Rapport de projet ISI-10 : AdaNet, Adaptive Neural Newtorks.

Luc Blassel luc.blassel@agroparistech.fr [1]
Romain Gautron romain.gautron@agroparistech.fr [1]

Introduction

Le but de ce projet est de reproduire les expériences et les méthodes issues du papier suivant : C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, S. Yang *AdaNet: Adaptive Structural Learning of Artificial Neural Networks*. En essayant de reproduire leur méthode qui consiste à construire des réseaux de neurones dont la structure est apprise et optimisée en même temps que les poids. Cette méthode sera appliquée sur une tache de classification binaire issue du jeu de données d'images CIFAR-10.

1. Comment marche AdaNet ?

La structure du réseau est générée. Le réseau entier qu'on appellera le réseau AdaNet, est augmenté à chaque itération par un sous réseau. Ce sous réseau est choisi en fonction de son effet sur une fonction objectif pour pouvoir sélectionner le meilleur sous réseau à rajouter au réseau Adanet à chaque étape. Pour bien comprendre le déroulement de ce processus on notera : f_t le réseau AdaNet a l'étape t, h_t et h'_t les sous réseaux candidats (a l'étape t) avec pour chacun de ces sous réseaux $h_{t,k}$ la k^{eme} couche du sous réseau et $h_{t,k,j}$ le j_{eme} neurone de cette couche. l est le nombre de couche maximale du sous réseau.

L'algorithme se déroule alors selon ces étapes :

1. **Initialisation du réseau :** On commence par générer les couches d'entrée et de sortie.
2. **Génération de sous-réseaux candidats :** On génère deux sous réseaux candidats *(cf. Figure 1)* :
   — Un qui a une profondeur identique au sous réseau généré précédemment $\rightarrow k$
   — Un qui augmente la profondeur de 1 par rapport au sous réseau précédent $\rightarrow k+1$

   Ces deux sous réseaux obéissent aux mêmes règles de connectivité. C'est à dire que le première couche du réseau du sous réseau h_t est reliée à la couche d'entrée de f_t, la dernière couche $h_{t,l}$ du sous réseau est forcement connectée à la couche de sortie de f_t. Pour les couches intermédiaires, chaque couche $h_{t,k}$ est forcement reliée à $h_{t,k-1}$ et peut être reliée aux couches de niveau précédent dans tous les sous réseaux précédemment générés, c'est à dire $h_{t\in[1,t-1],k-1}$. *Lors de la première itération il faut forcement choisir le sous réseau qui augmente la profondeur puisque* $k = 0$
3. **Choix du sous-réseau :** Parmi ces deux sous réseaux candidats $\mathbf{h}$ et $\mathbf{h}'$ on choisit celui qui, après un entraînement, minimise le plus la fonction objectif suivante :

$$F_t(\mathbf{w}, \mathbf{h}) = \frac{1}{m}\sum_{i=1}^{m} \Phi(1 - y_i f_{t-1}(x_i) - y_i \mathbf{w} \cdot \mathbf{h}(x_i)) + \mathcal{R}(\mathbf{w}, \mathbf{h})$$

   Avec x_i les exemples d'entraînement, y_i les réponses attendues, m Le nombre d'exemples,$\mathbf{w}$ Les poids associés au sous réseau candidat $\mathbf{h}$ et Φ une fonction (soit la fonction exponentielle soit la fonction logistique). Le terme $\mathcal{R}(\mathbf{w}, \mathbf{h})$ est un terme de régularisation qui est laissé à 0 lors des expérimentations.
   Si $F(\mathbf{w}, \mathbf{h}) > F(\mathbf{w}, \mathbf{h}')$ alors le sous réseau candidat $\mathbf{h}'$ est choisi et $\mathbf{h}_t \leftarrow \mathbf{h}'$

4. **Condition d'arrêt :** Une fois que le meilleur sous réseau est sélectionné on regarde si celui ci a apporté une amélioration au réseau AdaNet. Si la fonction objectif de l'entraînement de f_{t-1} *(qui n'est pas le même que celle utilisée pour le choix du meilleur sous réseau)* est meilleure avec le sous réseau que sans alors : $f_t \leftarrow f_{t-1} + \mathbf{w}^*(h)_t$ sinon l'intégration du sous réseau n'apporte rien de plus au réseau f_{t-1} et celui est retourné, terminant alors l'exécution de l'algorithme. Si cette condition d'arrêt n'est jamais vérifiée, l'algorithme s'arrête au bout de T itérations et retourne le réseau f_T

On voit donc bien que la génération et l'ajustement des poids du réseau se fait itérativement à chaque étape.

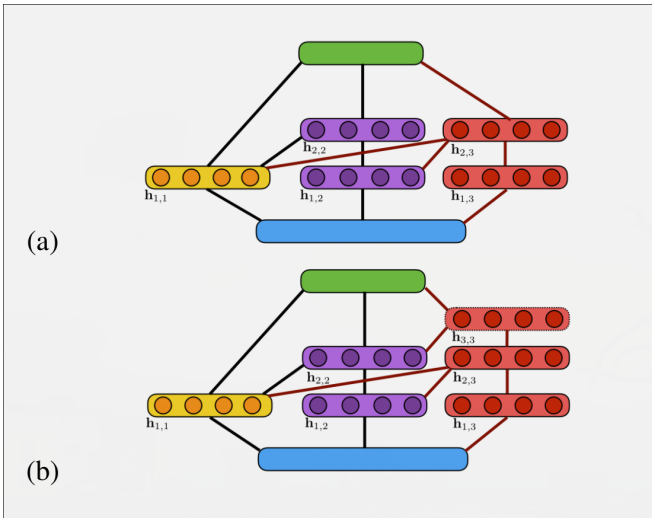

Figure 1. Les deux sous réseaux candidats générés à l'étape 3 (en rouge), un de profondeur $k = 2$ comme le sous réseau choisi à l'étape précédente (en bleu), et un de profondeur $k = 3$ qui augmente la profondeur générale de 1. Ce sont ces deux réseaux AdaNets potentiels qui vont être évalués

2. Choix lors des expérimentations

2.1. Choix du dataset

Une parmi deux expérimentations à reproduire était au choix :

— CIFAR-10 dataset
— Criteo dataset

Le choix s'est porté sur la taille du jeu de données à traiter. Si l'on ne retient que deux classes à traiter, CIFAR contient moins de 10000 éléments par classe tandis que le dataset Criteo contient 30 millions de lignes pour le set d'entraînement.

2.2. Choix du langage et de l'API

On programme sous `Python` et on utilise l'API `Keras` pour sa syntaxe simple et sa très grande modularité. Le backend utilisé est `Tensorflow` mais cela n'a aucune influence sur le code dans `Keras`.

2.3. Imprécisions du papier

2.3.1. Gestion des tenseurs multiples en entrée

On ne sait pas si lorsqu'une couche reçoit les entrées de plus d'une autre couche les tenseurs sont soit :

— juxtaposés (fonction `concatenate`)
— sommés (fonction `add`)

2.3.2. Régularisation des sous réseaux

On ne sait pas explicitement si les sous réseaux sont régularisés à l'aide d'une norme L_1 ou L_2. L'utilisation du terme λ nous fait choisir la norme L_1. On ne sait pas non plus si c'est la loss qui est régularisée ou les poids lors de l'apprentissage. On choisit de régulariser avec une norme L_1 les poids de chaque couche des sous réseaux ainsi que la couche de sortie.

3. Choix pour l'implémentation

3.1. Traitement des données en entrée

Pour traiter les images en dehors d'un réseau de convolution, on vectorise celles-ci en un vecteur de longueur $width * height * 3$

3.2. Architecture des couches d'entrée et de sortie

— La couche d'entrée est de taille $width * height * 3$ soit 3072 pour les images de CIFAR-10
— La couche de sortie contient un unique neurone et une activation sigmoïde, la classification étant binaire

3.3. Génération des sous réseaux

Nous choisissons de générer les sous réseaux aléatoirement. Chaque couche est connectée à la précédente du même sous réseau, mais les connexions avec les couches de niveau inférieur des autres sous réseaux est aléatoire.

3.4. Choix de la fonction objectif

Nous choisissons de générer les sous réseaux aléatoirement. Chaque couche est connectée à la précédente du même sous réseau, mais les connexions avec les couches de niveau inférieur des autres sous réseaux est aléatoire.

3.5. Fonction objectif

$$F_t(\mathbf{w}, \mathbf{h}) = \frac{1}{m} \sum_{i=1}^{m} \Phi(1 - y_i f_{t-1}(x_i) - y_i \mathbf{w} \cdot \mathbf{h}(x_i)) + \mathcal{R}(\mathbf{w}, \mathbf{h})$$

Avec x_i les exemples d'entraînement, y_i les réponses attendues, m le nombre d'exemples,$\mathbf{w}$ les poids associés au sous réseau candidat $\mathbf{h}$ et Φ la fonction exponentielle. Le terme $\mathcal{R}(\mathbf{w}, \mathbf{h})$ est un terme de régularisation qui est laissé à 0 lors des expérimentations.

On interprète cette formule avec les conditions ci-dessus comme :

$$F_t(\mathbf{h}) = \frac{1}{m} \sum_{i=1}^{m} \exp(1 - y_i \cdot pred_{t-1}(x_i) - y_i \cdot pred_{t-1+\mathbf{h}}(x_i))$$

4. Implémentation pratique

Nous avons tenté d'être le plus fidèle possible à l'implémentation du papier. Nous avons du prendre quelques décisions cependant lorsque le papier n'était pas explicite sur la méthode utilisée.

4.1. Génération et sélection des sous réseaux

Nous fournissons un paramètre à l'algorithme qui permet de spécifier combien de sous modèles candidats sont générés à chaque itération avec un paramètre *reps*. Étant donne que dans le papier il n'y a que deux types de sous-réseaux générés (même profondeur ou profondeur +1) nous avons fait le choix de générer des sous réseaux de même profondeur que le réseau du pas de temps précédent pour les $\frac{reps}{2}$ premières itérations et des sous réseaux plus profonds pour les itérations suivantes.

Les connexions entre les différentes couches suivent les règles de connectivité décrites plus haut, c'est à dire dernière couche du sous réseau reliée à la couche de sortie, première couche du sous réseau reliée à la couche d'entrée et toutes les couches internes sont reliées entre-elles au sein d'un même sous réseau. La variabilité entre les sous réseaux candidats apparaît dans la connectivité entre les sous réseaux candidats. Pour chaque couche du sous-réseau candidat considéré, un nombre aléatoire de connexions aux sous couches de profondeur inférieure de 1 parmi les sous réseaux générés aux pas de temps précédents, est sélectionné.

Le nombre de neurones par sous couche des sous-réseaux candidats est déterminé par un paramètre fixé au début de l'algorithme. Donc toute la variabilité entre les candidats vient du nombre de connexions aux sous-réseaux précédents et au changement de profondeur.

Tous les sous-réseaux candidats sont insérés dans le réseau adanet de l'étape précédente pour être évalués, après entraînement, sur un set de test, selon la fonction objective définie en 3.5. Le

résultat de la fonction objective le plus bas est mis en mémoire ainsi que le modèle adanet correspondant, et mis à jour si besoin à chaque évaluation de candidat. A la fin de l'évaluation des candidats le meilleur réseau est sauvegardé sur le disque et utilise comme modèle de base pour l'étape suivante.

4.2. Implémentation de la gestion d'entrées multiples

Comme dit dans la partie 2.3.1, il n'est pas dit dans le papier, comment la gestion du nombre de neurones entrant variable pour certaines couches. Si on concatène les tenseur des couches entrantes on change la taille d'entrée de la couche supérieure. Ceci pose un problème lors du chargement des poids du modèle adanet de l'étape précédente puisque le tenseur de poids sauvegardes et le nouveau tenseur ne font plus la même taille. Une solution que nous avons utilisée est de ne pas charger les poids précédents pour ces couches de concaténation. cependant cela peut faire perdre en performance. Nous avons donc préféré sommer les tenseurs entrants et garder les poids de l'itération précédente.

4.3. Problème de flexibilité

Puisque la structure change constamment au fur et à mesure des itérations on ne peut pas se contenter de sauvegarder les modèles directement au disque puisque l'on ne peut pas changer les modèles une fois qu'ils sont compilés. Pour pouvoir passer outre ce problème nous avons sauvegardé la structure exacte du modèle dans un dictionnaire et on reconstruit le modèle à partir de ce dictionnaire à chaque itération. Une fois construit on charge les poids de l'ancien modèle dans les couches qu'il a en commun avec le nouveau à chaque itération le dictionnaire correspondant au meilleur modèle est simplement sérialisable puisque ça n'est que du texte.

4.4. Condition d'arrêt

Dans le papier il est dit que l'algorithme s'arrête lorsque l'ajout d'un nouveau sous-réseau ne fait pas diminuer sa fonction objectif. Au regard de nos puissances limitées de calcul nous avons fait le choix de s'arrêter lorsque la diminution de la fonction objective passe sous un certain seuil. L'entraînement des différents sous-réseaux est aussi soumis à une condition similaire d'arrêt lorsque l'erreur de classification sur un jeu de validation ne diminue plus.

5. Résultats

Nous n'avons pas pu refaire toutes les optimisations d'hyper-paramètres qui ont été faites dans le papier, donc nous avons simplement choisi un jeu d'hyper-paramètres pour nos expérimentations :

- $n = 150$, le nombre de neurones dans chaque couche
- $\lambda = 0.000001$, le paramètre de normalisation
- $\eta = .001$, le taux d'apprentissage
- $T = 10$, le nombre de répétitions
- $reps = 5$ le nombre de sous-réseaux générés a chaque étape

Nous avons ensuite exécuté notre algorithme sur différents jeux de données binaires (les mêmes que ceux du papier), mesuré leur précision de classification sur un jeu de test et sur 10 exécutions pour avoir une performance moyenne et un écart-type. Il est à noter que le set CIFAR-10 est déjà séparé en un set de test et d'entraînement. On obtient alors les résultats de la table 1

Les résultats sont très variables en fonction des couples. Cela s'explique par le fait qu'aucune optimisation des hyper-paramètres n'a été faite. Ainsi, à la faveur des choix arbitraires que l'on a fait on peut se retrouver tant avec de très bon résultats (deer-horse) qu'avec de mauvais résultats.

6. Conclusion

Nous avons essayé dans ce projet de reproduire les résultats d'Adanet. Nous avons été contraints de faire des choix, notamment sur la génération des sous réseaux et nous avons du nous priver d'optimisation des hyper-paramètre (nécessité d'implémenter en plus une méthode sophistiquée). De ce fait, les résultats que nous avons obtenus sont, exception faite pour deer-horse, nettement

jeu de données	résultat de notre version	résultat papier
`deer-truck`	$\mathbf{0.2741 \pm 0.0391}$	0.9372 ± 0.0082
`deer-horse`	$\mathbf{0.8731 \pm 0.0097}$	0.8430 ± 0.0076
`automobile-truck`	$\mathbf{0.2521 \pm 0.0204}$	0.8461 ± 0.0069
`cat-dog`	$\mathbf{0.3085 \pm 0.0449}$	0.6924 ± 0.0129
`dog-horse`	$\mathbf{0.5711 \pm 0.0172}$	0.8350 ± 0.0089

Table 1. Comparaison des précisions de classification entre la version du papier et la notre

moins bons que ceux obtenus dans le papier. Certains des résultats que nous avons obtenus (deer-truck, automobile-truck et cat-dog) semblent biaisés puisque l'on obtient des précisions inférieures a 50%. Dans tous les cas, des performances nettement meilleures que celles obtenues dans le papier peuvent être obtenues via du transfert learning (précision supérieure à 90%). Ceci ne montre pas clairement l'intérêt de cette méthode très nécessiteuse de calculs (génération d'un grand nombre de sous réseaux). Cependant, si l'on ne dispose pas d'un modèle entraîné pour faire de l'apprentissage par transfert, alors cette approche peut être intéressante.