons aussi que la variation des pertes de la méthode Neuron pruning est plus rapide que weight pruning, car cette première méthode change une colonne toute entière contrairement à la deuxième méthode qui change que des connexions.

Nous nous sommes intéressés au graphe de précision aussi, pour étudier l'effet du paramètres K :

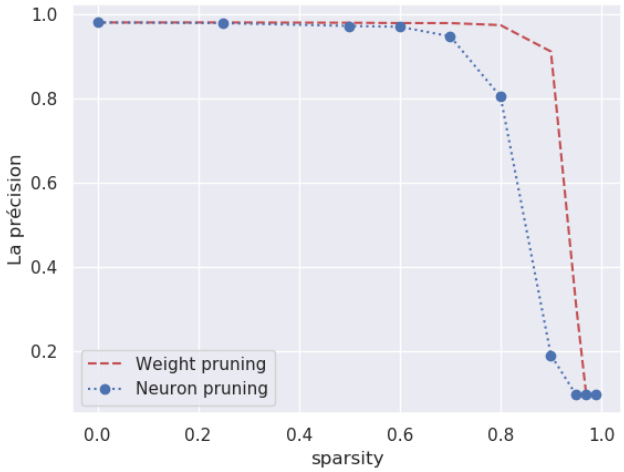


Figure 4 : Graphe de précision pour les deux méthodes de pruning

La précision est une fonction décroissante par rapport à K pour les deux méthodes, ce fait est du à l'utilisation de moins de paramètres dans notre réseau de neurone. Pour K=0.6, nous obtenons une précision de 0.981. Comparant cette valeur à la valeur 0.982 quand K=1, nous déduisons que ce grand changement de sparsity n'a pas affecté la précision de notre modèle, alors que d'autre part le réseau a été optimisé, d'où l'efficacité de ces méthodes.

Une autre expérience était de visualiser le temps d'exécution. Nous pouvons voir que le temps diminue avec K, alors nous avons réussi à optimiser le réseau au niveau du temps.

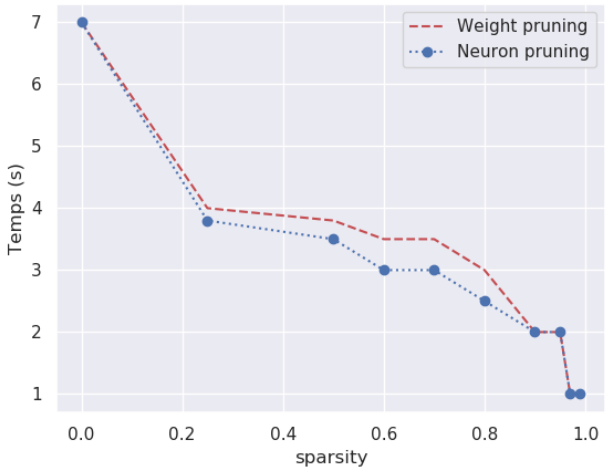
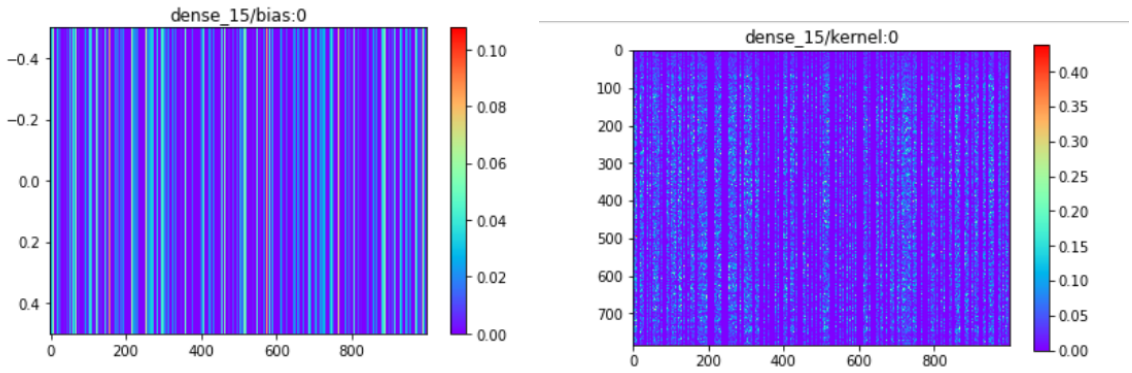


Figure 5 : Graphe de temps d'exécution pour les deux méthodes de pruning

Nous pouvons ainsi regardé les matrices creuses de poids pour k=0.5. Sur les graphes au dessous, nous avons tracé les matrices des deux méthodes en représentant les zéros avec la couleur blanche. Nous remarquons qu'il y a effectivement des colonnes blanches sur le deuxième graphe correspondant à la méthode "Neuron pruning".



(a) weight pruning **(b)** Neuron pruning

Figure 6 : Graphes des matrices de poids

6 Conclusion

En guise de conclusion, l'optimisation des réseaux de neurones en utilisant les méthodes de pruning a montré son efficacité à travers plusieurs tests, de plus pour un sparsity K autour de 0.6, nous ne perdons pas beaucoup de précision, alors que nous gagnons beaucoup d'optimisation et rapidité.

7 Bibliographie

-https ://colab.research.google.com/drive/102oKvefYhr-jrkJqR8d0wLGJmD9gNhpd

-https ://moodle.bordeaux-inp.fr/pluginfile.php/191449/mod_resource/content/2/cours67.pdf