PARCOURS DATASCIENTIST

PROJET 8

EVALUATION DE L'ALGORITHME ADANET
Application aux réseaux DNN, CNN et RNN.

mai 23, 2019

CONTENTS

Table of Contents

1 Introduction

Le choix de l'architecture d'un réseau de neurones est un problème non trivial, la complexité d'un réseau de neurone allant de pair avec la complexité du problème sous-jacent à résoudre.

Ce type d'architecture nécessite de choisir des hyper-paramètres tels que le nombre de couches du réseau, la nature de ces couches (convolution, dense…), le nombre d'unités (lesdits neurones) par couche. La recherche de ces hyper-paramètres peut s'avérer coûteuse en ressources et en temps.

Cette étude présente une évaluation de l'algorithme ADANET à l'aide d'images issues de la base de données Standford dogs dataset. C'est un problème de classification multi-classes consistant à prédire une race d'un chien à partir de son image.

L'algorithme est fondé sur la théorie de l'apprentissage statistique et des modèles ensemblistes. A ce titre, ADANET se veut une extension des méthodes ensemblistes aux réseaux de neurone.

L'algorithme ADANET a pour but de pallier la recherche exhaustive des hyper-paramètres d'un réseau de neurones. Le modèle prédictif est construit en apprenant la structure d'un réseau de neurones à partir de sous-réseaux de neurones candidats. La structure du réseau résultat satisfait aux exigences de complexité inhérentes aux réseaux de neurones avec une garantie d'apprentissage.

Les propriétés de la fonction de coût du modèle, formulée à partir de la modélisation d'un réseau de neurones et du principe de minimisation du risque structurel, sont décrites. Il en va de même de la garantie d'apprentissage issue de l'étude de la complexité des familles de classifieurs dont le réseau de neurones résultat est issu.

ADANET est un framework basé sur Tensorflow. Les grandes lignes de l'algorithme ADANET sont détaillées au travers de la mise en œuvre de la fonction de coût, la mise en œuvre de la régularisation et de l'algorithme dit Weaklearner.

Les réseaux de type DNN, CNN et RNN sont testés. Les jeux de données sont présentés pour chacun des cas de tests. L'implémentation utilise le package ADANET de Tensorflow. La construction des sous-réseaux candidats fournis pour l'apprentissage d'ADANET sont implémentés en Tensrorflow.

2 Les principes de l'apprentissage statistique

2.1 Le principe de minimisation de l'erreur empirique

C'est l'un des principes fondateurs de l'apprentissage statistique. Explorons la sémantique de l'expression en décrivant l'erreur empirique.

2.1.1 L'erreur empirique

L'erreur empirique est l'erreur que le réseau de neurones commet sur un échantillon de données.

Peut-on estimer cette erreur empirique ? Oui heureusement, mais sous certaines conditions, que l'on s'attachera à toujours remplir.

L'une de ces conditions est que le phénomène que l'on cherche à prédire est décrit par une loi de probabilité.

Prenons un exemple : on cherche a classer des chiens selon leur race. L'image d'un lévrier est présentée au réseau qui a été entraîné a reconnaître les races Dog allemand, lévrier et Pékinois.

Le réseau produit le résultat suivant :

Image d'un chien présenté au réseau	Probabilités de prédiction	
Lévrier	Dog Allemand	10 %
	Lévrier	89 %
	Pékinois	1 %

Le phénomène que l'on cherche à prédire est décrit par une loi de probabilité. Dans ce cas, on montre que l'erreur empirique est représentée par **l'espérance** de l'erreur de prédiction.

Notons que la loi de distribution ainsi déterminée par les résultats de l'expérience de classement d 'un chien selon son image, n'est pas connue à priori.

2.1.2 En statistiques, pourquoi l'espérance ?

Intuitivement, l'espérance, dans le champ des lois de probabilités, est une valeur aléatoire que l'on s'attend à trouver, en moyenne, si on répète un grand nombre de fois la même expérience aléatoire.

Transposée à l'erreur empirique à estimer, l'espérance de l'erreur du réseau de neurones est la valeur que l'on s'attend à trouver si on fait un grand nombre de prédictions avec ce réseau.

L'espérance est une grandeur statistique. On peut raisonnablement s'attendre à ce que l'espérance de l'erreur empirique, calculée sur un échantillon, ne change pas pour des prédictions sur des données que le réseau n'a pas encore traitées.

Peut on généraliser l'espérance de l'erreur empirique, à des données autres que l'échantillon ?
Dans l'affirmative, on obtient alors un réseau capable de produire une erreur de généralisation approximée par l'erreur empirique. Si les conditions d'une telle disposition sont satisfaites, alors, l'algorithme est dit consistant avec la distribution des prédiction des données.

Intuitivement, on peu s'attendre à ce que cela soit le cas si les données d'entraînement tendent vers l'infini. Le réseau aura appris tous les cas de données possibles.

Autrement formulé, en notant $\hat{E}$ l'espérance empirique du réseau et E son espérance, un ensemble des données contenant M échantillons, demandons nous sous quelles hypothèses la proposition suivante :

$$\lim_{M \to \infty} \hat{E}(M) = E$$

est elle vraie ? Cette équation traduit le fait que si le nombre de données croît à l'infini, alors l'espérance de l'erreur empirique de prédiction tend à devenir égale à l'espérance de l'erreur de prédiction du réseau, soit, l'erreur que l'on s'attend à trouver en moyenne.

Pour ce faire, les données doivent satisfaire deux hypothèses:

- Tous les couples (donnée, prédiction) doivent avoir la même loi de distribution.
- Chaque nouvelle observation apporte de nouvelles informations au réseau, par là même, enrichie le réseau en information.

Ces hypothèses sont appelées I.I.D. pour expériences Indépendantes et Identiquement Distribuées.
NB : la loi de distribution évoquée ici pour les prédictions n'est généralement pas connue à priori.

2.1.3 Comment s'énonce le principe de minimisation du risque empirique ?

Pour rappel, le risque empirique est l'erreur mesurée sur les données d'entraînement de l'algorithme. C'est l'écart constaté entre une valeur attendue et une valeur mesurée par la fonction de prédiction.

Le principe de minimisation du risque empirique énonce que la fonction de prédiction est recherchée dans une classe de fonctions qui rend minimum l'erreur empirique, i.e. l'erreur sur les données d'entraînement.

En trouvant la fonction qui minimise le risque empirique, l'espérance de l'erreur sur des données d'entraînement est la plus petite.

2.2 Le principe de minimisation du risque structurel

2.2.1 L'erreur de généralisation

A partir d'un échantillon de données, une fonction ***f*** de prédiction a été construite de façon à être performante sur ce jeux de données,i.e. conforme au principe de minimisation du risque empirique.

Mais qu'en est il de la performance d'un tel estimateur sur un jeux de données qui n'a pas été utilisé dans la phase d'entraînement ?

La réponse est qu'il est probable que l'erreur mesurée sur ce jeux de données, appelé les données de test, soit supérieur à l'erreur empirique (mesurée sur les données d'entraînement).

C'est ce que traduit l'inégalité suivante :

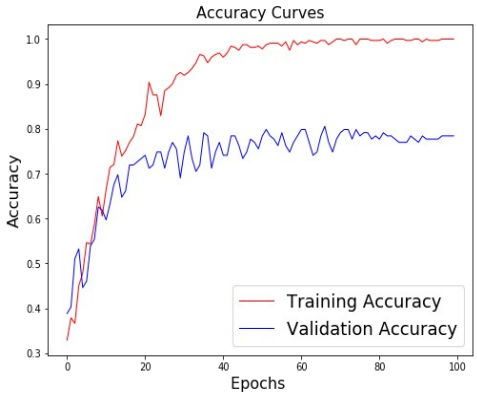
$$\widehat{\Re}(f,S) \leqslant \Re(f)$$

avec les notations:

$$\widehat{\Re}(f,S) : l'erreur\ empirique$$
$$f : l'estimateur$$
$$S : les\ données\ d'entraînement$$
$$\Re : l'erreur\ de\ généralisation$$

Cette inégalité est vraie en probabilité.

Le diagramme ci dessous, mesurant la précision d'un estimateur construit sur des données d'entraînement et évalué par rapport à des données de test, illustre l'inégalité précédente :



La courbe en bleue traduit une précision inférieure à celle de la courbe en rouge, indiquant une erreur de généralisation plus grande que l'erreur empirique.

2.2.2 La consistance du principe de minimisation du risque empirique

En se posant la question, peut on espérer construire un estimateur (une fonction de prédiction) qui satisfasse aux exigences du principe de minimisation du risque empirique et dont l'erreur empirique soit un reflet de l'erreur de généralisation ?
En répondant par l'affirmative a cette question, il sera assuré que si l'erreur empirique décroît, alors l'erreur de généralisation en fera autant.

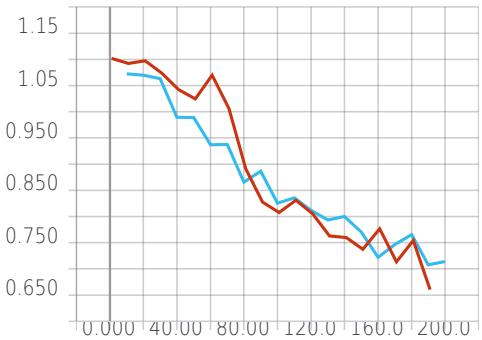
La réponse à la question est affirmative à la condition que le principe de minimisation du risque empirique soit consistant. En d'autres termes, la consistance traduit le fait qu'en plus que l'erreur empirique soit un reflet de l'erreur de généralisation, l'algorithme à l'origine de la construction de

l'estimateur doit rechercher ce dernier dans une famille d'estimateurs ***F*** qui satisfasse, en probabilité, l'inégalité suivante :

$$\forall f \in F : \Re(f) \leqslant \hat{\Re}(f, S) + C(F)$$

Le principe de minimisation du risque structurel est la traduction de cette inégalité. Les estimateurs devront être recherchés dans des familles d'estimateurs qui satisfassent à cette inégalité. L 'algorithme de recherche devra écarter les autres.

Le diagramme ci-dessous illustre la consistance du principe de minimisation de l'erreur empirique :



Les courbes des fonctions d'erreurs des données d'entraînement et de test décroissent toutes deux, avec la même allure.

2.2.3 Comment affiner la borne supérieure de l'erreur de généralisation ?

L'inégalité précédente montre que plus petite sera la valeur de ***C(F)***, plus petite sera l'espérance de l'erreur de généralisation ***R(F)***.

Optimiser la valeur de cette borne supérieure est un des enjeux des travaux de recherche en apprentissage statistique.

Ces travaux passent par l'étude des familles d'estimateurs. Plus précisément, il a été établie [2] que la mesure de la complexité d'une famille d'estimateurs permet de donner une valeur à cette grandeur ***C*** de l'inégalité précédente.

Les incidences de la complexité sur les performances des estimateurs sont l'objet du paragraphe qui suit.

3 La complexité d'une classe de fonctions

Lorsque le réseau de neurones apprend, il est en capacité de faire des prédictions Y à partir de données X.

Ce faisant, entreprendre la prédiction de la donnée X par le réseau ainsi appris, revient à appliquer à X une fonction G que l'on peut formellement définir comme suit :

- G : X → G(X)=Y

Ces fonctions G appartiennent à une classe : celle des fonctions construites à partir des données d'entraînement et qui satisfasse au principe de minimisation du risque structurel. Ce principe a pour origine la consistance du principe de minimisation du risque empirique.

3.1 Qu'est ce que la complexité ?

C'est un concept qui est inhérent au nombre d'éléments qui compose un système.

A titre d'exemple le problème consistant à classifier des chiens à partir de leur image en deux races est à priori, moins complexe que celui de les classer en dix races. Un problème de classification

d'images en couleurs est plus complexe que celui de classer des images en noir et blanc. Les plans couleurs participent à l'augmentation de la complexité.

Il en va de même pour les réseaux de neurones et les estimateurs en général. Une fonction polynomiale de degré deux est plus complexe qu'une fonction linéaire. Un réseau de neurones artificiel à 16 couches est plus complexe qu'un estimateur appartenant à la famille des perceptrons avec une couche dite cachée.

3.1.1 Quels sont les enjeux de la complexité ?

Le lien entre la complexité d'un problème de classification (par exemple) et celle de l'estimateur associé est la performance. Un estimateur plus complexe aura un taux d'erreurs sur les données d'entraînement inférieur à celui d'un estimateur moins complexe.

Pour les besoins de performance, la complexité de l'estimateur doit être « en rapport » avec celle du problème.

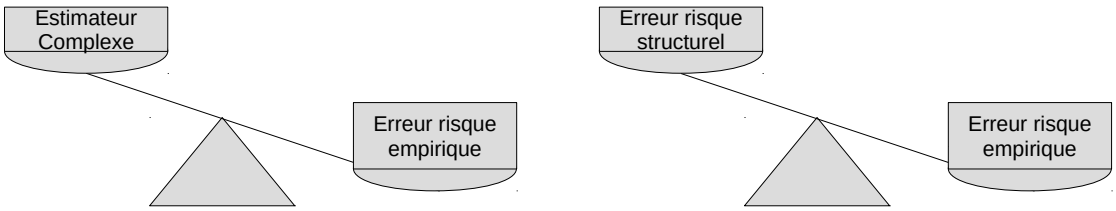
Cependant plus un estimateur est complexe, moins grande est la performance de l'estimateur à généraliser. Le taux d'erreur de généralisation d'un estimateur complexe est supérieur à celui d'un estimateur moins complexe. Les performances de généralisation d'un estimateur décroît avec l'augmentation de sa complexité.

3.1.2 En quoi la complexité nuit-elle à la généralisation ?

Les données utilisées dans la construction de la fonction dite estimateur, ne sont pas parfaites. Elles contiennent du bruit. Ces données sont (toujours) une manifestation imparfaite du problème. La fonction résultat, recherchée dans une famille d'estimateurs d'une certaine complexité (et satisfaisant aux exigences de minimisation du risque structurel) intégrera, en plus de l'information contenue dans les données, le bruit de ces dernières. Ce bruit, intégré dans l'estimateur, dégrade ses performances de généralisation.

L'estimateur doit faire face au dilemme suivant, illustré par le graphique ci-dessous :

- plus il est complexe et plus il sera performant sur les données d'entraînement
- plus il est complexe et moins il sera performant sur les données de test.



3.1.3 D'où vient le bruit des données ?

D'une part, il est possible que les données ne soient pas assez riches, qu'elles soient en quantité insuffisantes pour représenter toute la complexité du problème dont elles sont une manifestation. S'il n'y a pas assez d'images de chiens pour entraîner un algorithme de prédiction de l'appartenance d'un chien à sa race, alors l'algorithme s'acquittera imparfaitement de sa tâche. Il est aussi possible que bien que nombreuses, ces données soient dépendantes les unes des autres. Cette dépendance entre les données n'apporte pas d'information supplémentaire à l'algorithme pour que celui-ci apprenne la fonction de prédiction. Pire, ces corrélations vont amplifier le bruit. Il y a aussi le fait qu'issues de mesures imparfaites, ces données intègrent une partie d'erreur, du bruit. C'est le cas par exemple de données issues de capteurs imprécis.

Une fonction trop complexe présente un risque, celui d'apprendre parfaitement sur des données d'entraînement bruitées. La fonction apprise aura apprise de l'information, mais aussi du bruit. Ce bruit appris peut être tels que la fonction apprise se révèle peu performante dans sa capacité de prédiction sur des données non apprises. Il en résultera une fonction apprise avec une grande variance, susceptible de prédire du bruit.

Si la fonction apprise est constituée d'un ensemble de fonctions, le risque d'erreur de généralisation est dit risque structurel.

- La mesure du risque empirique mesure le risque de faire des erreurs sur les données d'apprentissage (les données d'entraînement).
- La mesure du risque structurel mesure le risque de faire des erreurs de généralisation (des erreurs sur des données n'appartenant pas aux données d'apprentissage).

3.1.4 Comment mesure t-on la complexité d'une classe de fonctions?

Dans les paragraphes précédents, un lien a été établie entre la complexité d'un estimateur et ses performances.
Ce concept de complexité a été mis en évidence dans l'inégalité qui sanctionne le principe de minimisation du risque structurel. Plus précisément, la complexité est mis en lumière dans le terme ***C(F)*** de l'inégalité :

$$\forall f \in F : \Re(f) \leqslant \hat{\Re}(f, S) + C(F)$$

Vapnik et Chervonenkis, en définissant la dimension VC d'une famille d'estimateurs mesurent la complexité de cette famille.

Avec la dimension VC, il a été établie l'inégalité suivante pour une classe de fonctions binaires (i .e à valeur dans [-1, +1] : [5]:

$$\forall f \in F : \Re(f) \leqslant \hat{\Re}(f, S) + \sqrt{\frac{8\,VC \ln\left(2\frac{em}{VC}\right) + 8\ln\left(\frac{4}{\delta}\right)}{m}}$$

Avec les notations :

$$VC : la\,dimension\,de\,Vapnik - Chervonenkis\,d'une\,classe\,d'estimateurs$$
$$m : nombre\,de\,données\,d'entraînement$$
$$\delta : précision\,avec\,laquelle\,l'inégalité\,est\,vraie\,en\,probabilité$$
$$e : nombre\,d'Euler\,avec\,\ln(e) = 1$$

Cette équation est donnée à titre inducative. Elle donne, dans le formalisme établis par Vapnik et Chervonenkis, une valeur à ***C(F)***.

On notera le résultat remarquable que pour $\lim_{m \to \infty} C(F) = 0$

le principe de minimisation du risque structurel est alors exprimé dans l'inégalité :

$$\forall f \in F : \Re(f) \leqslant \hat{\Re}(f, S)$$

Les chercheurs n'ont cesse d'optimiser la valeur de ***C(F)***. Son estimation avec la dimension de Vapnik-Chervonenkis a montré à l'usage ses limites. A cette mesure de la complexité d'une famille d'estimateurs a été substituée la mesure de la complexité de Rademacher.

3.1.5 Pourquoi la complexité de Rademacher?

Pour information, Hans Rademacher était un mathématicien Allemand, contemporain d'Albert Einstein et a lui aussi, enseigné à l'université de Princeton.

Rademacher a mis au point une mesure de la complexité d'une fonction construite à partir de données comme la quantité d'information que cette fonction est en capacité d'extraire de ces données. Dans le cadre du principe de minimisation du risque structurel, la mesure de Rademacher permet d'obtenir de meilleurs résultats que la dimension de VC.

Dans [2], Peter L. Barlett et Shahar Mendelson montrent que la complexité de certaines classes de fonctions peut être utilisée conjointement avec l'erreur empirique, pour la majoration de l'erreur de généralisation. « *La borne majorant l'erreur de généralisation est alors calculable sur une base d'entraînement quelconque* » [5]. Autrement dit, quelque-soit la taille des données d'entraînement

utilisées pour construire l'estimateur, l'erreur de généralisation peut être bornée ce, en utilisant des estimateurs dans une famille dont la complexité est mesurée avec le formalisme de Rademacher.

3.1.6 Comment la complexité est elle réduite ?

La complexité du réseau limite sa capacité de généralisation, i.e sa capacité à prédire correctement des données n'ayant pas été utilisées dans la phase d'apprentissage. Ces données sont les données dites de test. Un compromis entre l'erreur de généralisation et l'erreur empirique est trouvé en réduisant la complexité de l'estimateur.

Dans [2], l'article décrit l'application de la propriété de pouvoir borner l'erreur de généralisation indépendamment de la taille des données d'entraînement aux réseaux de neurones. Fait important, dans l'article, un estimateur est exprimé comme une combinaison linéaire d'estimateurs de base. Ces estimateurs appartiennent à des classes de fonctions, classes différenciées par la valeur de leur complexité.

L'article montre alors comment la majoration de l'erreur de généralisation peut être utilisée dans l'algorithme de construction d'un estimateur pour pénaliser (régulariser) la complexité des estimateurs de base sélectionnés dans le processus de construction.

3.2 Garanties d'apprentissage

La sélection des sous-réseaux candidats est pilotée par la garantie d'apprentissage.
La formulation du problème garantie que le réseau construit étape par étape apprend, i.e qu'en fin d'apprentissage, le réseau résultat est celui qui aura le plus minimisé la fonction de coût.

Cette garantie vient du fait que la fonction à optimiser pour faire décroître l'erreur de prédiction à chaque étape est strictement convexe. Elle garantie donc l'existence d'une solution unique, de par le fait de l'existence d'un minimum globale.

4 Modélisation ADANET d'un réseau de neurones

4.1 Le modèle ADANET d'architecture d'un réseau de neurones

Dans ce modèle, un estimateur est un sous-réseau de neurones et l'estimateur final est composé d'un ensemble de sous réseaux.

Un réseau de neurones en couches est une approximation d'une fonction de prédiction, notée ***f***.

Dans [1], cette dernière est exprimée comme un combinaison linéaire de sous-réseaux h_j, chacun des ***j*** sous-réseaux représente une couche de l'ensemble :

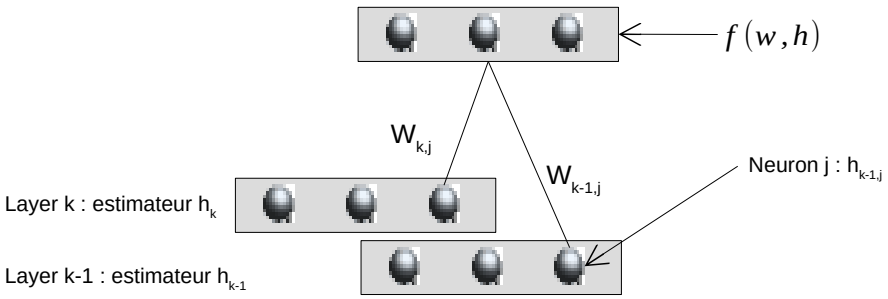
$$f(w,h)=\sum_{j}^{N} w_j h_j$$

Dans cette expression, les poids W_j modélisent la contribution d'une couche ***j*** aux autres couches de numération inférieure.

En prenant en compte un nombre de neurones k par couche, la dernière fonction peut être récrite :

$$f(w,h)=\sum_{j}^{N}\sum_{k}^{M} w_{j,k} h_{k,j}$$

Le graphique ci-dessous donne une représentation de ce modèle :



4.2 Régularisation de la complexité de Rademacher

Cette représentation différencie la contribution de chacune des couches d'un réseau de neurones à l'estimateur ***f***. En mettant en œuvre la complexité de Rademacher, cette différentiation permet d'obtenir une majoration plus précise de la borne supérieure de l'inégalité de minimisation du risque structurel que celle établie en [2].

Avec la formulation d'un estimateur adoptée précédemment, l'inégalité devient :

$$\Re(f) \leqslant \hat{\Re}(f,S) + \frac{4}{\rho}\sum_{k}^{l} \|W_k\|_1 \mathbb{C}(\widetilde{H_k}) + \frac{2}{\rho}\sqrt{\frac{\ln(l)}{m}} + K(l,m,\ldots)$$

Avec les notations suivantes :

$$\mathbb{C}(\widetilde{H_k}) : complexité\ de\ Rademacher\ de\ la\ famille\ d'\ estimateurs\ \widetilde{H_k}$$
$$f : estimateur$$
$$S : jeux\ d'\ entraînement\ de\ dimension\ m$$
$$m : la\ dimension\ du\ \ jeux\ de\ données$$
$$l : nombre\ de\ couches\ du\ réseau\ de\ neurones$$
$$K : une\ expression\ telle\ que \lim_{m \to \infty} K(m) = 0$$

L'expression précédente a ceci de remarquable :

- tous les termes de majoration de l'erreur empirique ne dépendent que de la structure de l'estimateur et du jeux de données.
- La complexité de Rademacher est régularisée (pénalisée) par la norme 1 des poids W_k. Cette régularisation découle « naturellement » de la formulation du modèle de réseau de neurones établie en 4.1

Cette régularisation signifie que les estimateurs de base h_k composant l'estimateur ***f(w,h)*** peuvent être recherchés dans des familles d'estimateurs de complexité croissante sans pour cela nuire à la capacité de l'estimateur à généraliser.

En effet, cette régularisation de la fonction de coût a pour effet de rendre les valeurs des poids ***W*** d'autant plus faible que les estimateurs de base h_j seront complexes. Cette propriété garantie que l'estimateur résultat peut être construit en s'enrichissant d'estimateurs de base de plus en plus complexes et que l'erreur de généralisation sera le reflet de l'erreur empirique. C'est la garantie d'apprentissage dont il est question dans [1].

4.3 Minimisation de la fonction objective

L'apprentissage d'un réseau de neurones passe par la résolution du problème d'optimisation consistant en la minimisation d'une fonction de coût, encore dite fonction « objective ».

> Une telle fonction n'a pas de propriété d'objectivité par opposition à la subjectivité. C'est un anglicisme qui se réfère à une fonction cible dont la solution par optimisation donne le meilleur résultat.

Pour l'algorithme ADANET, la fonction objective $F_t(X,W)$ dérive de l'inégalité établie en 4.2 et s'écrit :

$$\Re(f) \leqslant F_t(w, x_i) = \frac{1}{m}\sum_{i=1}^{m} \Phi(1 - y_i . f_t(x_i, w_i)) + \sum_{j=1}^{N} \Gamma_j \|W_j\|_1$$

Avec les notations :

$$\Phi : fonction\ de\ coût\ comme\ la\ cross-entropie\ ou\ les\ moindres\ carrés\ ,\ selon\ la\ nature\ du\ problème\ .$$
$$X_i : la\ ieme\ donnée\ d'\ entraînement$$
$$Y_i : l'\ étiquette\ (classification)\ ou\ la\ valeur\ correspondante\ de\ X_i\ (regression)$$
$$f_t(X_i, W_i) : Valeur\ prédite\ de\ X_i\ par\ l'\ estimateur\ f$$
$$\Gamma_j : approximation\ de\ la\ complexité\ de\ la\ famille\ F\ à\ laquelle\ h_j\ appartient$$

La complexité de Rademacher est approximée comme :

$$\Gamma_j = \lambda\sqrt{l_j} + \beta$$

- $\boldsymbol{\lambda}$ et $\boldsymbol{\beta}$ sont des hyper-paramètres de l'algorithme d'ADANET,
- $\boldsymbol{l_j}$ le nombre de couches de l'estimateur $\boldsymbol{h_j}$

Pour trouver une solution en $\boldsymbol{W}$ à la fonction de coût $\boldsymbol{F_t(X,W)}$ l'algorithme de descente utilisé est dit de descente par blocs de coordonnées [8]. Cette famille d'algorithmes est emprunté aux méthodes ensemblistes. La fonction de coût définie ici réputée convexe assure l'unicité de la solution.

5 L’algorithme Adanet

5.1 Que fait l’algorithme Adanet ?

Adanet construit un réseau de neurones à partir de sous-réseaux candidats. Le résultat est appelé un ensemble.

5.2 Comment Adanet construit-il l’ensemble ?

Adanet construit l’ensemble à partir de sous-réseaux dits candidats qui lui sont fournis. La construction des sous réseaux n’est pas de la responsabilité d’Adanet. Adanet est responsable de la construction de l’ensemble. C’est au développeur de construire les sous réseaux candidats. Pour se faire, le développeur implémente une classe qui hérite de l’interface *adanet.subnetwork.Builder*.
La sélection du meilleur sous-réseau candidat est du ressort de l’algorithme Weaklearning.

5.3 Quel est le type des sous-réseaux candidats ?

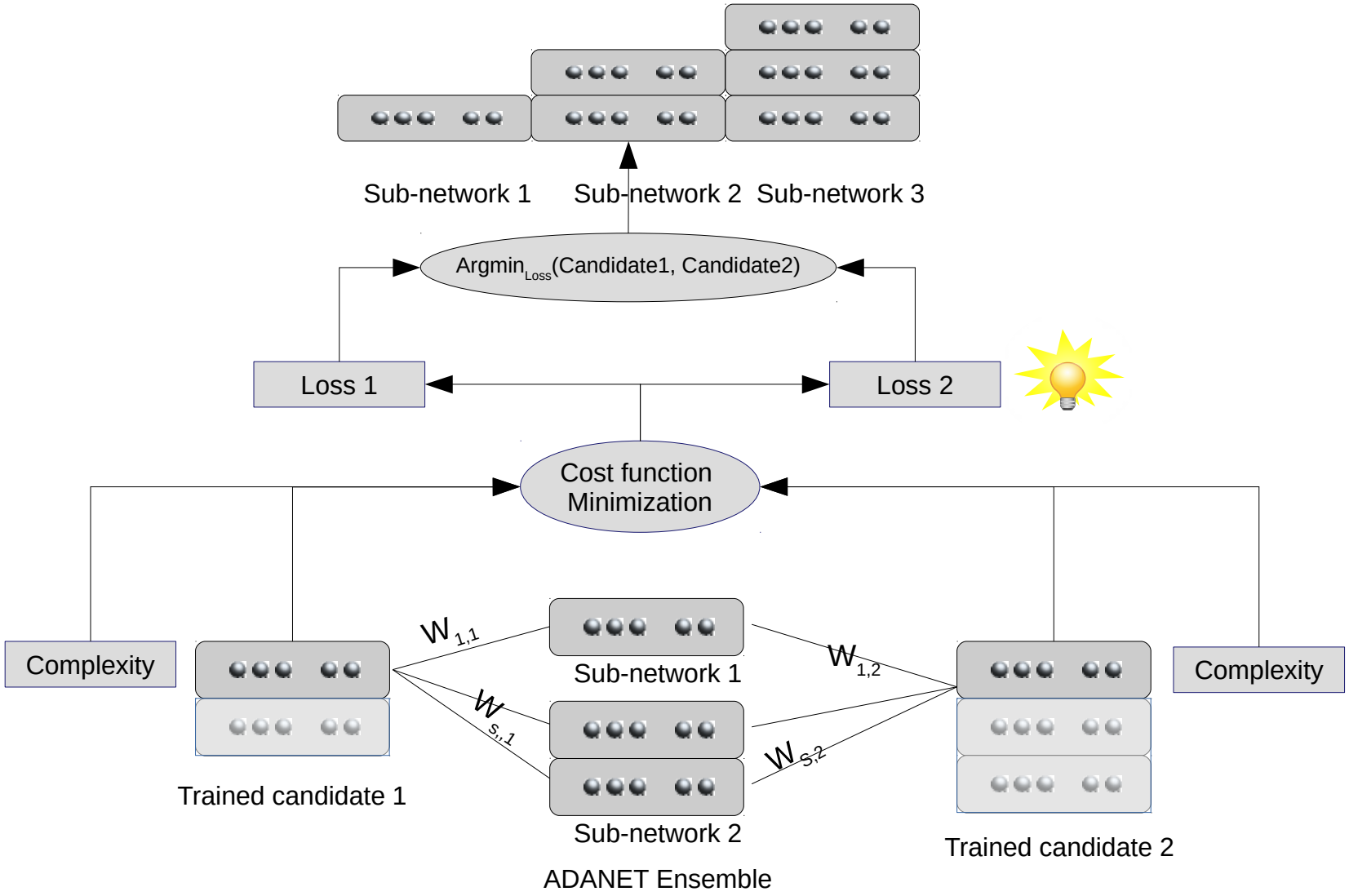
Un sous-réseau candidat est constitué de neurones en couches. A ce titre, les sous réseaux candidats peuvent être de type DNN, CNN, RNN, GAN… La formulation établie en 4.1 autorise à considérer tout type de réseaux organisés en couches de neurones.

A chaque étape de la construction, Adanet demande à tester un sous réseau. Si les conditions du test sont satisfaites (voir ci-dessous), le sous réseau candidat intègre l’ensemble et un nouvel ensemble est ainsi constitué.

5.4 Comment Adanet sélectionne t-il les candidats ?

Le critère de sélection est la valeur de la fonction de coût. A l’étape T+1, la fonction de coût est calculée en intégrant à l’ensemble, le sous réseau candidat. Si la fonction dernière est inférieure à la fonction de coût de l’ensemble à l’étape T, alors le sous-réseau candidat est sélectionné pour faire partie de l’ensemble. Un nouvel ensemble est ainsi créé. Dans le cas contraire, le sous-réseau candidat est rejeté et Adanet demande à tester un nouveau candidat.

En utilisant la valeur de la fonction de coût de l’ensemble comme critère de sélection des sous réseaux candidats, on s’assure qu’à chaque étape, l’erreur de généralisation du nouvel ensemble sera inférieure à celle de l’ensemble à l’étape précédente. Ce, du fait que la réduction de la fonction objective réduit aussi la complexité du nouvel l’ensemble.



L'algorithme weaklearner d'ADANET

6 Applications

L'algorithme Adanet est ici appliqué au problème de classement de chiens en race, ce, à partir d'une image.
Le jeu de données est décomposé de la façon suivante :

- Il y a trois races de chiens
- Le total d'images utilisées pour l'entraînement de l'algorithme est de l'ordre de 400
- Le total d'images utilisées pour l'évaluation de l'algorithme est de l'ordre de 40.

La phase d'entraînement est composée d'étapes. La fonction de coût décroît à chaque étape.
A chaque étape de la phase d'entraînement, une partie seulement des données d'entraînement est utilisée. Il s'agit d'un lot ou batch en anglais.
La période est l'unité au cours de laquelle toutes les données d'entraînement sont utilisées.

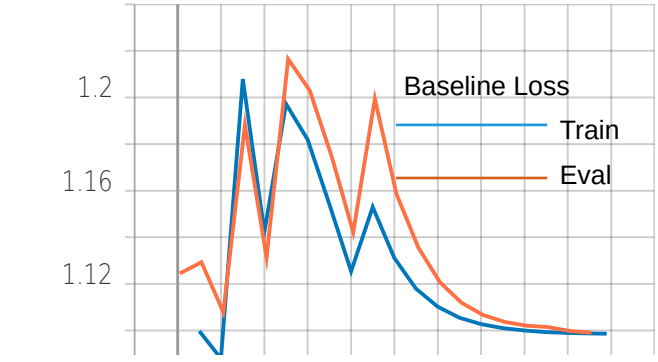
Plusieurs configurations de sous réseaux sont testées.

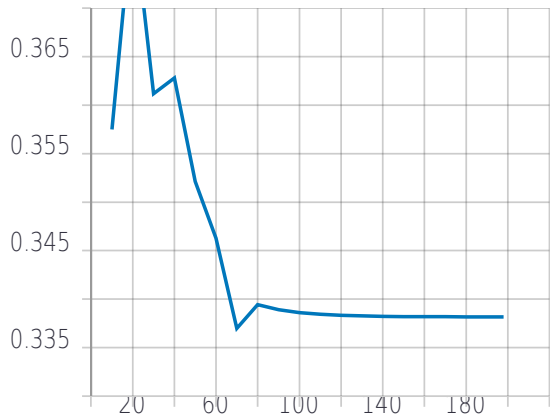
L'environnement d'exécution est Google Cloud Platform, la machine virtuelle instanciée a été configurée avec 56GB de mémoire, 8 CPU.
Les algorithmes ont été implémentés avec Tensorflow 1.13. Les visuels ont été produits avec Tensorboard.

L'initialisation de Xavier est mis en œuvre en étape préliminaire de l'algorithme. Cette disposition permet de prévenir une trop forte augmentation ou atténuation du signal au fur et à mesure que ce dernier se propage dans les couches profondes du réseau.

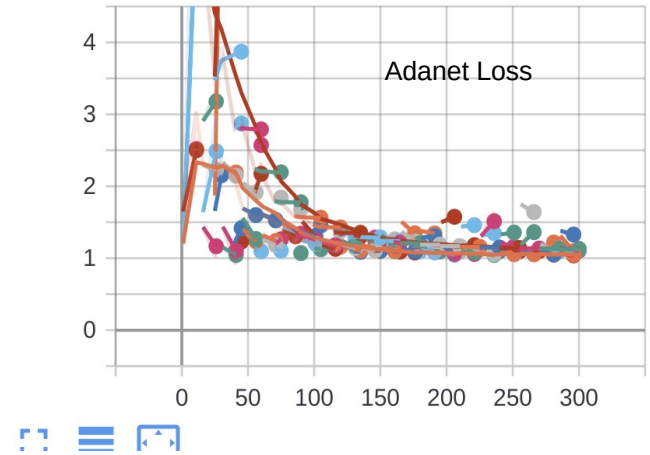
La technique de bacth normalisation est appliquée en entrée de chacune des couches des réseaux candidats est appliquée afin d'augmenter la vitesse d'apprentissage de chacun des sous-réseaux candidat.

6.1 Application à un réseau DNN

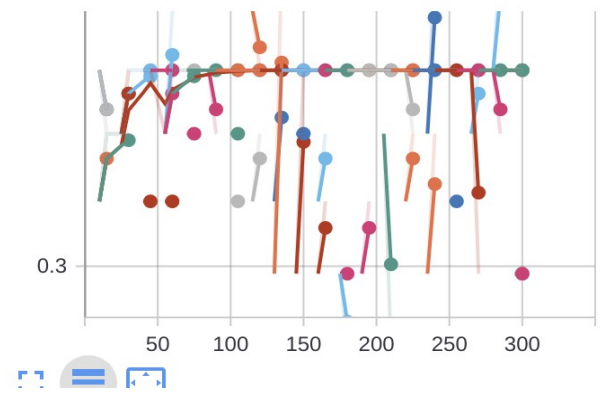




loss



accuracy



Les diagrammes ci-dessus mettent en évidence la différence de comportement di réseau ADANET.
La fonction de coût d'ADANET décroît régulièrement avec le nombre d'itérations. Les points de part et d'autres des courbes des fonctions de coûts sont les contributions de tous les sous-réseaux candidats à l'ensemble. L'enveloppe inférieure des fonctions de coût représente la fonction de coût de l'ensemble constitué de sous-réseaux.

6.2 Application à un réseau CNN

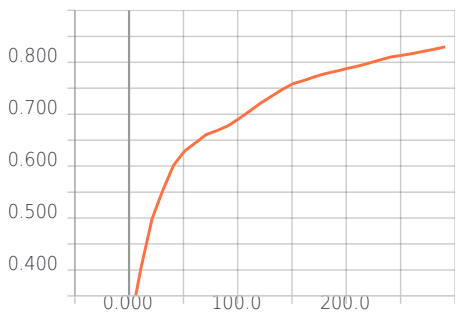
Le dataset sur lequel l'algorithme ADANET est appliqué contient un nombre relativement modeste de paramètres.

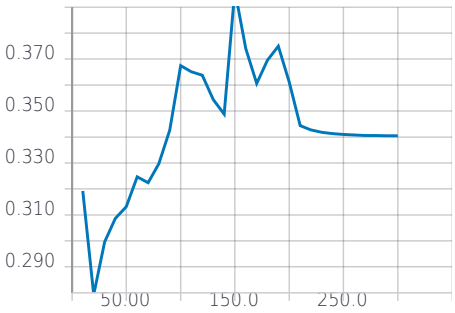
De fait, l'efficacité de cet algorithme est ici mis à l'épreuve dans sa capacité à apprendre un réseau suffisamment complexe pour afficher des performances honorables et suffisamment régulé pour être en mesure de généraliser avec de bonnes performances.

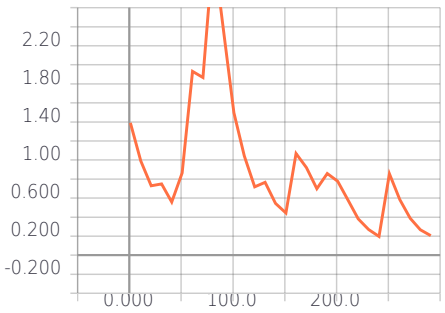
L'utilisation de réseaux pré-entraîné sur un dataset de taille modeste conduit le réseau à sur-apprendre et à overfiter ses performances, i.e à ne pas prédire correctement des données ne faisant pas partie des données d'entraînement.
Habituellement sur des jeux de données modeste, l'algorithme SVM appliqué à la sortie de couches de convolution donne de meilleurs résultats que le ré-entraînement des couches denses.

6.2.1 Baseline de test

Afin de pouvoir mettre en évidence l'apport de l'algorithme ADANET, la baseline mise en œuvre consiste en un réseau CNN constitué de 3 couches de convolution et de deux couches denses.







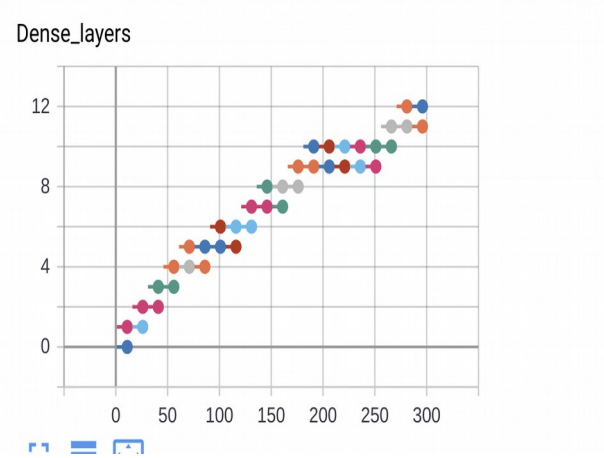
Eval
Train

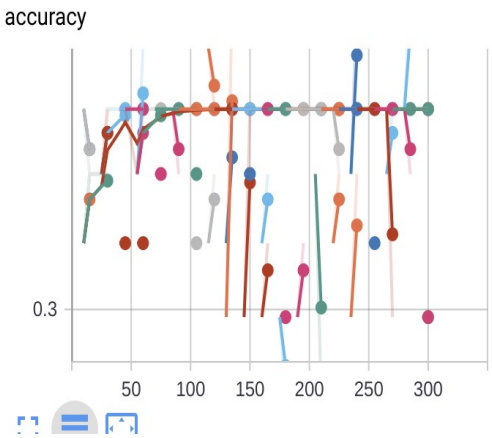
CNN network
Conv layers : 2
Dense layers : 1
Kernel : (5,5)
Filters : 32
Steps : 300

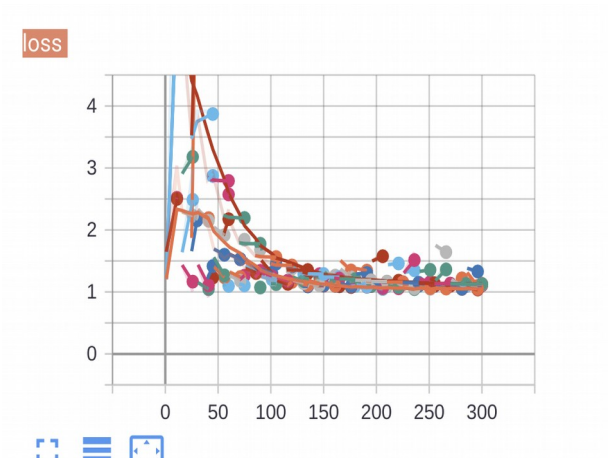
La fonction de coût des données d'entraînement décroît et la précision croît. On constate cependant que la précision sur les données d'évaluation du modèle suivent une courbe erratique.

6.2.2 *Apprentissage adaptatif avec des sous-réseaux denses*

Pour rappel, ces sous réseaux sont constitués de couches de neurones, dits « fully connected » d'une couche à l'autre, comme sur le schéma ci-dessous.
L'effet d'absence du terme de complexité (fixé à zéro) est mis en avant dans les diagrammes ci-dessous :







En haut à gauche, la construction de l'ensemble à partir des pairs de sous-réseaux candidats, fournis à Adanet à chacune des itérations d'Adanet.

En haut à droite, l'évolution de la précision de l'ensemble.

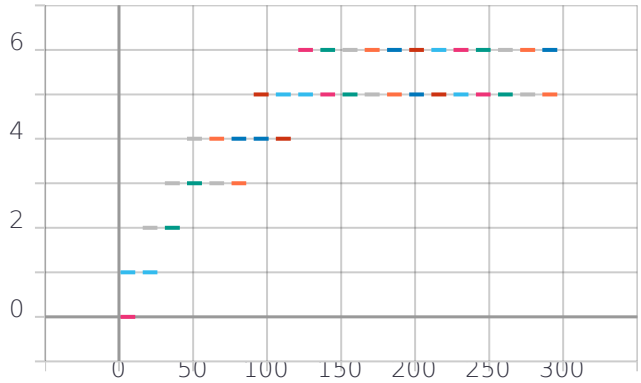
En bas à droite, l'évolution de la fonction de coût.

La régularisation de la complexité a été désactivée. Il en résulte un ensemble résultat dont la complexité augmente avec le nombre d'itérations, jusqu'à atteindre un réseau composé de sous-réseaux candidats à 12 couches denses.

La précision obtenue après un total de 300 étapes décomposées en 20 itérations Adanet, est de l'ordre de 30 %. L'amélioration de cette valeur n'est pas ici, le but recherché. Ce qui est mis en avant, c'est la décroissance de la fonction de perte au fur et à mesure que l'algorithme construit un ensemble à partir de sous réseaux candidats.

Quelle est l'effet de la complexité sur la structure résultante ?

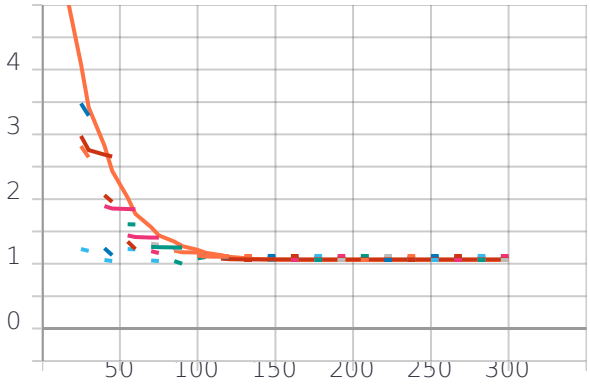
Les diagrammes qui suivent montrent l'effet du terme de régularisation. Ce dernier étant activé, la complexité de l'ensemble résultat est limité par le terme de régularisation. Le nombre maximal couches est limité à 6, en 20 itérations Adanet. Cette valeur est à comparer à la structure de 12 couches lorsque le terme de régularisation est fixé à 0.



Après 130 étapes, la structure ne croît plus en nombre de couches.

Le terme de complexité de Rademacher porte sur le nombre de couches d'un sous-réseau candidat.

Par construction, c'est donc ce nombre de couches qui est pénalisé lors de la construction de l'ensemble.



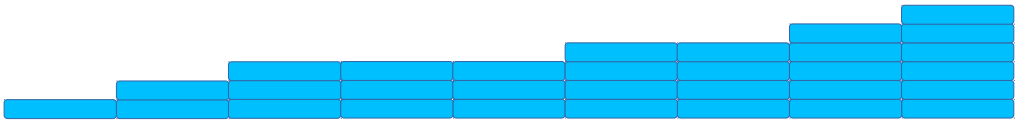
La stagnation de la croissance de la complexité de la structure de l'ensemble, autour de 150 étapes, correspond à la stagnation de la décroissance de la fonction de coût, donc, de la décroissance du risque d'erreur structurel.

Quelle est la structure résultante de l'ensemble ?

La structure construite par Adanet à l'issue de ses itérations est représentée dans Tensorbaoed sous forme textuelle comme ci-dessous :

| CNN_layer_1 | CNN_layer_2 | CNN_layer_3 | CNN_layer_3 | CNN_layer_3 | CNN_layer_4 | CNN_layer_4 | CNN_layer_5 | CNN_layer_6 |

Cette structure peut être représentée par un schéma en couches comme ci-dessous :



Le nombre d'étapes témoigne de la complexité du problème à résoudre. Vu d'une autre perspective, cela revient à décomposer la complexité d'un problème en sous-problèmes plus simples.

> Comparaison n'est pas raison !
>
> C'est de cette façon qu'à l'école, nous nous sommes exercés, implicitement, à résoudre les problèmes complexes posés dans nos livres d'apprentissage. La solution apparaît après avoir résolu une succession de questions intermédiaires. Ces questions sont autant de problèmes « élémentaires » composant le problème global. Ce dernier a été partitionné en une succession de problèmes plus simples.
>
> En poussant l'analogie, nos réponses correctes sont autant de classifieurs élémentaires formant la solution. Ceux d'entre eux n'apportant pas de réponse satisfaisante ont été écartées.
>
> Reconnaissons qu'il est plus facile de résoudre les problèmes plus simples, une fois ceux-ci posés, que de décomposer le problème complexe en intermédiaires plus simples.

Comment, néanmoins, expliquer cette stagnation des performances ?

L'allure des courbes matérialisant les performances montre qu'on améliorerait pas significativement les résultats en laissant Adanet continuer à itérer.

Une première réponse tient sans doute la nature du réseau. Pour un problème de classification d'images ou encore de reconnaissance visuelle, les réseaux de convolution donnent de meilleurs résultats. Dans ce cas de figure, les sous-réseaux candidats sont des réseaux denses.

Une seconde réponse peut être avancée. Le nombre d'images utilisées pour l'entraînement, de l'ordre de 400, est faible eu regard à la complexité inhérente de ce problème de classification. Ne perdons pas de vue qu'en entraînant le réseau, on construit une fonction de prédiction à partir des données d'entraînement. Cette fonction est représentée par un ensemble de paramètres. Ces paramètres sont ici, les poids et les biais ajustés itération après itération dans les neurones formant les couches des sous-réseaux candidats. Ces paramètres sont d'autant plus représentatifs de la fonction de prédiction que le nombre de données d'entraînement est élevé. Ce qui n'est probablement pas le cas ici, de par le faible nombre d'images utilisées pour l'entraînement. Le problème mis en œuvre est sans doute trop simple, auquel cas, il est dit biaisé. Les données d'entraînement sont dites non stationnaires, car elles ne sont pas représentatives de chacune des races de chien que l'on cherche à classer.

Cette hypothèse de représentativité est fondamental dans la formulation des problèmes d'apprentissage. En général, on s'assure de travailler avec des jeux de données suffisamment importants pour satisfaire l'hypothèse de stationnarité.

Ce point est amplement détaillé dans le premier chapitre de [5].

6.3 Application à un réseau de neurones récurrent dit « RNN »

Un réseau RNN est habituellement constitué d'une cellule à l'intérieur de laquelle, plusieurs couches de neurones interagissent entre elles. Cette interaction est le fait différenciateur d'un réseau dit « feed-forward ».

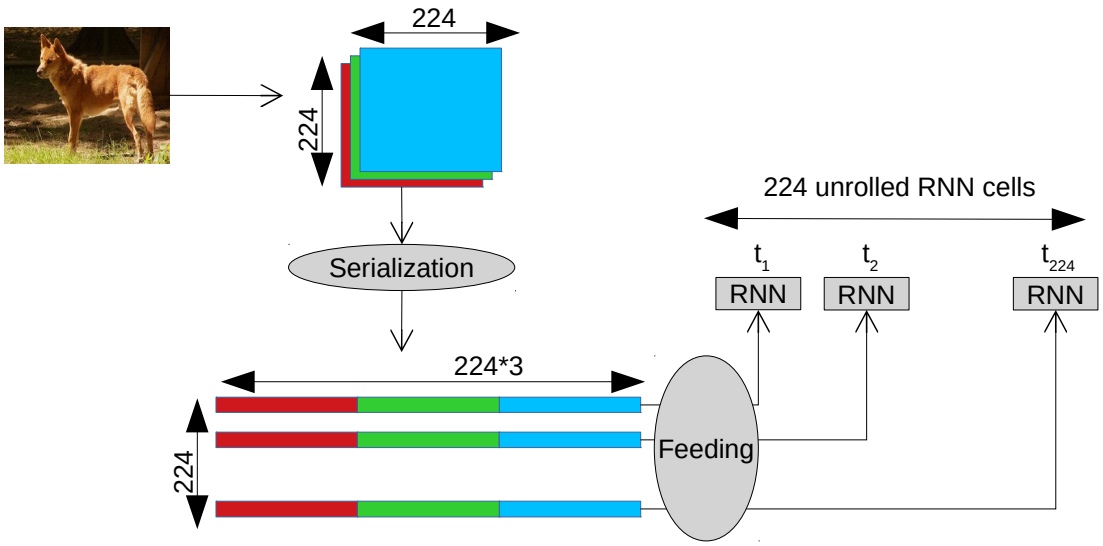
Pour modéliser le fonctionnement en séquences dans le temps d'un réseau RNN, la cellule est déroulée par tranches de temps, en couches. Les couches ainsi « déroulées » de la cellule, partagent les mêmes valeurs de poids et de biais présents dans chacun des neurones composant la cellule.

Pour bénéficier des avantages de la profondeur des réseaux de neurones obtenus en empilant des couches de neurones, il est tout autant possible d'empiler des cellules de réseaux récurrents [6]. C'est sur ce principe que sont construits les sous-réseaux candidats qui alimentent l'algorithme Adanet.

6.3.1 Sérialisation des images

Chacune des images va être traitée comme une série temporelle. Pour ce faire, chaque image est découpée en « tranches » de pixels. Ce découpage rend dépendantes les séquences successives de pixels composant l'image. Cette sérialisation rend possible la résolution du problème de classification par un estimateur en réseaux récurrent de neurones (RNN).

La figure ci-dessous illustre le processus pour une image en couleur.

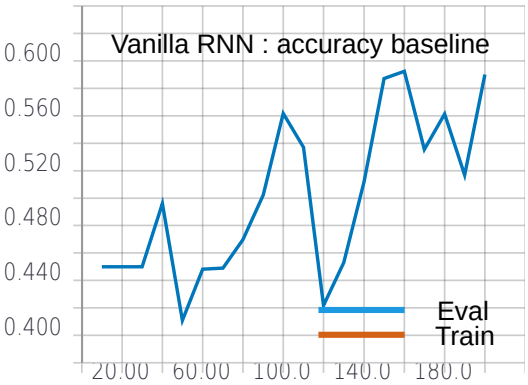


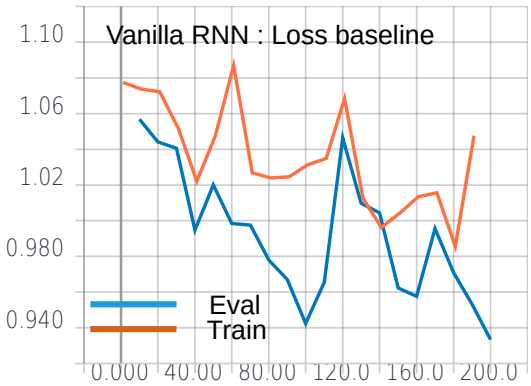
L'image de dimension (224*224*3) avec les trois plans couleurs est transformée en une matrice (224*776). Une cellule RNN est déroulée, dans le temps, en 224 étapes successives. Chacune de ces étapes est alimentée par un vecteur de dimension 224*3=776 pixels.

Ainsi, la dernière tranche de pixels, à l'étape t_{224} dépend de toutes les précédentes, allant de t_1 à t_{223}. Cette dépendance est modélisée par la relation de récurrence qui existe au sein de la structure d'une cellule RNN formant réseau de neurones.

6.3.2 Application aux cellules de type «Vanilla RNN »

Ce sont les premières formes de réseau récurrents parmi la famille de réseaux récurrents.
Les résultats ci-dessous sont affichés pour la baseline et le réseau ADANET.



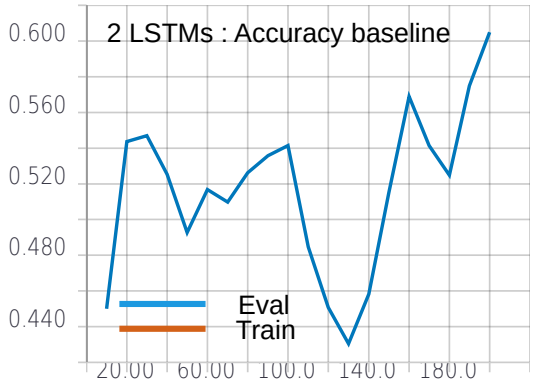


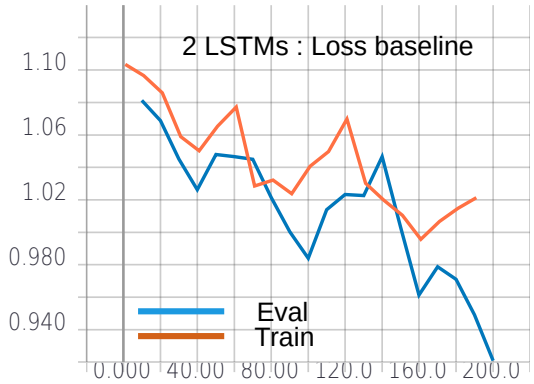

```
NN type              : ..................... RNN
Units in dense layer : ..................... 10
Number of layers     : ....................... 1
Dropout rate         : ..................... 0.0
Seed value           : ..................... 42
Nb of classes (logit): ....................... 3
Weights initializer  : ......... truncated_normal
Batch normalization  : ..................... True


Cell type            : ..................... RNN
Hidden units         : ..................... 128
Stacked cells        : ....................... 1
Time steps           : ..................... 224
```

Les cellules Vanilla RNN ne sont pas conçues pour traiter des données en séries temporelles avec des dépendances de long terme, i.e des dépendances sur un nombre d'étapes importantes.

6.3.3 Application aux cellules de type «LSTM RNN »

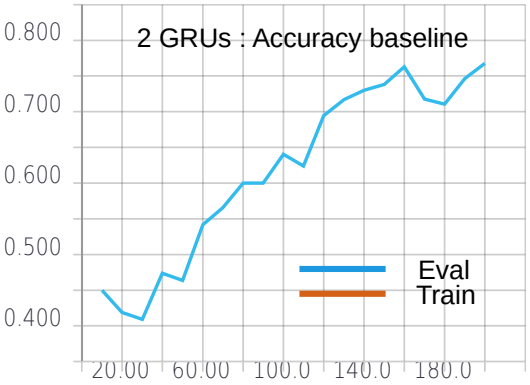


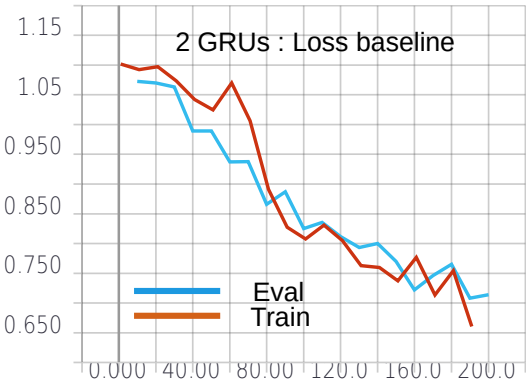



```
NN type              : ..................... RNN
Units in dense layer : ..................... 10
Number of layers     : ...................... 2
Dropout rate         : ..................... 0.0
Seed value           : ..................... 42
Nb of classes (logit): ...................... 3
Weights initializer  : ....... truncated_normal
Batch normalization  : .................. True

Cell type            : .................. SLSTM
Hidden units         : ..................... 128
Stacked cells        : ...................... 2
Time steps           : ..................... 224
```

6.3.4 Application aux cellules de type «GRU RNN »



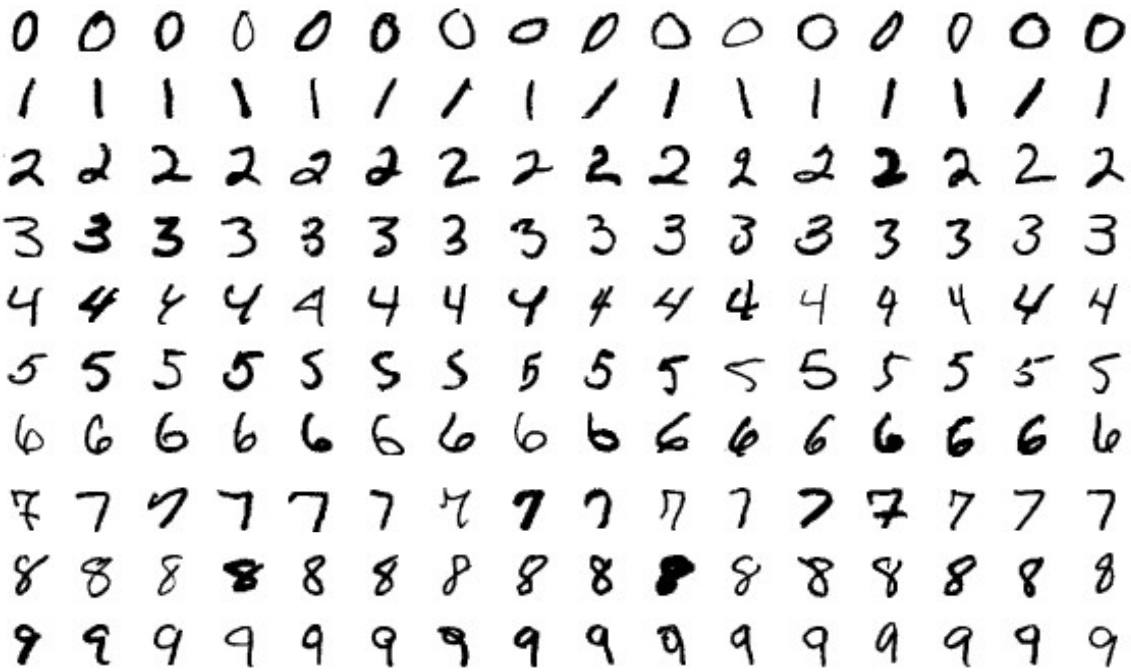



```
NN type              : ..................... RNN
Units in dense layer : ..................... 10
Number of layers     : ....................... 2
Dropout rate         : ..................... 0.0
Seed value           : ..................... 42
Nb of classes (logit): ....................... 3
Weights initializer  : ...... truncated_normal
Batch normalization  : ................... True

Cell type            : ..................... GRU
Hidden units         : ..................... 128
Stacked cells        : ....................... 2
Time steps           : ..................... 224
```

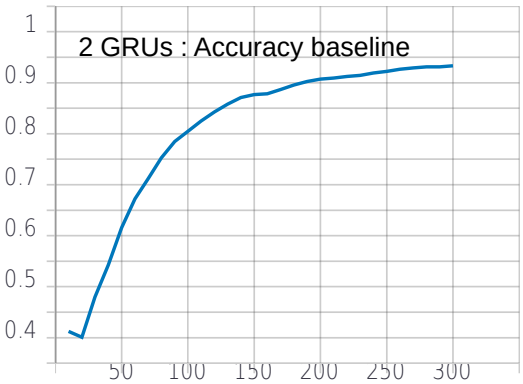
6.3.5 ADANET : dataset MNIST

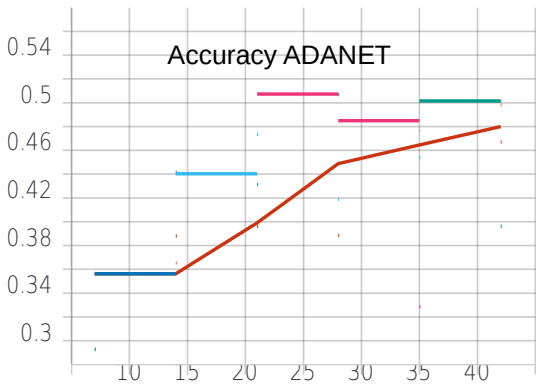
L'application d'ADANET au dataset standford dogs nécessite des ressources importantes. Pour palier le manque de ressources, le dataset MNIST a été utilsé pour tester les performances d'ADANET et regards de celles de la baseline.

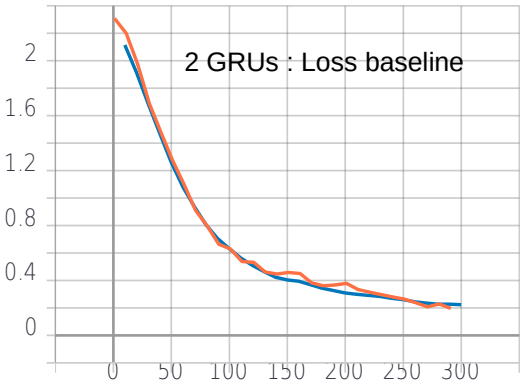


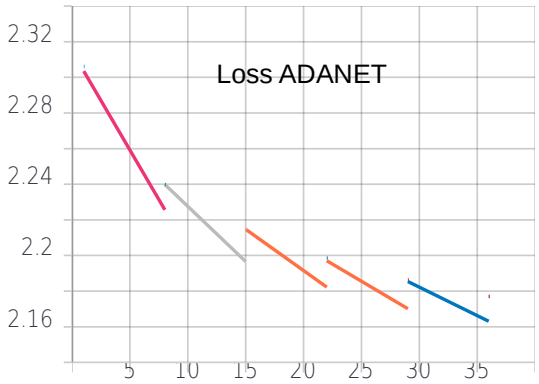
Ce dataset est constitué d'images en noir et blanc, de (28,28) pixels, avec 55000 observations dédiées à l'entraînement des estimateurs et 10000 observations dédiées à l'évaluation des estimateurs.

Les figures ci-dessous présentent les résultats comparés des baseline et ADANET:







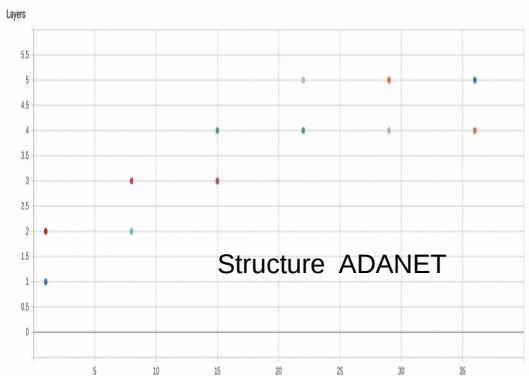


Les paramètres pour l'algorithme ADANET :

```
Global steps  : ................................ 300
NN type             : ........................... RNN
Features shape      : ........................... (28, 28)
Adanet boosting iter.: ........................... 40
Adanet iter per boost: ........................... 7
Dropout rate        : ........................... 0.0
Seed value          : ........................... 42
Nb of classes (logit): ........................... 10
Adanet regularization: ........................... 1e-05
Weights initializer  : ........................... truncated_normal
Batch normalization  : ........................... True
Learn mixture weights: ........................... True
Cell type           : ........................... SGRU
Hidden units        : ........................... 128
Stacked cells       : ........................... 2
Time steps          : ........................... 28
```

L'algorithme ADANET demande beaucoup plus de ressources pour construire l'ensemble de sous-réseaux.
Le temps d'obtention des résultats de la baseline, moins de 3 minutes, sont à mettre en regard avec les 5 heures nécessaires pour produire le résultat d'ADANET.
Les performances d'ADANET sont sans doute du fait qu'un vecteur de « mixture weights » a été utilisé en place d'une matrice de « mixture weights ». Ce choix a été dicté par la difficulté de formulé la notion de dernière couche pour un ensemble RNN. En place de la dernière couche, les logits des candidats ont alimentés l'algorithme ADANET.

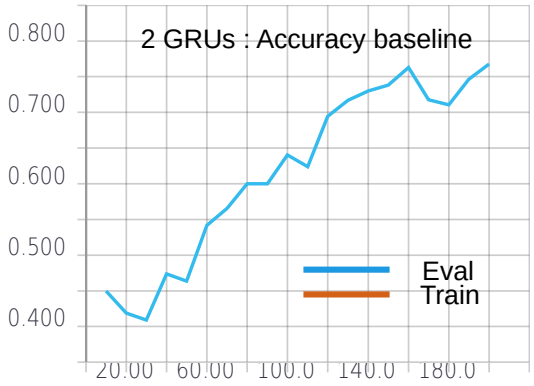
Le diagramme ci-dessous exhibe la structure de sous-réseaux GRU résultante :

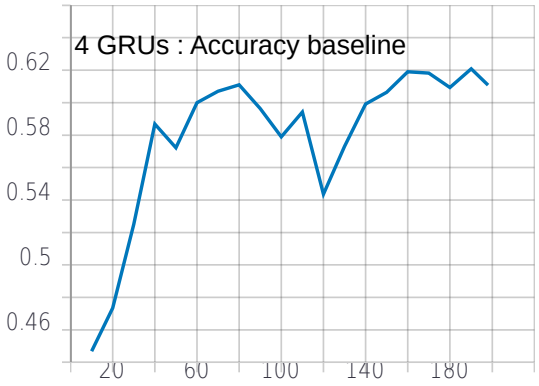


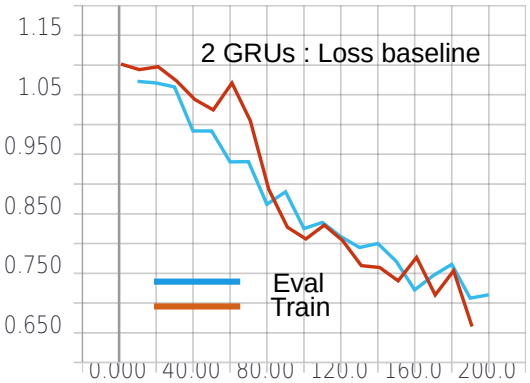
La complexité de l'ensemble croît jusqu'à atteindre un nombre maximal de sous-réseaux candidats à 5 cellules GRU empilées.
Conformément aux résultats attendus quant à la fonction de coût, celle-ci diminue toujours au fil des itération de façon augmenter les performances en précision sur l'ensemble de test.

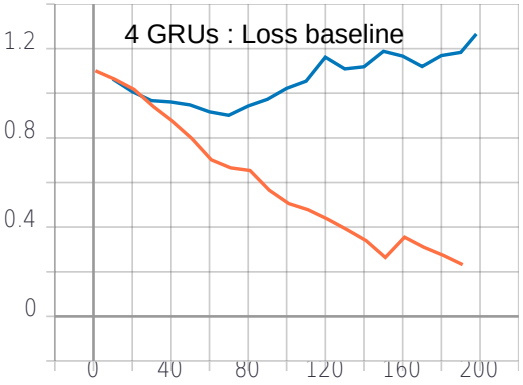
6.3.6 *Effets de la complexité sur un réseau GRU*

Les diagrammes du paragraphe 6.3.4 montrent les relatives bonnes performances d'un réseau GRU en deux couches.
Les diagrammes ci-dessous montrent les effets de l'accroissement de la complexité d'un réseau GRU en passant de deux couches à 4 couches.










```
NN type              : ................... RNN
Units in dense layer : ..................... 10
Number of layers     : ...................... 2
Dropout rate         : ................... 0.0
Seed value           : ..................... 42
Nb of classes (logit): ...................... 3
Weights initializer  : ...... truncated_normal
Batch normalization  : .................. True

Cell type            : ................... GRU
Hidden units         : ................... 128
Stacked cells        : ...................... 2
Time steps           : ................... 224
```

Les diagrammes à droite montrent l'évolution des performances d'un réseau GRU à quatre couches consistant à empiler 4 cellules GRU. La complexité de ce dernier assure une décroissance de l'erreur empirique à un niveau plus bas quel e réseau GRU à 2 couches. Cependant, les performances du réseau à 4 couches sont dégradées quant à sa capacité de généralisation.

7 Conclusions

Pour tous les types de réseaux de neurones étudiés, l'algorithme ADANET met en évidence la propriété de garantie d'apprentissage. La diminution de la fonction du coût étape après étape, calculée sur les données d'entraînement, s'accompagne d'une baisse de l'erreur de généralisation, calculée sur les données de test.
L'algorithme ADANET réalise un bon compromis entre les performances de généralisation et la minimisation du risque empirique.

En inhibant le terme de régularisation de la complexité, la structure du réseau croît avec le nombre d'itérations. Néanmoins, il est remarquable de constater que l'augmentation de la complexité, en termes de nombre de couches des sous-réseaux formant l'ensemble ADANET, ne pénalise pas les performances de généralisation de l'ensemble résultat.

Pour l'application d'ADANET aux réseaux de convolution, la stratégie de croissance des couches de convolution doit être affinée. Une stratégie de croissance des couches denses doit aussi être étudiée. La combinaison de croissance couches de convolution / couches denses devrait aussi faire l'objet d'investigations.

ADANET nécessite néanmoins des ressources plus importantes, en mémoire vive, que ses équivalents en « baseline ».
La valeur de la complexité de Rademacher d'un sous réseau est approximée par la racine carré du nombre de couches. Le modèle mathématiques représentant un réseau de neurones autorise à prendre en compte, dans le calcul de la complexité, le nombre de neurones constituant une couche. Ce point peut être un axe d'amélioration de la performance d'ADANET.

L'hyper-paramètre $\boldsymbol{\lambda}$, régulant la complexité de la solution, nécessite une heuristique pour une optimisation fine du résultat.

Enfin, Il serait pertinent d'entreprendre l'étude de la construction d'un réseau GAN à partir de sous-réseaux du même nom en utilisant l'algorithme ADANET.

ANNEXES

8 Références

[1] AdaNet: Adaptive Structural Learning of Artificial Neural Networks (2017, Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, Scott Yang)

[2] Rademacher and Gaussian Complexities: Risk Bounds and Structural Results (2002, Peter L. Bartlett, Shahar Mendelson)

[3] Combining multiple TensorFlow Hub modules into one ensemble network with AdaNet https://medium.com/tensorflow/combining-multiple-tensorflow-hub-modules-into-one-ensemble-network-with-adanet-56fa73588bb0

[4] Introducing AdaNet: Fast and Flexible AutoML with Learning Guarantees https://ai.googleblog.com/2018/10/introducing-adanet-fast-and-flexible.html

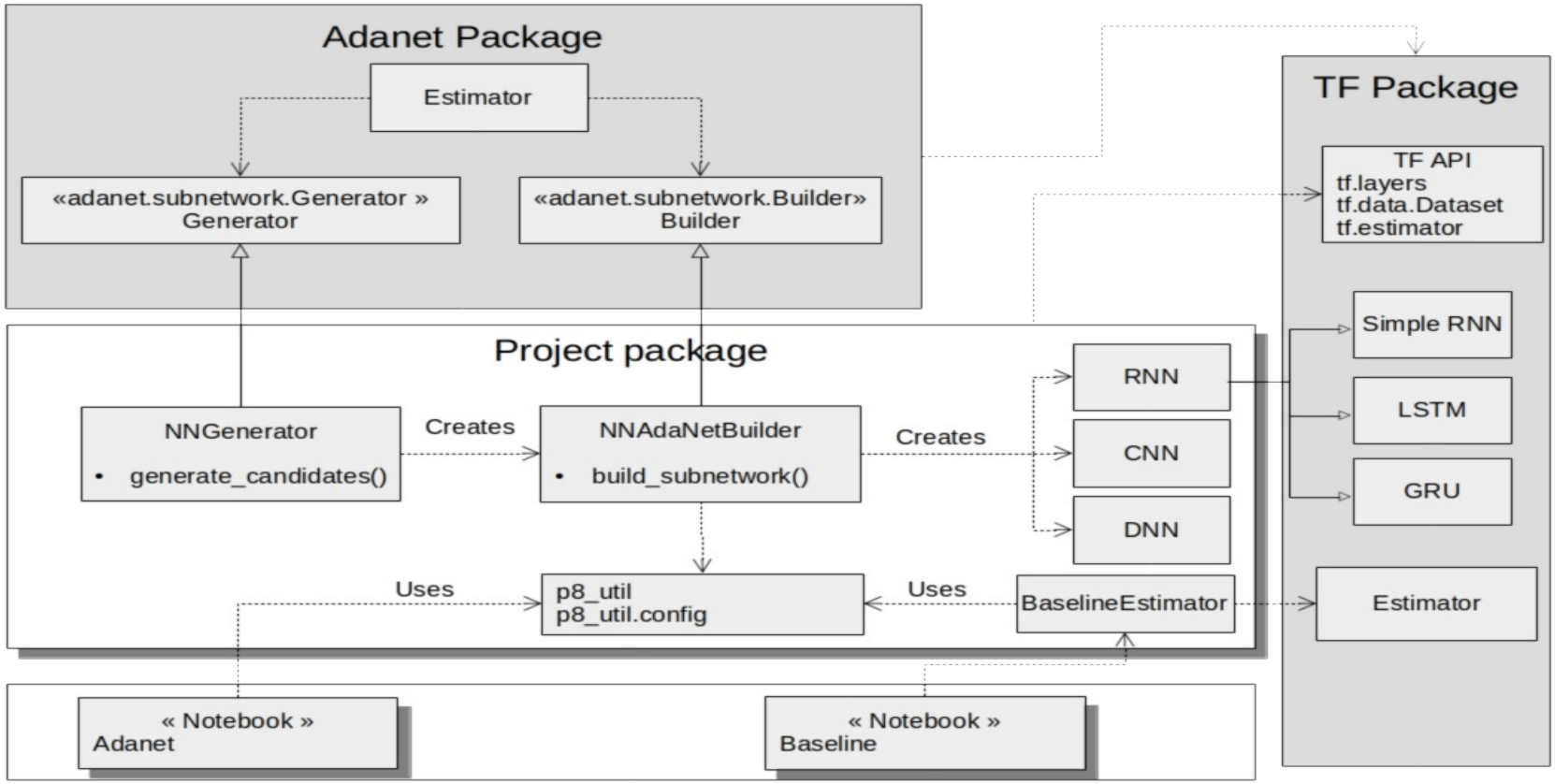
[5] Apprentissage machine : de la théorie à la pratique(Massih-Reza AMINI, Editions Eyrolles, 2015) https://core.ac.uk/download/pdf/51938971.pdf

[6] How to construct a deep RNN neural network (https://arxiv.org/pdf/1312.6026.pdf, 2014)

[7] Adanet documentation (https://buildmedia.readthedocs.org/media/pdf/adanet/latest/adanet.pdf Mai 2019)

[8] : Coordonates Descent Algorithms (Spehen J. Wright) https://arxiv.org/pdf/1502.04759.pdf Feb. 2015.

9 *Ingénierie logicielle*



Les notebook ***Adanet.ipynb*** et ***Baseline.ipynb*** permettent de produire des résultats éponymes.

Ils utilisent tous deux le fichier ***p8_util_config.py*** fixant la configuration de chacune des expérimentations. Ce, dans la perspective de faciliter les comparaisons issues des expérimentations.

Le fichier ***p8_util.py*** implémente les fonctions utilitaires nécessaires aux expérimentations du projet.

Le fichier ***BaselineEstimator.py*** utilise la classe ***Estimator*** de Tensorflow. ***BaselineEstimator*** est une implémentation de l'estimateur customisé de la baseline, utilisant les API ***tf.estimator*** fournies par Tensorflow.

Le fichier ***NNGenerator.py*** implémente la classe ***NNGenerator*** qui est une extension de la classe ***Generator*** du framework ***Adanet***. Cette classe a la charge de renvoyer les sous-réseaux candidats à l'algorithme Adanet. Pour ce faire, elle délègue la fabrication des sous-réseaux à la classe ***NNAdanetBuilder*** implémentée dans le fichier ***NNAdanetBuilder.py.***

La classe ***NNAdanetBuilder*** construit les différents types de sous réseaux, RNN, CNN et DNN. Ces derniers utilisent l'API ***tf.layers*** de Tensorflow pour les classes du même nom.

10 Formulation de la fonction de coût régulée d'ADANET

L'apprentissage, étape par étape, consiste à faire diminuer, étape par étape, la fonction qui mesure l'erreur de la construction.

Le mécanisme dit de régularisation consiste ajouter un terme positif à la fonction de coût (encore appelée fonction de risque).

Supposons que toutes les données existantes et imaginables soient disponible pour construire la fonction. Dans ce cas, le risque d'erreur de généralisation est exactement le risque de l'erreur empirique, puisque la fonction n'aura jamais à traiter de données autres que celles ayant servies à sa construction. Avec cette hypothèse, la fonction du risque empirique s'écrit « simplement » :

- $R(X,Y,w)=\frac{1}{m}\sum_{i=1}^{m}\Phi\left(\sum_{j=1}^{N}\left(w_j\cdot h_j(X_i),Y_i\right)\right)$
  - La fonction Φ est une fonction dite objective, dont l'optimisation mesure la distribution statistique de l'erreur empirique due aux données d'entraînement. Dans le cas d'un problème de classification, la fonction objective peut être la « cross-entropie ». Dans le cas d'un problème de régression traité linéairement, l'optimisation de la fonction objective conduira à la mesure la plus petite de l'erreur distribuée selon une Gaussienne et sera l'expression du problème d'optimisation consistant à la minimisation de l'erreur quadratique.

Mais dans le cas le plus probable, les données ayant servies à construire la fonction ne sont qu'un sous ensemble des données possibles et imaginables que la fonction aura à traiter. A la fonction de risque précédente, est alors ajoutée d'un autre terme pour tenir compte de la complexité et s'écrit :

- $R(X,Y,w)=\frac{1}{m}\sum_{i=1}^{m}\Phi\left(\sum_{j=1}^{N}\left(w_j\cdot h_j(X_i),Y_i\right)\right)+\sum_{j=1}^{j=N}\left(\lambda\cdot r\left(h_j(X_i)\right)+\beta\right)\cdot\|W_j\|_{L1}$
  - Le terme $r(h)$ mesure la complexité de Rademacher de la $j^{ème}$ fonction contribuant à l'ensemble, soit, h_j
  - Les termes λ et β sont des hyper-paramètres, qui permettent, avec du savoir-faire, de la connaissance sur les données ou encore, beaucoup de ressources, d'ajuster la contribution de la complexité au risque d'erreur de généralisation.

La réduction de cette fonction R implique aussi la réduction de la complexité par la réduction du terme supplémentaire qui lui a été ajoutée.

11 Heuristique Adanet

Adanet permet d'apprendre un réseau à partir de sous réseaux candidats. Cet algorithme peut être rattaché à la classe des méthodes ensemblistes. Ces méthodes construisent un bon estimateur à partir d'apprenants dits faibles. Ces apprenants faibles font juste un peu mieux qu'une prédiction aléatoire. Pour un problème de classification binaire aura une performance d'un peu plus de 50 %.

La recette présentée ici, consiste à évaluer la structure d'un apprenant faible. Une fois fixée, cette structure est utilisée pour la fabrication des sous-réseaux candidats qui alimentent Adanet.