Réduction de modèles neuronaux profonds par adaptation de l'architecture

Paul VANDAME[1], Christophe KARAM[2], Loic ARGENTIER[2], Jocelyn CHANUSSOT[1]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-Lab, 38000 Grenoble, France

[2] Data Science Experts, 19 rue des Bergers, Grenoble, France

[paul.vandame, jocelyn.chanussot]@grenoble-inp.fr,
[Christophe, Loic]@dse-datascienceexperts.com

Résumé – À mesure que les applications en apprentissage profond se développent dans l'univers de l'embarqué, la nécessité de réduire la taille des modèles devient une problématique de premier plan. Récemment, des techniques de *pruning* et de *channel pruning* ont été employées pour augmenter la parcimonie ou diminuer la taille des modèles, respectivement, et des méthodes de distillation ont été employées pour compresser les réseaux. Cet article présente une nouvelle approche pour la simplification de l'architecture des réseaux convolutifs sans descente de gradient. La méthode est appellée ANNA pour *Architecture Neural Network Adaptation*. Cette simplification permet de créer des réseaux potentiellement composés de différentes fonctions d'activation et d'un nombre inférieur de couches et de neurones. Elle introduit également du *pruning* de connexions. Elle peut également créer de nouvelles connexions entre des neurones situés à plusieurs couches de distance, lorsque cela est bénéfique, modifiant ainsi la topologie globale du réseau. Nous avons appliqué la méthode sur des réseaux de petite taille avec quelques millions de paramètres entrainés sur la base MNIST[1]. Aujourd'hui, nous montrons être en mesure de transformer une architecture comme MobileNet entraînée avec ImageNet et comportant 4,23 millions de paramètres ([2]). Cette architecture une fois simplifiée ne contient plus que 3,28 millions de paramètres, avec seulement 1,1 million de paramètres non nuls, combinant ainsi les avantages de différentes méthodes de *pruning*.

Abstract – As deep learning applications move from the cloud to the edge, the need to reduce the model's computational load is ever-growing. Recently, pruning and channel pruning techniques have been employed to increase the sparsity or decrease the size of the model, respectively, and knowledge distillation methods have been employed to compress networks. This paper introduces a novel method for simplifying neural network architecture without any gradient descent named ANNA for Architecture Neural Network Adaptation. This simpliciation can create networks potentially composed of different activation functions, fewer layers and neurons, and introduce connection pruning. It also gives the ability to create new connections between neurons several layers away, modifying the global topology. As a proof of concept, we focused on small networks developed on MNIST[1] dataset. Moreover, we are able to morph an ImageNet-pretrained MobileNet architecture with 4.23M parameters [2], into one with 3.28M parameters, of which only 1.1M are non-zero parameters, thereby combining the advantages of different pruning methods.

1 Introduction

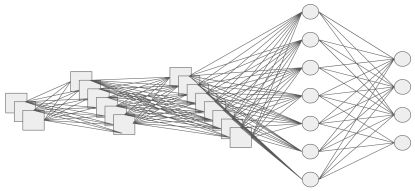
Comment choisir une architecture de réseau optimale ? Pourquoi choisir certains hyperparamètres plutôt que d'autres ? Les réseaux de neurones servant à approcher des fonctions dont les formes explicites sont inconnues, le choix d'une architecture qui serait parfaitement adaptée reste un verrou scientifique majeur. En conséquence, les modèles utilisés sont souvent génériques, capables de résoudre un large éventail de problèmes. Il en va ainsi des réseaux neuronaux convolutifs (CNN) qui peuvent être utilisés dans des problèmes de détection d'objets ou de classification d'image en tout genre. Des réseaux comme Mobilenet [2] ou Resnet [3] sont ré-entraînés et spécialisés facilement sans changer leurs hyperparamètres. De nombreuses architectures génériques ont été proposées dans la littérature et se sont révélées extrêmement puissantes pour traiter une grande variété d'applications. Toutefois, cette robustesse et cette polyvalence ont un prix : ces réseaux sont généralement surdimensionnés.

Une fois entraîné, même un modèle générique offre une solution particulière à la tâche à accomplir. Après l'apprentissage, nous pouvons donc être tentés d'ajuster l'architecture et les poids qui en résultent afin de réduire la charge de calcul. En particulier, cela peut réduire considérablement le temps d'inférence et permettre une mise en œuvre efficace sur des processeurs aux ressources limitées.

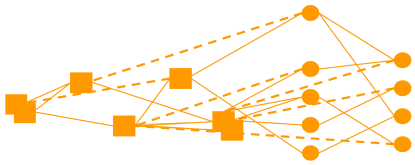
À cette fin, le pruning [4] vise à créer des matrices de poids parcimonieuses. Certaines de ces méthodes sont de type *post-training*, agissant sur un réseau déjà formé, comme la méthode RigL [5], tandis que d'autres sont des techniques *sparse training*, comme AC/DC [4]. AC/DC supprime les paramètres non essentiels au cours de l'apprentissage grâce à une méthode itérative de seuillage dur. Il a la particularité d'utiliser la version dense du réseau tout au long de la formation afin de créer progressivement une version parcimonieuse aussi précise que possible. Ces algorithmes nécessitent classiquement une adaptation des poids par une descente de gradient classique.

Dans cet article, nous proposons une nouvelle approche, ANNA (pour *Architecture Neural Network Adaptation*), capable

de combiner les avantages des approches de *pruning* [4], *channel pruning* [6] et distillation [7] sans faire appel à l'algorithme de la descente de gradient. ANNA est un analyseur de réseau qui prend un modèle entraîné et crée une architecture équivalente en sélectionnant uniquement les connexions utiles. Il peut également développer de nouvelles connexions lorsque cela est bénéfique, comme le montre la figure 1. Le résultat produit par ANNA est une nouvelle architecture avec des fonctions d'activation potentiellement différentes. L'erreur résiduelle acceptable par rapport au réseau initial est choisie par l'utilisateur. L'algorithme s'y adapte et produit une architecture équivalente du modèle de départ, mais avec une architecture moins gourmande en ressources.



(a) Réseau de neurones dense



(b) Architecture équivalente

FIGURE 1 – Le réseau dense original (**a**) est transmis à ANNA. Les cercles représentent les neurones et les carrés les canaux convolutifs. La sortie d'ANNA est présentée dans (**b**). Seule une fraction des neurones initiaux subsiste. Certaines connexions (lignes pleines) ont été jugées importantes et sont conservées. De nouvelles connexions canalisant des informations importantes (lignes en pointillée) entre des couches distinctes ont été ajoutées.

Dans cet article, nous commençons par définir quelques termes, puis nous décrivons les étapes principales de la méthode avant de présenter et discuter les résultats.

2 Définition

Dans cette section, on se propose de définir les termes et notations qui seront utilisés dans les sections suivantes.

- ***Feature map*** : chaque canal de sortie d'une couche convolutive. Le nombre de *feature maps* en sortie d'une couche convolutive est toujours égal à son nombre de canaux de sortie.
- **Sortie de couche** : Une sortie de couche est le tenseur de sortie d'une couche du réseau avant l'application d'une fonction d'activation, qu'il s'agisse d'une couche convolutive ou d'une couche dense. Lors du traitement par ANNA, ce concept inclut également la "couche" d'entrée.
- **Nœud** : la sortie d'un neurone dans une couche dense, ou une seule *feature map* dans une sortie de couche convolutive. Un nœud faisant référence à un canal ou à un neurone spécifique à l'intérieur de la sortie d'une couche. Une **sortie de couche** est donc un vecteur de **nœuds**.
- **Fonction de modification de la forme** : une fonction qui modifiera la forme de la sortie après un nœud en utilisant une ou plusieurs fonctions telles que *max-pooling, flatten*, etc.
- **Lien** : un lien est constitué d'une fonction d'activation suivie d'une fonction de modification de forme.
- **Architecture équivalente** : deux architectures différentes sont considérées comme équivalentes si elles conduisent à des résultats comparables. À partir d'une architecture initiale, ANNA est capable de construire une architecture équivalente (plus légère). La différence maximale acceptable en termes d'erreur quadratique moyenne (EQM) entre les résultats des deux architectures est fixée par l'utilisateur.

3 Description de la méthode

L'algorithme ANNA est divisé en trois étapes séquentielles :

- Un calcul de similarité pour tous les liens potentiels visant à mesurer l'importance de chaque lien et chaque nœud dans le réseau par rapport à un entraînement donné.
- Un algorithme de *matching pursuit* visant à simplifier de manière optimale l'architecture initiale.
- Une fusion des noeuds en couches visant à reconstruire une architecture équivalente avec une topologie adaptée.

4 Calcul de similarité pour tous les liens potentiels

L'objectif de cette étape est de calculer la similarité entre toutes les combinaisons possibles de nœuds. Nous obtenons d'abord toutes les sorties des couches, c'est-à-dire tous les nœuds du réseau. Chaque nœud provenant d'une couche dense sera défini comme un vecteur $\overrightarrow{N_{out}}$, chaque scalaire de ce vecteur correspondant à un résultat pour un échantillon de l'ensemble de données. Ces vecteurs ont donc la même dimension que le nombre d'échantillons d'apprentissage. Par exemple, dans l'ensemble de données MNIST[1] chaque vecteur est de dimensions 50 000. Chacun de ces vecteurs sera normalisé (méthode *z-score*). Dans le cas où le noeud provient d'un canal d'une couche convolutive, chaque pixel de la *feature map* peut être assimilé à un échantillon et on retrouve donc bien un vecteur.

Nous voulons être en mesure de reconstruire chacun de ces vecteurs, donc pour chaque $\overrightarrow{N_{out}}$ nous allons :

1. Trouver tous les candidats nœuds $\overrightarrow{N_{in}}$ qui peuvent être utilisés par $\overrightarrow{N_{out}}$ sans rompre la causalité, et qui se trouvent à une distance inférieure à $d_{connection}$ de $\overrightarrow{N_{out}}$, afin de limiter l'espace de recherche.

2. Pour chaque $\overrightarrow{N_{out}}$ identifié, trouver toutes les combinaisons possibles de fonctions d'activation et de modification de forme qui constitueront les liens reliant chaque $\overrightarrow{N_{in}}$ à $\overrightarrow{N_{out}}$.

3. Nous disposons maintenant pour chaque $\overrightarrow{N_{out}}$ d'une liste de $\overrightarrow{N_{in}}$ et de toutes les fonctions de lien qui les relient à $\overrightarrow{N_{out}}$. Cette liste est appellée **liste des liens potentiels**. Nous pouvons alors calculer toutes les similarités nécessaires comme indiqué dans l'équation 1.

$$\left\langle \overrightarrow{N_{out}},\, link_i(\overrightarrow{N_{in}}) \right\rangle \tag{1}$$

4. Enfin, nous calculons toutes les similarités entre les différents $\overrightarrow{N_{in}}$ comme montré dans 2.

$$\left\langle link_i1(\overrightarrow{N_{in1}}),\, link_i2(\overrightarrow{N_{in2}}) \right\rangle \tag{2}$$

Un procédé équivalent est utilisé si le noeud $\overrightarrow{N_{out}}$ provient d'une couche convolutive. Cette étape nécessite de faire passer les données d'entraînement à travers le réseau, de manière exhaustive ou non, mais de manière unique. Les données passeront au travers de cette étape mais ne seront jamais intégralement stockées. Les produits scalaires sont estimés en calculant lot par lot la contribution de chaque échantillon.

4.1 Simplification optimale de l'architecture initiale (algorithme de *matching pursuit*)

L'objectif de cette étape est de reconstruire un graphe de calcul en ne conservant que les connexions les plus utiles entre les nœuds. Ce processus est itératif, nœud par nœud. Pour chaque nœud $\overrightarrow{N_{out}}$ à reconstruire, nous disposons d'un dictionnaire de tous les **liens potentiels** avec tous les noeuds $\overrightarrow{N_{in}}$ possibles.

Pour sélectionner les connexions utiles et leurs poids appropriés, ANNA utilise un algorithme d'*orthogonal matching pursuit* (OMP) [8]. L'utilisation de cet algorithme est basée sur l'*extreme learning machine* (ELM) [9]. Pour fonctionner, le *matching pursuit* n'a besoin que des produits scalaires déjà calculés dans la première étape. Les données d'entraînement ne sont pas utilisées pour cette étape.

Les nœuds d'une couche convolutive sont reconstruits de la même manière, puisque la convolution est également une opération linéaire [1].

ANNA opère couche par couche, en commençant par reconstruire les sorties finales du modèle. Elle sélectionne les nœuds qui lui sont utiles par un OMP. Ensuite, ces nœuds utiles sont eux-mêmes reconstruits selon la même procédure. Ce processus se poursuit de manière itérative depuis les sorties jusqu'aux entrées. Le graphe résultant est présenté en Figure 1b.

1. Les notions de *stride*, de *padding* et de taille de masque convolutif sont également pris en compte dans le calcul

4.2 Du graphe de calcul à un réseau structuré

À ce stade, tous les nœuds utiles composant le graphe de calcul ont été reconstruits, mais le réseau a perdu sa structure antérieure : de nouvelles connexions ont été établies et d'autres ont été supprimées.Il s'agit simplement d'un graphe d'opérations. Il est alors nécessaire de regrouper ces nœuds dans une structure en couches pour accélérer le calcul. Les nœuds sont regroupés en couches par le biais d'un processus de regroupement hiérarchique qui vise à minimiser la parcimonie. Au moment de fusionner les noeuds entre eux, le but est de minimiser la parcimonie afin d'augmenter au maximum le *channel pruning*. Un dendrogramme est construit en prenant comme entrée tous les noeuds. On regarde le nombre de zéros que ferait apparaître chaque paire de noeuds si elle formait une couche. La parcimonie ainsi créée forme la mesure de ce dendrogramme. La parcimonie est alors une conséquence de la fusion du graphe de calcul en un réseau structuré en couches. Le réseau résultant forme une architecture équivalente, utilisant des couches convolutives classiques simples, des couches denses et des opérations supplémentaires, similaires aux composants d'un modèle ResNet [3]. Ce modèle est également produit sous forme de code à l'aide des fonctions et classes classiques de Torch [10] et a donc un format compatible ONNX (*Open Neural Network Exchange*).

5 Résultats

Nous présentons des résultats obtenus sur deux configurations. Le premier a été réalisé pour simplifier un réseau convolutif séquentiel de 6 couches entraîné sur MNIST [1]. Ce réseau avant simplification était composée de $904 \cdot 10^3$ paramètres et donnait une précision de 99.76%. En réglant différemment les différentes options de reconstruction nous pouvons obtenir différents réseaux équivalents. L'un d'entre eux est présenté en Figure 2. La simplification optimale fournit un modèle avec un nombre de paramètres égal à $139 \cdot 10^3$ dont seulement $27 \cdot 10^3$ sont non nuls. La précision de ce réseau convolutif simplifié équivalent est de 99.22%.

Cette méthode a également été appliquée sur des réseaux tels que Mobilenet [2] entrainés sur ImageNet. Les résultats sont présentés dans le tableau 1.

TABLE 1 – Résultats sur MobileNetV1

Pruning Method	Acc. (%)	# Params	# Non-Zero Params
Dense [2]	73.4	4.23M	4.23M
AC/DC-1 [4]	70.2	4.23M	1.07M
AC/DC-2 [4]	66.1	4.23M	0.44M
MetaPruning-1 [11]	70.9	3.17M	3.17M
MetaPruning-2 [11]	66.1	2.12M	2.12M
Uniform Pruning [11]	68.4	3.17M	3.17M
ANNA-1	**73.3**	**4.23M**	**3.33M**
ANNA-2	**71.3**	**4.18M**	**2.79M**
AC/DC-1+ANNA-3	**69.9**	**3.28M**	**1.15M**
AC/DC-1+ANNA-4	**65.9**	**2.62M**	**0.94M**

Pour MobileNet, ANNA a été calculé avec seulement 10% de

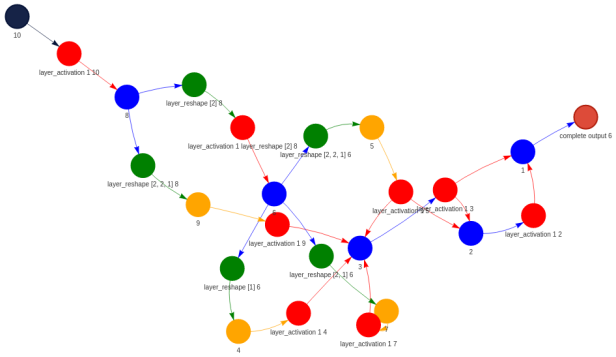


FIGURE 2 – Architecture équivalente résultant d'une simplification par ANNA. Les points noirs représentent les entrées du réseau, les points rouges indiquent les fonctions d'activation, et les points bleus sont des couches convolutives ou linéaires. Les points verts représentent des fonctions de remodelage telles que *Max-Pooling* ou *Flattening* ou un mélange de plusieurs opérations de ce type. Les points jaunes sont des couches de sélection, capables de prendre un vecteur en entrée et de n'extraire que les composantes utiles. Ils sont systématiquement placés entre une fonction *flatten* et une couche dense. Ce réseau simplifié à une précision de 99.37 % pour $22 \cdot 10^3$ paramètres non nuls et $966 \cdot 10^3$ paramètres au total.

l'ensemble des données Imagenet [12]. ANNA n'a pas besoin de traiter l'ensemble des données pour obtenir un bon résultat au sens de la précision. Par rapport à l'algorithme AC/DC[4], méthode de référence en *pruning*, ANNA a pu réduire encore le nombre de paramètres de 22 % (de 4,23 M à 3,28 M), au prix d'une réduction minime de la précision de 0,3 point et d'une parcimonie légèrement réduite. MetaPruning [11] est une méthode de référence en *channel pruning*, qui produit un réseau simplifié (le nombre de paramètres est fortement réduit) mais dont le résultat reste dense (parmi les paramètres restants, peu sont nuls). ANNA permet d'obtenir un nombre similaire de paramètres que MetaPruning, mais avec l'avantage supplémentaire d'avoir environ 65% de parcimonie en plus. ANNA permet de choisir le taux d'erreur que l'on veut et fait une reconstruction en fonction. Le taux d'erreur sur les neurones de sortie de la dernière couche est choisi. L'erreur tolérable sur les nœuds précédents pour chaque reconstruction est estimée en fonction des poids. le nombre de liens choisis pour chaque reconstruction est différent pour chaque noeud et dépend de ce seuil. Différentes reconstructions peuvent donc être intéressantes en fonction de la précision du modèle requis.

6 Conclusion

ANNA s'est avéré capable d'ajuster l'architecture d'un réseau, d'introduire le concept de réseaux équivalents et de créer de nouvelles connexions entre les couches indirectement connectées. Il est également capable de simplifier les réseaux en supprimant leur redondance linéaire, et de fusionner les avantages du *pruning* et du *channel pruning* dans un compromis contrôlé. Sa particularité de ne pas utiliser la rétropropagation le rend utilisable en addition de toute autre méthode de manière assez efficace, comme l'ont montré les résultats obtenus avec AC/DC.

En n'utilsant pas de descente de gradient, ANNA est en rupture philosophique avec les méthodes classiquement utilisées. Toutefois, un point critique d'ANNA est sa consommation en ressources de calcul qui la rend aujourd'hui inutilisable pour des réseaux de taille supérieure à 25M de paramètres avec des jeux de données comme Imagenet [12].

En perspectives, il peut être intéressant d'étudier les possibilités offertes par ANNA en modifiant la définition du produit scalaire utilisé. Nous pourrions reconstruire le réseau dans un autre espace de Hilbert. Enfin, une reconstruction nœud par nœud pourrait nous permettre de déterminer quelles voies sont les plus importantes pour quelles sorties. En outre, la création de modèles équivalents nous permet de comparer les similitudes et les divergences entre différentes architectures à résultat équivalent.

Références

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[2] Andrew G. Howard, Zhu, and co. Mobilenets : Efficient convolutional neural networks for mobile vision applications. *CoRR*, 2017.

[3] Kaiming He, Xiangyu Zhang, Ren, and co. Deep residual learning for image recognition. *CVPR*, 2015.

[4] Alexandra Peste, Eugenia Iofinova, and co. AC/DC : alternating compressed/decompressed training of deep neural networks. *NIPS*, abs/2106.12379, 2021.

[5] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery : Making all tickets winners. *ICML*, 2020.

[6] Xiaohan Ding, Hao, and co. Resrep : Lossless cnn pruning via decoupling remembering and forgetting. *ICCV*, 2021.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS*, 2015.

[8] Gerlind Plonka, Sebastian Hoffmann, and Joachim Weickert. Pseudo-inverses of difference matrices and their application to sparse signal approximation. *Linear Algebra and its Applications*, 503 :26–47, 2016.

[9] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine : Theory and applications. *Neurocomputing*, 70(1) :489–501, 2006. Neural Networks.

[10] Adam Paszke, Sam Gross, and co. Pytorch : An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[11] Zechun Liu, Haoyuan Mu, Zhang, and co. Metapruning : Meta learning for automatic neural network channel pruning. *ICCV*, 2019.

[12] Olga Russakovsky, Deng, and co. Imagenet large scale visual recognition challenge, 2014.