UNIVERSITÉ PARIS-DAUPHINE

APPRENTISSAGE PROFOND

MASTER IASD

L'apprentissage profond pour le jeu de Go

Auteurs:
Emilie CHHEAN
Yves TRAN

March 21, 2021

Rapport de projet - L'apprentissage profond pour le jeu de Go

Emilie CHHEAN[1] **Yves TRAN**[1]

Abstract

Ce document est une synthèse concernant notre projet en apprentissage profond sur le jeu de Go. Durant notre projet, plusieurs architectures ont été mises en place pour tenter de développer le meilleur modèle sachant prédire le coup suivant (policy) et le gagnant de la partie (value) à partir d'un état donné tout en restant en deçà du million de paramètres. À l'issu de cette étude, deux architectures se sont démarquées se basant sur des modules Squeeze-and-Excitation ainsi que sur le mécanisme d'attention qui ont su battre nos modèles résiduels de référence.

1. Équipements

Afin de mener ce projet, nous nous sommes basé sur l'environnement proposé par Google Colab. Chaque instance dispose de 12.5GB de RAM, d'un CPU Intel Xeon et d'un GPU. Cette configuration nous a empêché d'importer l'intégralité des données. Par conséquent, entraînements et expériences se sont réalisés sur des batchs de $N = 100\,000$ instances de jeu à la fois.

2. Premiers essais

L'utilisation des blocs résiduels dans la construction de réseaux profonds a connu un grand succès pour les tâches visuelles [7]. Pour le jeu de Go, les modèles résiduels ont su augmenter les performances de jeu [4] et c'est ce qui a constitué notre point de départ. Nous avons commencé par des modèles ayant un maximum de blocs résiduels tout en respectant la contrainte du million de paramètres. Puisque, le modèle comporte deux sorties, il nous a semblé intéressant d'optimiser les performances en créant des branches indépendantes par sortie. Cependant, les performances étaient peu satisfaisantes comparées à un modèle comportant un tronc (partie du réseau partagé par les deux sorties) plus profond. Chaque sortie dispose de moins de paramètres car une partie du réseau est réservée à la deuxième sortie. De ce fait, il devient plus intéressant de se focaliser sur le tronc du réseau.

Afin de créer les modèles ayant davantage de largeur et de profondeur, nous avons opté pour des convolutions Depth-Wise qui appliquent l'opération de convolution pour chaque channel séparément. Nous avons utilisé ces couches pour créer des Inverted Residual Blocks [3] [9]. Ces couches nous ont notamment permis de créer des modèles plus profonds mais le coût du temps d'exécution qui était relativement long par rapport aux couches convolutives classiques nous a encouragé à explorer d'autres voies.

Ainsi, nous avons gardé l'emploi des couches convolutives classiques. L'ajout de couches BatchNormalization dans les blocs résiduels a permis une augmentation de la performance dès les premiers epochs.

Nous avons aussi tenté de modifier le bloc résiduel par un wide-dropout bloc, de Wide Residual Network [14]. Avec ces blocs, nous voulions les rendre aussi fins que possible en faveur d'une augmentation de leur profondeur et d'une diminution des paramètres. Cependant, la performance de nos modèles ne fut pas prometteuse et nous n'avons donc pas intégré ces blocs à notre architecture.

Nous avons témoigné un grand intérêt pour les blocs Squeeze-and-Excitation [15] qui permettent notamment de pondérer l'information par channel. En plus d'être peu coûteux en nombre de paramètres, cette méthode nous a permis de constater un gain de performance par rapport à nos modèles résiduels initiaux. Néanmoins, un biais se dessinait entre la précision en entrainement et en validation. Nous avons donc mis en place une couche de Dropout [10] ce qui a grandement affecté le modèle et un large gain de performance a été remarqué. De ce fait, cela a constitué un nouvel axe de recherche qui nous a mené aux blocs Convolutional Block Attention Module qui pondère l'information spatiale [8] dont la performance est semblable au modèle précédant.

En parallèle, plusieurs fonctions d'activation ont suscité notre intérêt. Comme alternative à la fonction ReLU, nous avons tenté GeLU (équation 1) 1 [6] qui a été utilisé dans des modèles profonds à succès [13] [11]. Dans notre cas, les performances s'étaient appauvries.

[1]Université Paris-Dauphine, Paris, Ile-de-France, France. Correspondence to: Tristan Cazenave.

Rapport de projet pour l'enseignement de l'apprentissage profond - Master IASD - 2020/2021.

$$GELU(x) = \frac{1}{2}x(1 + tanh(\frac{\sqrt{2}}{\pi}(x + 0.044715x^3))) \quad (1)$$

En revanche, la fonction Hard-swish (équation 2) 2 a permis un léger gain de performance. [1]

$$hswish(x) = x\frac{ReLU6(x+3)}{6} \quad (2)$$

Concernant l'optimizer, nous en avons testé : Adagrad, RMSProp, Adadelta [16], RAdam [5] et enfin Adam [2]. Ce dernier, Adam, nous donnait les meilleurs résultats.

3. Modèle final

3.1. Bloc d'entrée

Le module d'entrée est similaire à Golois4 [4] et se sépare de deux branches convolutives illustrées dans la figure 1. Nous avons remarqué qu'utiliser une convolution 3x3 permet d'obtenir une plus grande précision sur la politique après plusieurs epochs comparé à une convolution 1x1 qui réalise un plus fort démarrage mais avec une précision plus faible sur le long terme.

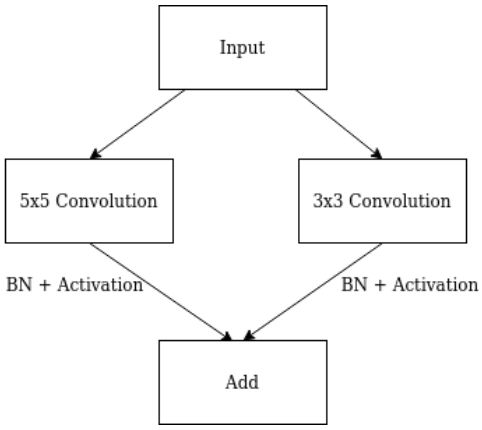

Figure 1. Bloc d'entrée prenant des plans de 19x19x21 en entrée et renvoie des plans de taille 19x19x190.

3.2. Corps du modèle

Les deux sorties du modèle partagent un tronc commun que l'on discutera dans cette section.

3.2.1. Bloc résiduel

Les blocs utilisés sont résiduels et sont présentés dans la figure 2. Nous avons constaté un gain de performance

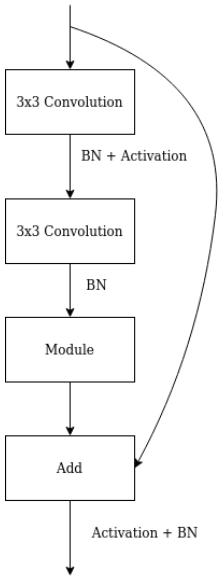

Figure 2. Bloc résiduel utilisé. La couche nommé "Module" désigne soit un bloc SE ou CBAM selon le modèle.

en ajoutant des couches de BatchNormalization (BN) [12] après les convolutions.

Ces blocs sont enchaînés à la suite pour constituer le tronc du modèle. Six blocs utilisés. La raison est expliquée dans la section suivante.

Comme annoncé précédemment, nous avons trouvé deux sous-blocs de performance quasi-équivalent entre-eux et améliorent celle d'un réseau résiduel simple. Il s'agit des sous-blocs Squeeze-and-Excitation (SE) et Convolutional Block Attention Module (CBAM). Nous avons donc deux modèles utilisant respectivement ces sous-blocs. Ces blocs ont l'avantage de contenir peu de paramètres.

3.2.2. Squeeze-and-Excitation

Notre module SE illustré dans la figure 3 dispose d'une couche de Dropout avec une probabilité de 0.2. Cet ajout a été introduit car nous avons constaté un biais entre les performance en entraînement et en validation. Cette couche a permis de réduire considérablement ce biais ce qui a grandement améliorer les performances. [10]

Ce bloc dispose d'un Average Pooling qui effectue la moyenne des valeurs par channel et renvoie un vecteur en 1 dimension. Il est suivi par une couche Dense qui permet de réduire la taille du vecteur. Nous avons choisi de le réduire par un facteur de 4, soit 23 neurones. La couche Dense suivante a pour but de rendre la taille initiale du vecteur et se compose de 90 neurones. Le fonction sigmoïde est utilisée pour que chaque valeur soit comprise entre 0 et 1. Chacune de ces valeurs vont se multiplier à un channel ce qui permet

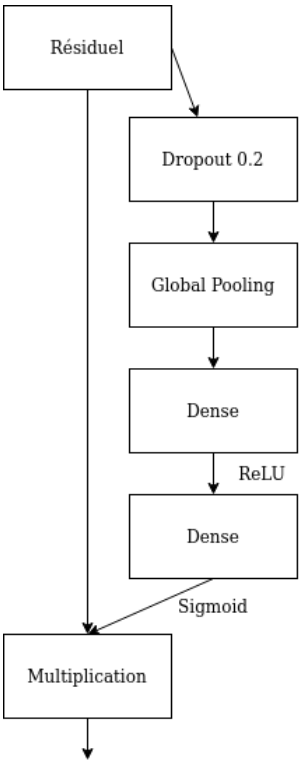

Figure 3. Bloc SE

de déterminer les channels les plus importants à la sortie du bloc. De ce fait, nous avons donc jugé plus intéressant d'agrandir le réseau par la largeur que par la profondeur. Le nombre de blocs résiduels est gardé à 6 mais le nombre de channels à la sortie des blocs a été augmenté à 90.

3.2.3. Convolutional Block Attention Module

Par rapport au bloc SE, l'intérêt du CBAM est de pouvoir évaluer l'importance spatiale des channels en plus de l'importance de chaque channel. Ce module est résumé dans la figure 4 est comportes deux sous-modules : un pour l'attention par channel, un pour l'attention spatiale.

L'attention par channel illustré dans la figure 4 est semblable à au bloc SE à l'exception qu'un Max Pooling s'effectue en parallèle du Average Pooling. Les auteurs du CBAM ont mentionné une amélioration des performances grâce à l'information que rajoute un MaxPooling. Ces couches partagent par la suite les mêmes couches Dense. Les sorties sont sommées et passées en entrées d'une fonction sigmoïde.

L'attention spatiale agrège l'information à travers les channels pour une position donnée. Le Pointwise Average réalise la moyenne des channels tandis que le Pointwise Max récupère la valeur maximum pour chaque position. Les deux plans en sortie sont concaténés et passé en entré d'une convolution 7x7 produisant un plan. Les valeurs sont

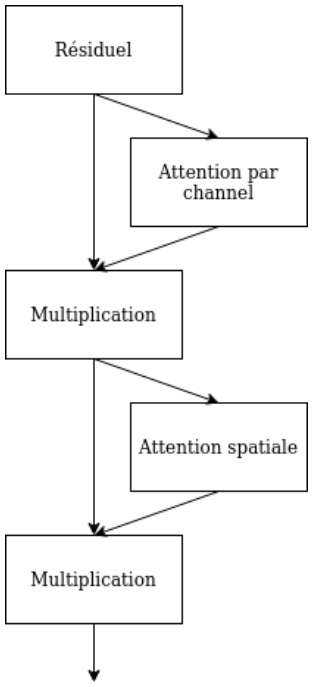

Figure 4. Bloc CBAM. À gauche, le module d'attention par channel. À droite, le module d'attention spatiale.

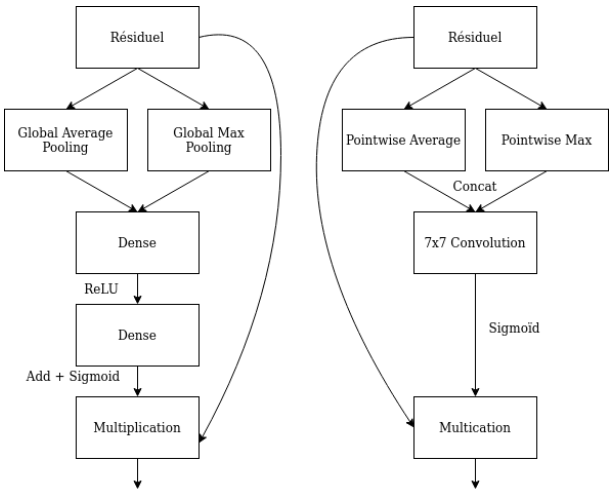

Figure 5. À gauche, le module d'attention par channel. À droite, le module d'attention spatiale.

compressées entre 0 et 1 grâce à la fonction sigmoïde puis multipliées avec les plans d'entrée, channel par channel.

3.3. Blocs de sortie

3.3.1. Sortie pour la précision de la politique

Afin de prédire le coup à jouer, le bloc de sortie pour la politique est illustré dans la figure 6. Il est composé d'une

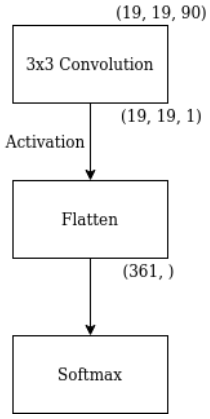

Figure 6. Bloc de sortie pour prédire le prochain coup à jouer.

couche de convolution 3x3 à 1 filtre suivi d'une fonction d'activation. Le couche Softmax permet de renvoyer une distribution de probabilité sur les coups possibles.

3.3.2. Sortie pour la valeur

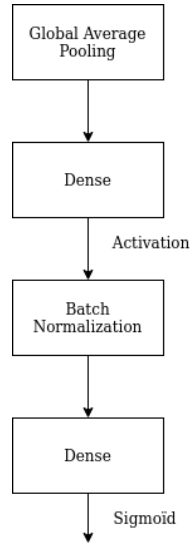

Figure 7. Bloc de sortie pour prédire le gagnant de la partie.

Le bloc de sortie pour prédire le gagnant est résumé dans la figure 7. Elle se compose de deux couches Dense consécutives. Une sigmoïde est utilisée pour déterminer la valeur. Nous avons remarqué qu'introduire une couche BatchNormalization aidait à stabiliser l'apprentissage de la value qui à tendance à fortement fluctuer durant l'entraînement.

3.4. Les hyperparamètres

Concernant les hyperparamètres: le nombre de filtres est de 90, et la taille des batchs est de 256. La policy loss utilisée est la categorical cross entropy et la value loss est la binary cross entropy. Après avoir testé plusieurs optimizers, celui que nous avons sélectionné est Adam. L'apprentissage s'est déroulé sur 600 à 700 epochs. Différents taux d'apprentissage ont été utilisés : 0.001 pour les 370 premiers epochs, 0.0003 pour les 20 suivants (de 370 à 390 epochs) et 0.0001 pour les derniers epochs. En effet, nous avons pu voir qu'en mettant un pas plus petit au fur et à mesure des epochs, nous avions de meilleures performances.

Modèle	Policy	Value	Loss
ResNet	0.4429	0.2027	3.2601
SE-Dropout-ResNet	0.4934	0.1811	2.2642
CBAM-ResNet	0.4940	**0.1807**	**2.2614**
DWC-CBAM-ResNet	**0.4948**	0.1856	2.3800

Table 1. Accuracy sur la politique et MSE sur la value en validation sur 100 000 instances de jeu. La loss est la somme entre la categorical et binary crossentropy. Ces valeurs représentent la performance des modèles en fin d'entraînement lorsque la loss stagne sur plusieurs epochs. ReLU est utilisée pour tout modèle.

4. Performances

Afin de comparer nos modèles, nous avons réalisé un réseau résiduel sans module supplémentaire. Il est uniquement composé de blocs résiduels et de BatchNormalization. La fonction d'activation utilisée au sein des blocs est la ReLU.

Le tableau 1 récapitule les résultats obtenus sur 4 modèles. La fonction d'activation utilisée pour ces modèles est la ReLU.

- ResNet : le réseau résiduel de référence,
- SE-Dropout-ResNet : le modèle résiduel avec bloc SE et Dropout au sein de ce bloc,
- CBAM-ResNet : modèle résiduel avec bloc CBAM,
- DWC-CBAM-ResNet : modèle constitué de Inverted-Residual-Block [3] avec des couches DepthwiseConv de 190 filtres d'élargissement et 90 filtres en rétrécissement.

Le tableau 1 montre que les blocs SE et CBAM permettent une amélioration des performances avec un gain de plus de 0.05 pour la politique et 0.02 pour la valeur. La loss a chuté d'un point. Le modèle utilisant des DepthwiseConv donne des résultats légèrement plus médiocre qu'avec des couches convolutives.

Le tableau 2 recense les performances des modèles lorsque la fonction hswish est utilisée comme fonction d'activation à la place de la ReLU. Cela a permis d'améliorer les performances de la politique et de la value.

MODÈLE	POLICY	VALUE	LOSS
H-CBAM-RESNET	0.5005	0.1790	2.2198
H-SE-DROPOUT-RESNET	**0.5015**	**0.1786**	**2.2185**

Table 2. Accuracy sur la politique et MSE sur la value en validation sur 100 000 instances de jeu. La loss est la somme entre la categorical et binary crossentropy. Ces valeurs représentent la performance des modèles en fin d'entraînement lorsque la loss stagne sur plusieurs epochs. Hswish est utilisée pour ces modèles.

De plus, dans la figure 8, nous pouvons voir l'évolution de l'accuracy pour la politique en validation, ainsi que l'évolution de l'erreur quadratique moyenne en validation, en fonction du nombre d'epochs.

Conclusion

Finalement, nous avons décidé de retenir le modèle H-SE-Dropout-ResNet. Ce modèle résiduel avec des blocs Squeeze-and-Excitation et Dropout etw la fonction d'activation hswish, donne d'assez bonnes performances: 0.501 en policy, 0.178 en value et 2.218 en loss.

References

[1] Andrew Howard - Mark Sandler - Grace Chu - Liang-Chieh Chen - Bo Chen - Mingxing Tan - Weijun Wang - Yukun Zhu - Ruoming Pang - Vijay Vasudevan - Quoc V. Le - Hartwig Adam. "Searching for MobileNetV3". In: *arxiv* (2019). eprint: 1905.02244.

[2] Diederik P. Kingma - Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arxiv* (2014). eprint: 1412.6980.

[3] Tristan Cazenave. "Mobile Networks for Computer Go". In: *arxiv* (2020). eprint: 2008.10080.

[4] Tristan Cazenave. "Residual Networks for Computer Go". In: *IEEE Transactions on Games, Institute of-Electrical and Electronics Engineers* 10 (2018).

[5] Liyuan Liu - Haoming Jiang - Pengcheng He - Weizhu Chen - Xiaodong Liu - Jianfeng Gao - Jiawei Han. "On the Variance of the Adaptive Learning Rate and Beyond". In: *arxiv* (2020). eprint: 1908.03265v3.

[6] D. Hendrycks et K. Gimpel. "GAUSSIAN ERROR LINEAR UNITS(GELUS)". In: *arxiv* (2016). eprint: 1606.08415.

[7] Shaoqing Ren Kaiming He Xiangyu Zhang and Jian Sun. "Deep Residual Learning for Image Recognition". In: *arxiv* (2015). eprint: 1512.03385.

[8] Sanghyun Woo - Jongchan Park - Joon-Young Lee - In So Kweon. "CBAM: Convolutional Block Attention Module". In: *arxiv* (2018). eprint: 1807.06521.

[9] Mark Sandler Andrew Howard Menglong Zhu Andrey Zhmoginov et Liang Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *arxiv* (2018). eprint: 1801.04381.

[10] Nitish Srivastava - Geoffrey Hinton - Alex Krizhevsky - Ilya Sutskever - Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks fromOverfitting". In: *Journal of Machine Learning Research* 15 (2014).

[11] Alec Radford - Jeffrey Wu - Rewon Child - David Luan - Dario Amodei - Ilya Sutskeve. "Language Models are Unsupervised Multitask Learners". In: *openai* (2018).

[12] Sergey Ioffe - Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arxiv* (2015). eprint: 1502.03167.

[13] Jacob Devlin - Ming-Wei Chang - Kenton Lee - Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arxiv* (2018). eprint: 1810.04805.

[14] Sik-Ho Tsang. "Review: WRNs — Wide Residual Networks". In: *Towards data science* (2018).

[15] Jie Hu - Li Shen - Samuel Albanie - Gang Sun - Enhua Wu. "Squeeze-and-Excitation Networks". In: *arxiv* (2019). eprint: 1709.01507.

[16] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *arxiv* (2012). eprint: 1212.5701.

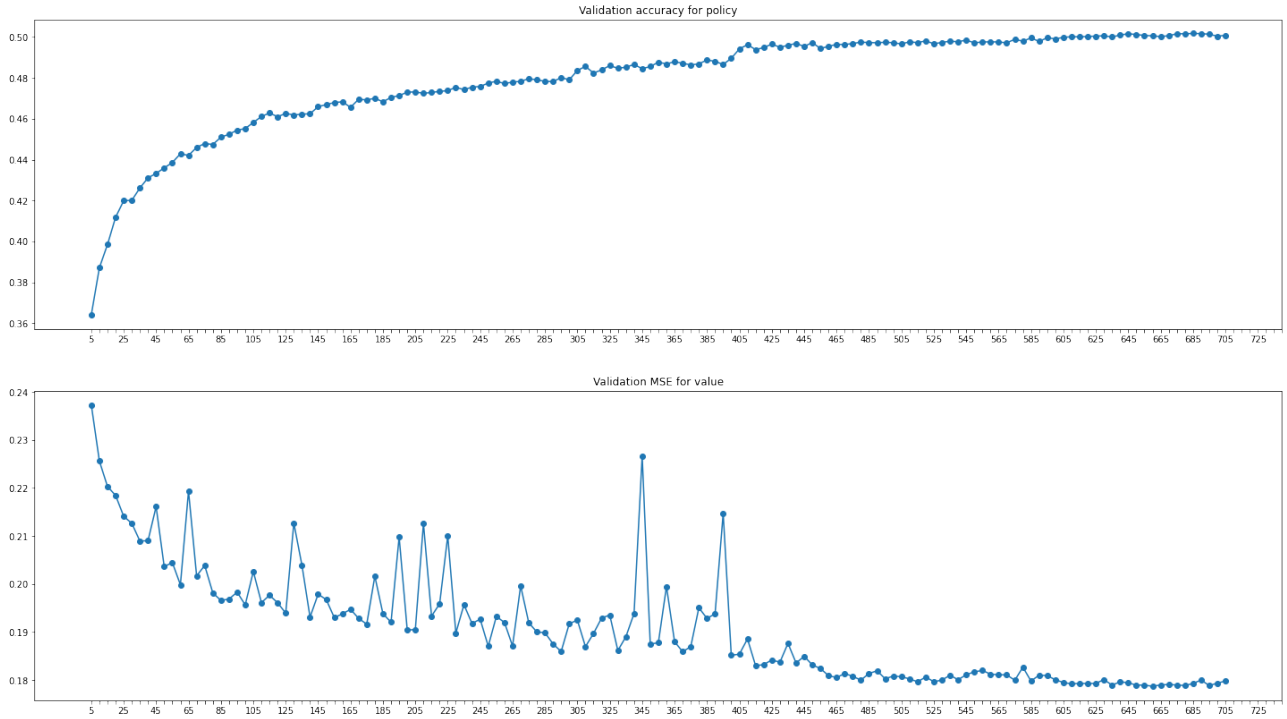

Figure 8. Evolution de l'accuracy et de la value en validation.