Deep Learning Project - Master IASD

Théo Delemazure - Guillaume Bressan

Janvier 2020

1 Mise en place du projet

Le but du projet est de construire un réseau de neurones de moins d'un million de paramètres qui imite au mieux le comportement du réseau *AlphaGo* de *DeepMind*. Nous avons pour cela utilisé les données générées par *golois* sur les **GPU** de *Google* grâce à *Google Colab*. L'énoncé demandait de faire le réseau en *Keras*.

2 Réseau

Notre réseau le plus performant est consultable en Figure 1. Il est inspiré du réseau *alphago* et il est séparé en trois parties :

- Les entrées passent d'abord par une succession de couches convolutives *conv2D* avec un *dropout* de 0.2 (afin d'éviter l'*overfit*) et une couche résiduelle (*add*) après chaque *conv2D*. Les *layers* on 32 *channels* et des filtres de taille 3×3.
- Le résultat de la première partie est dupliqué et passe dans un réseau avec des couches convolutives résiduelles croisées : c'est-à-dire qu'il y a deux chemins en parallèle et l'entrée de l'un arrive en résidu sur la sortie de l'autre. Les layers ont 32 *channels* et des filtres de taille 3×3. Il y a également du *dropout*.
- Enfin, les résultats sont rassemblés par concaténation et dupliqués, pour finir dans deux têtes : celle dédiée à la *policy* et celle dédiée à la *value*. Chaque tête contient une couche de *Conv2D* avec très peu de *channels* de sortie et quelques couches denses.

Il est à noter que nous avons tenté beaucoup d'autres structures, mais aucune n'a été très convaincante. Nous avons par exemple modifié les entrées pour donner un poids négatif aux pions noirs et positifs aux pions blancs, nous avons également tenté différents hyperparamètres (tailles de filtres, nombre de filtres, etc.).
Une autre approche a été testée, elle est basée sur une vision micro/macro : le réseau se sépare dès la deuxième couche en deux branches, l'une avec des couches de convolution utilisant des filtres de petite taille (micro) et l'autre avec des filtres de grande taille (macro) ; les sorties des branches sont ensuite multipliée puis passent dans des couches de convolution. La *loss* de la *policy* descend jusqu'à 3.3 et l'accuracy de la *policy* atteint 20

3 Entraînement

Nous entrainons nos modèles avec l'optimiseur *Adam* qui semble plus efficace que *SGD*. Nous entrainons le réseau en deux temps :

1. Tout d'abord, on entraîne le réseau sur une dizaine d'epochs avec un *learning rate* de 10^{-3} jusqu'à atteindre environ 28% de *policy_acc* et 3.5 de *loss*. L'entraînement se fait sur 500000 parties générées par *golois*
2. Puis, soit en utilisant le paramètre *decay* dans l'optimiseur, soit à la main, nous diminuons petit à petit le learning rate. Nous entrainions sur plusieurs set de 100000 nouvelles données pendant 2 ou 3 epochs.

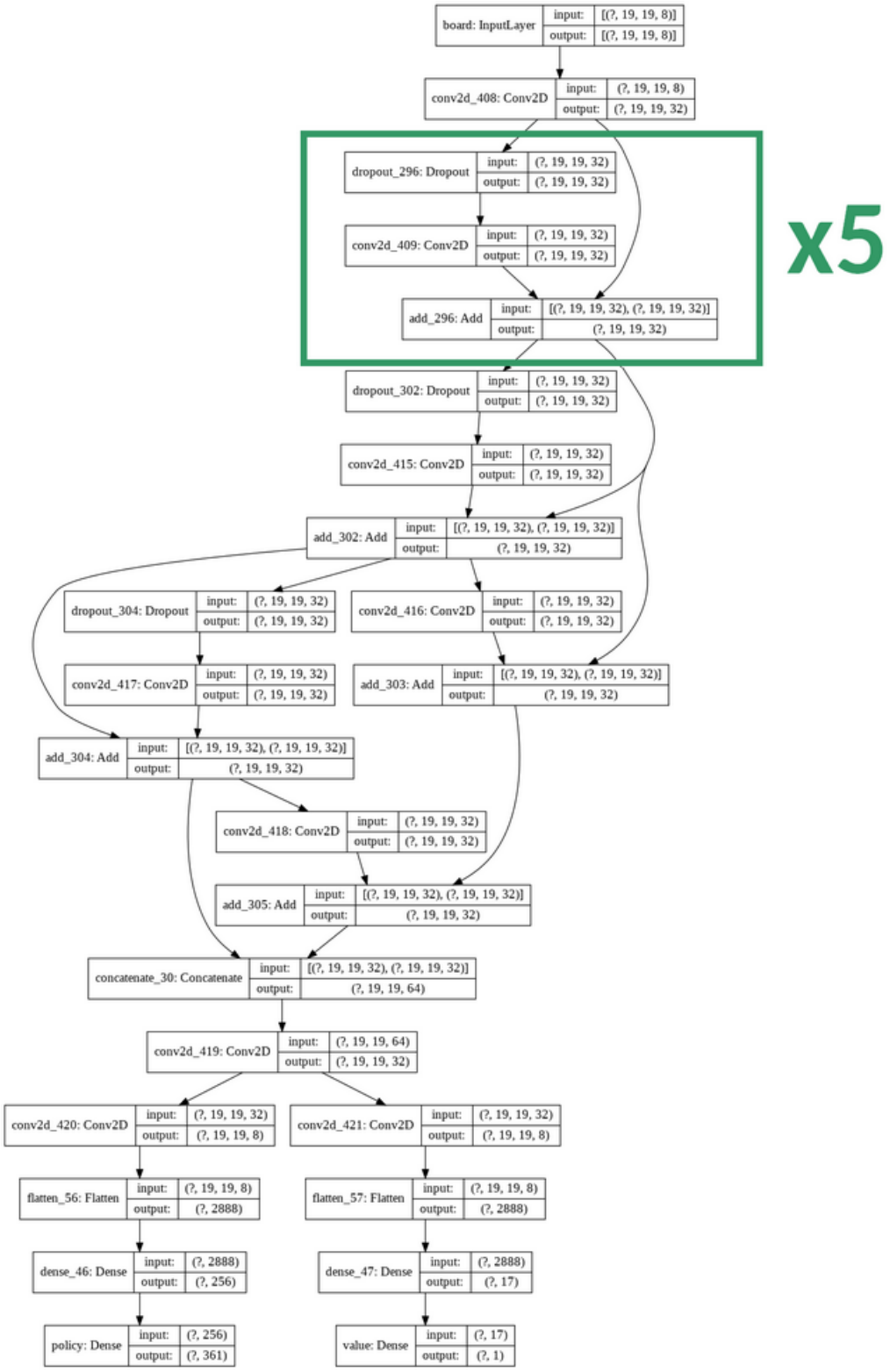

Figure 1: Notre réseau le plus performant

3. Nous changeons parfois le poids de la *value* dans la *loss* pour diminuer celle-ci, car autrement elle ne bouge pas.

Après ces deux étapes, nous obtenons au mieux une *policy_accuracy* de 33%, une *value_accuracy* de 64% et une loss d'environ 3.