

PROJET MACHINE LEARNING

TOUZENE Yasmine
RANAIVO HARISON Manitra Andr  a
QU  RET Olivier

Groupe 8 – 4IBD

Table des matières

LE MACHINE/DEEP LEARNING	3
<i>Introduction</i>	<i>3</i>
<i>De l'humain au deep learning.....</i>	<i>4</i>
<i>Introduction aux réseaux de neurones</i>	<i>5</i>
<i>Définition.....</i>	<i>6</i>
<i>Principe</i>	<i>7</i>
LES DIFFERENTS RESEAUX DE NEURONES CLASSIQUE.....	8
LE PERCEPTRON SIMPLE (PS).....	8
<i>Définition.....</i>	<i>8</i>
<i>Caractéristique.....</i>	<i>9</i>
<i>Fonctionnement mathématique</i>	<i>10</i>
<i>Les fonctions de calcul de W.....</i>	<i>10</i>
LE PERCEPTRON MULTICOUCHE (MULTI LAYER NETWORK)	11
<i>Définition.....</i>	<i>11</i>
<i>Caractéristique.....</i>	<i>12</i>
<i>Fonctionnement mathématique</i>	<i>12</i>
RESEAUX DE NEURONES CONVOLUTIFS (CNN)	15
<i>Définition.....</i>	<i>15</i>
<i>Fonctionnement.....</i>	<i>16</i>
RESEAUX DE NEURONES RECURRENTS (RNN).....	17
<i>Définition.....</i>	<i>17</i>
<i>Comment les réseaux de neurones récurrents apprennent.....</i>	<i>17</i>
<i>Unités de mémoire à court terme</i>	<i>19</i>
RESEAU HYBRIDE (MELANGE DE CNN & RNN)	20
<i>Définition.....</i>	<i>20</i>
<i>Architecture et fonctionnement.....</i>	<i>20</i>
ARCHITECTURES RECENTES D'APPRENTISSAGE EN PROFONDEUR	22
RESEAU RESIDUEL (RESNET).....	23
<i>Définition.....</i>	<i>23</i>
<i>Fonctionnement mathématique</i>	<i>23</i>
COMPARAISON DES DIFFERENTS MODEL SELON LEURS CAS D'USAGE	25
LES DIFFERENTES FONCTIONS UTILISEES.....	27
INTRODUCTION	27
OPTIMISEUR.....	27
<i>Algorithme qui permet de minimiser la fonction de perte pondérée.</i>	<i>27</i>
<i>Dropout.....</i>	<i>29</i>
<i>Fonctions de perte dans les réseaux de neurones</i>	<i>29</i>
<i>Activation.....</i>	<i>30</i>
LES METRIQUES.....	31
<i>Métriques pour évaluer les algorithmes d'apprentissage automatique en Python</i>	<i>31</i>
<i>Categorical_accuracy.....</i>	<i>31</i>
<i>Sparse_categorical_accuracy</i>	<i>31</i>
<i>Confrontation.....</i>	<i>32</i>
<i>Précision binaire:.....</i>	<i>32</i>
<i>Exactitude catégorique :</i>	<i>32</i>
<i>Précision catégorique clairsemée :</i>	<i>32</i>
<i>Précision catégorique maximale :</i>	<i>32</i>
EXPERIMENTATION	33

<i>Introduction</i>	<i>33</i>
<i>Méthodologie.....</i>	<i>34</i>
<i>Prétraitement de la donnée cifar10.....</i>	<i>34</i>
<i>Transformation de la donnée en fonction du model</i>	<i>36</i>
NOS RESULTATS	37
INTRODUCTION	37
PERCEPTRON SIMPLE	37
<i>Postulat</i>	<i>37</i>
<i>Résultats.....</i>	<i>37</i>
<i>Conclusion</i>	<i>38</i>
PERCEPTRON MULTICOUCHES.....	39
<i>Postulat</i>	<i>39</i>
<i>Résultats.....</i>	<i>39</i>
<i>Conclusion</i>	<i>43</i>
RNN.....	44
<i>Postulat</i>	<i>44</i>
<i>Résultats.....</i>	<i>44</i>
<i>Conclusion</i>	<i>45</i>
CNN.....	46
<i>Postulat</i>	<i>46</i>
<i>Résultats.....</i>	<i>46</i>
<i>Conclusion</i>	<i>46</i>
CNN-LSTM	47
<i>Postulat</i>	<i>47</i>
<i>Résultats.....</i>	<i>47</i>
<i>Conclusion</i>	<i>47</i>
RESNETS	48
<i>Résultats.....</i>	<i>48</i>
CONCLUSION	50
CONCLUSION.....	51
<i>Lexique</i>	<i>52</i>

Le machine/deep Learning

Introduction

Avec l'avènement de l'intelligence artificielle, l'essor des objets connectés et l'augmentation des capacités technologiques, nombreuses recherches convergent à simuler le comportement humain afin de reproduire ses capacités cognitives dans les outils qui nous entourent.

Dans ce contexte le Deep Learning, littéralement "l'apprentissage profond", est devenu l'un des axes de recherche les plus explorés. Alors qu'est-ce que le Deep Learning, qu'est-ce qu'un réseau de neurones, quel lien avec notre cerveau humain, comment cela fonctionne et surtout pour quelles applications ?

C'est dans cette approche qu'a été fait ce projet de machine/deep learning afin de comprendre tout le paradigme derrière les réseaux de neurones.

De l'humain au deep learning

Pour comprendre le Deep Learning et surtout les réseaux de neurones, il ne suffit pas de s'intéresser aux mathématiques et à la technologie.

Comprendre le Deep Learning, c'est avant tout remonter à l'élément central de notre vision du monde : l'être humain.

L'Homme a en effet toujours cherché à reproduire sa façon d'être en refaisant une simulation de lui-même, de quelque façon que ce soit.

Simuler l'humain, c'est chercher à reproduire les différentes capacités qu'il utilise au quotidien : sociales, comportementales, éthiques, physiques... Dans le lot, celle qui renferme le plus de mystère et qui sans doute revêt le plus d'attrait pour la recherche appliquée est la capacité cognitive. C'est elle qui nous permet de connaître, mémoriser, raisonner, apprendre ou encore parler.

Afin de comprendre cette approche de l'humain, nous avons étudié l'an dernier différents algorithmes tels que la recherche locale naïve, les algorithmes génétiques et enfin l'approche au niveau des réseaux de neurones. Ce qui nous a amené au machine/deep learning.

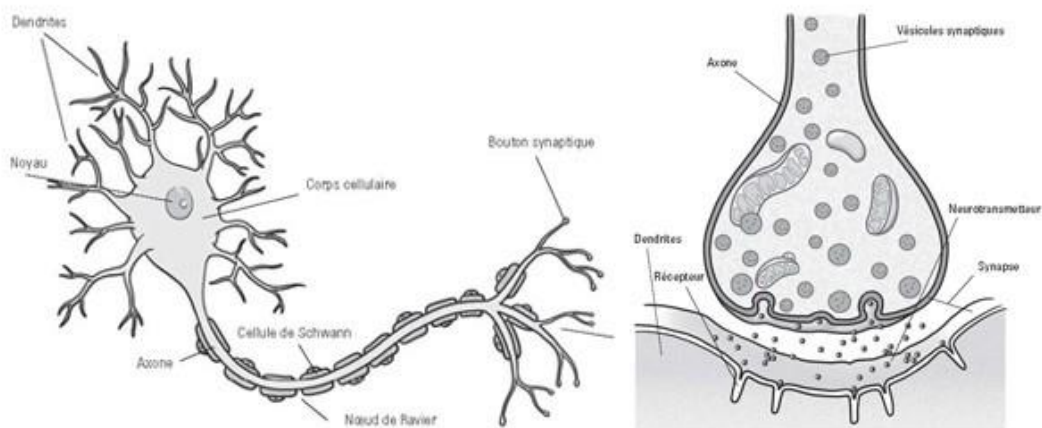
Son objectif est précisément de simuler le cerveau humain par des procédés informatiques. Imaginez un ordinateur qui puisse stocker autant d'informations que notre mémoire et qui sache décider et agir en même temps !

Plonger dans le Deep Learning, c'est donner corps à ces concepts de science-fiction à travers les réseaux de neurones, se rendre compte que les avancées sont tangibles et les résultats déjà impressionnants. Les recherches des géants du secteur comme Facebook, Google ou encore Apple dans ce domaine nous le prouvent chaque année. C'est par cette approche que nous allons donc parler des réseaux de neurones.

Introduction aux réseaux de neurones

Un neurone se compose d'un corps cellulaire, d'un axone qui représente le lien de transmission des signaux et d'une synapse qui permet le déclenchement d'un potentiel d'action dans le neurone pour activer une communication avec un autre neurone. Il faut savoir que la force d'un réseau de neurones réside dans la communication de ses neurones à travers des signaux électriques qu'on nomme "influx nerveux". Ces signaux se caractérisent par des fréquences qui jouent un rôle important au niveau de la propagation des signaux dans le réseau en question.

Ci-dessous une représentation d'un neurone issu du centre de recherche ICM



Les réseaux de neurones sont actuellement très utilisés dans de nombreux domaines industriels ou de la vie de tous les jours : reconnaissance d'images, d'écriture, de sons, analyse de cours boursiers, classifications. Ils font fonctionner des sondes spatiales ou des robots sur Mars.

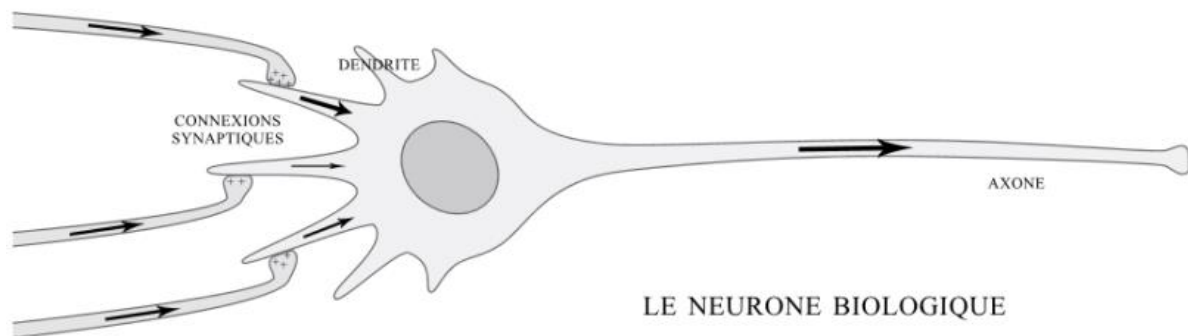
Un neurone biologique peut être représenté ainsi :

- La fonction d'un neurone est de transmettre l'influx nerveux, un signal électrique, mais il ne s'agit pas d'une simple liaison passive. Structuellement le neurone est relié, d'une part à un ensemble de dendrites et d'autre part à un axone.

Définition

Les dendrites constituent les entrées et servent à accumuler de la charge électrique au niveau du neurone jusqu'à atteindre un certain seuil. C'est alors que le neurone transmet un signal via l'axone qui agit comme sortie du neurone.

L'expérience montre de plus que les dendrites ne sont pas toutes équivalentes, certaines contribuent plus fortement que d'autres à atteindre le seuil, on associe donc les dendrites à un poids.



Les réseaux de neurone en informatique sont inspirés des neurones biologiques.

Principe

Chaque réseau de neurones doit apprendre en fonction de la donnée fournie en entrée et de la sortie souhaitée. Cette phase est nommée phase d'apprentissage.

Couplée à cette phase d'apprentissage, une phase de test existe et permet de tester le modèle sur un jeu de données similaire non connu du modèle.

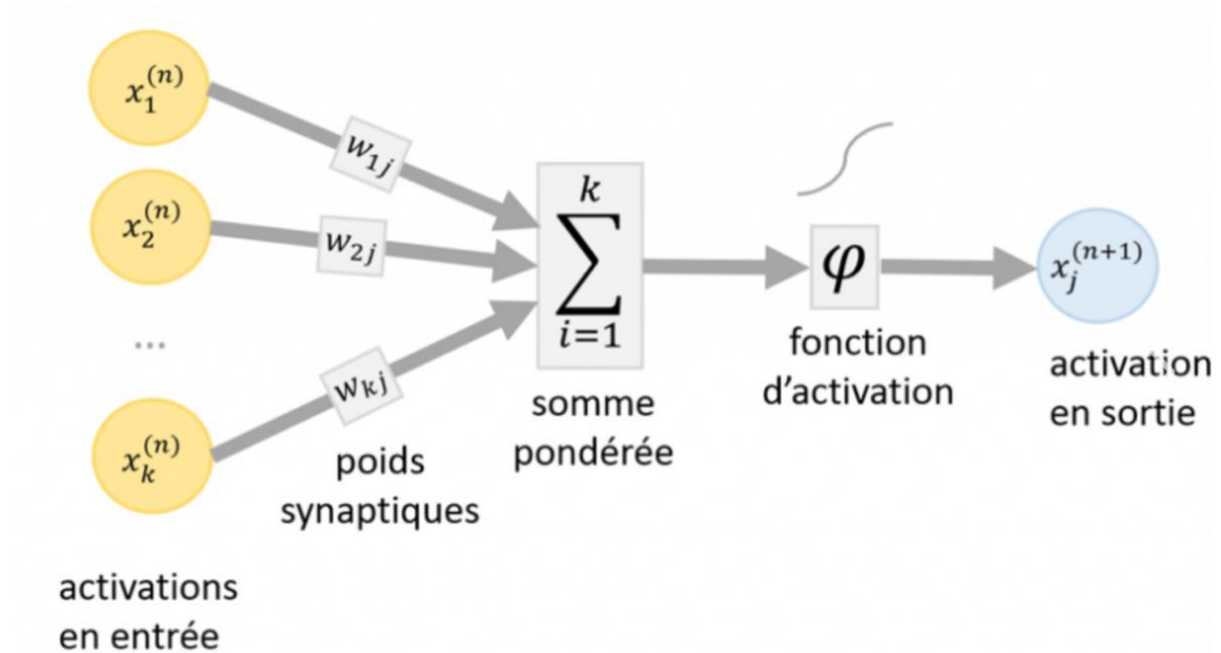
La première phase permet au modèle de s'autocorriger tandis que la seconde permet de s'assurer que le modèle n'est pas en sur-apprentissage.

Il existe plusieurs types de réseau de neurones inspiré des neurones biologiques qui seront présenté par la suite.

Les différents réseaux de neurones Classique

Le perceptron Simple (PS)

Définition



Le perceptron est un algorithme d'apprentissage supervisé de classification dit *classifieurs binaires* (c'est à dire séparant deux classes).

Il a été inventé en 1957 par Frank Rosenblatt au laboratoire d'aéronautique de l'université Cornell.

C'est un modèle inspiré des théories cognitives de Friedrich Hayek et de Donald Hebb. Il s'agit d'un neurone formel muni d'une règle d'apprentissage qui permet de déterminer automatiquement les poids synaptiques de manière à séparer un problème d'apprentissage supervisé.

Caractéristique

Un perceptron possède :

- n entrée(s) X .
- n poids W .
- Une fonction de Somme Σ .
- Une fonction d'activation j (ou fonction de transfert).
- Une sortie Y .

Fonctionnement mathématique

Chaque entrées X_i

- Chaque liaison entre une entrée X_i et la fonction somme possède un poids W_i
- La fonction somme est la suivante :
 $Z = f(W_1 * X_1 + W_2 * X_2 + \dots + W_i * X_i)$

Une fois cette somme calculée, on applique une fonction d'activation selon le type de procédé que l'on veut :

Classification :

- Si $z > 0 \Rightarrow Y = 1$
- Si $z < 0 \Rightarrow Y = 0$

Régression :

- Une fonction affine

Les fonctions de calcul de W

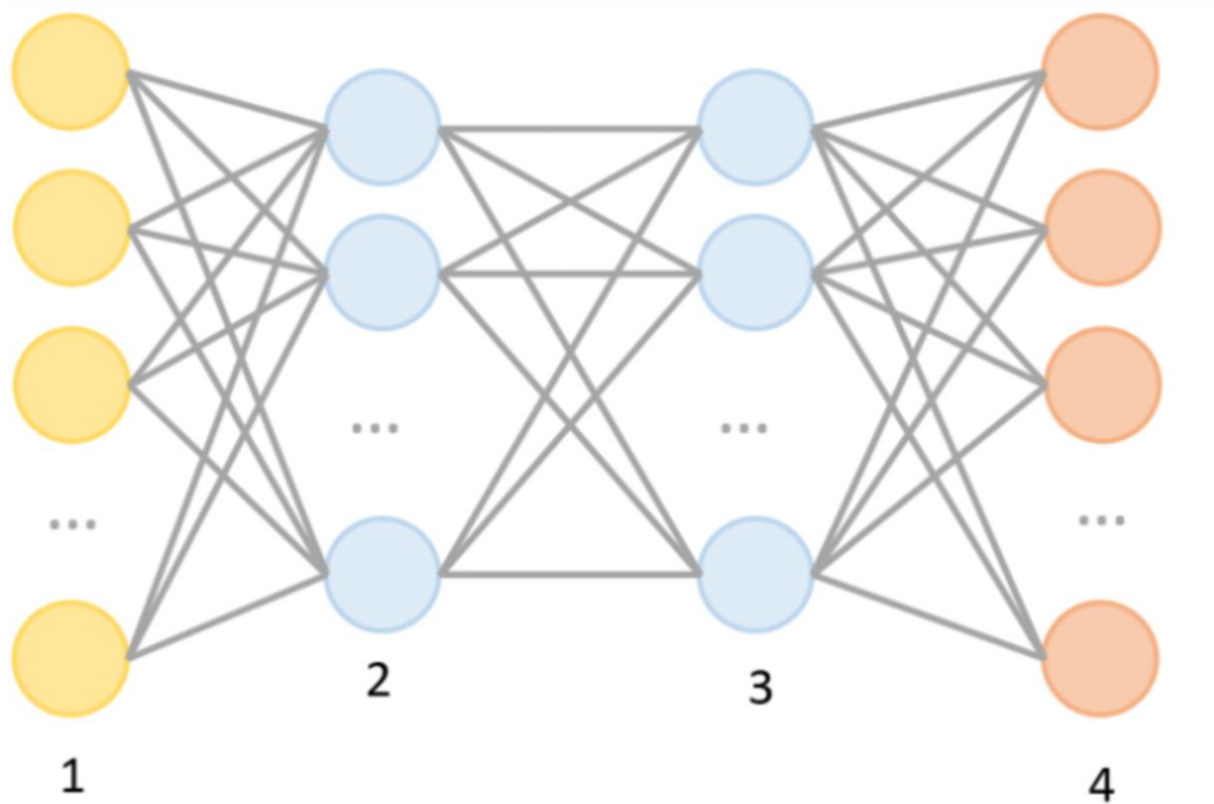
Classification : Perceptron Learning Algorithm	$W \leftarrow W + \alpha Y^k X^k$
Classification : Règle de Rosenblatt	$W \leftarrow W + \alpha (Y^k - g(X^k)) X^k$
Régression linéaire : Pseudo inverse	$W = ((X^T X)^{-1} X^T) Y$

On applique une fonction d'activation/transfert (voir partie sur les fonctions d'activation) sur la somme totale pour définir sa valeur.

$$S = b + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n = b + \sum_{i=1}^n w_i \cdot x_i$$

Le perceptron Multicouche (Multi Layer Network)

Définition



Le perceptron multicouche (multilayer perceptron MLP) est un type de réseau neuronal formel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie. Chaque couche est constituée d'un nombre variable de neurones, les neurones de la dernière couche (dite « de sortie ») étant les sorties du système global.

Caractéristique

Le Perceptron multicouche basé sur le modèle du Perceptron Simple a plusieurs couches de neurones liées entre elles. Chaque couche a un ou plusieurs neurones.

Le Perceptron simple ne pouvait classifier que des données séparées par un hyperplan. Or nous avons d'autres données plus complexes séparables par des hyper-surfaces.

Nous avons donc travaillé avec un perceptron Multicouche qui se compose de :

- Une couche d'entrée
- Une ou plusieurs couches cachées
- Une couche de sortie

Fonctionnement mathématique

Première couche (couche d'entrée) :

La ligne de données arrive en entrée du réseau, dans la première couche, dit couche d'entrée.

Tous les neurones de la couche d'entrée vont ensuite dans chaque entrée de la première couche cachée.

1ère Couche cachée :

Il est possible de calculer la sortie de tous les neurones de la première couche cachée en appliquant la formule de la somme avec le biais :

$$S = b + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n = b + \sum_{i=1}^n w_i \cdot x_i$$

Puis en lui appliquant une fonction d'activation, dans cet exemple on a utilisé la fonction tanh :

$$x_j^l = \theta(s_j^l) = \text{Tanh}\left(\sum_{i=0}^{d^{l-1}} w_{ij}^l x_i^{l-1}\right)$$

Couche caché et couche de sortie :

Avec les sorties de la première couche cachée, vous pouvez ensuite calculer les sorties de la 2ème couche cachée, puis la 3ème couche cachée, et ainsi de suite jusqu'à la sortie. Votre information s'est donc propagée dans l'ensemble du réseau.

Recalcul de delta

Recalcule un Delta en fonction de l'erreur, ce delta permet par la suite de recalculer les poids

Pour la classification :

$$\delta_j^L = (1 - (x_j^L)^2) \times (x_j^L - y_j)$$

$$\delta_i^{l-1} = (1 - (x_i^{l-1})^2) \times \sum_{j=1}^{d^l} (w_{ij}^l \times \delta_j^l)$$

Pour la régression :

$$\delta_j^L = (x_j^L - y_j)$$

$$\delta_i^{l-1} = (1 - (x_i^{l-1})^2) \times \sum_{j=1}^{d^l} (w_{ij}^l \times \delta_j^l)$$

$$w_{ij}^l \leftarrow w_{ij}^l - \alpha x_i^{l-1} \delta_j^l$$

Les fonctions d'activation (ou fonction de transfert).

Il existe plusieurs fonctions d'activation dans un réseau, les fonctions d'activations sont énoncées plus tard dans le rapport pour plus de précision.

Calcul des poids W

Il faut mettre à jour les poids afin d'adapter le modèle aux données.

$$w_{ij}^l \leftarrow w_{ij}^l - \alpha x_i^{l-1} \delta_j^l$$

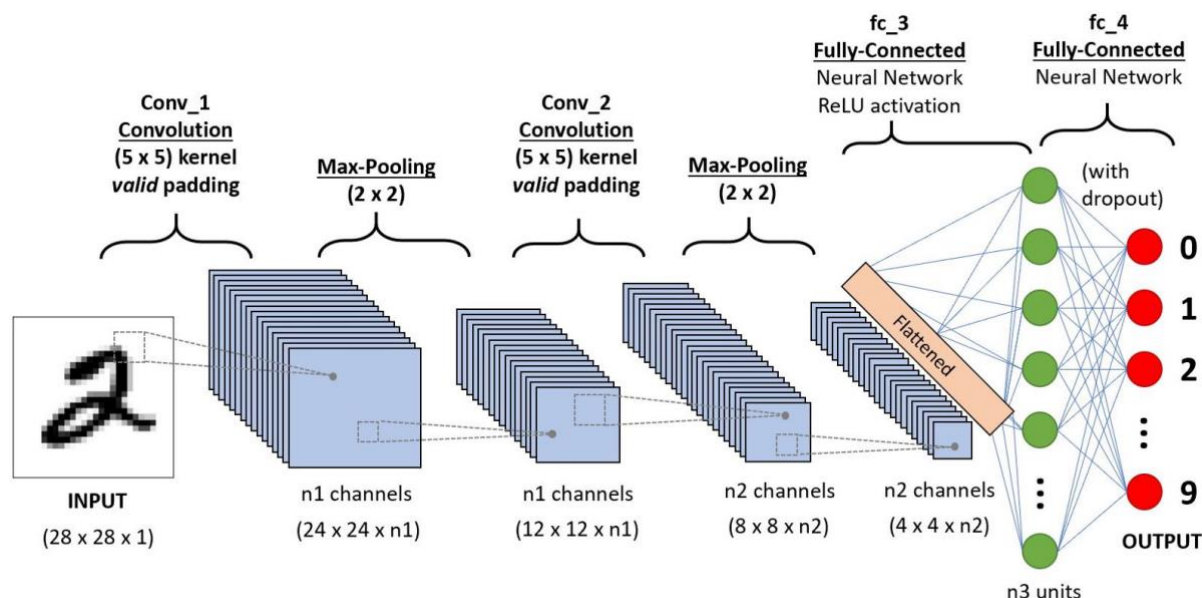
Durant la phase de retro-propagation, on part de la fin du réseau pour remonter vers les couches en amont du réseau en corrigeant les états et en mettant à jour les poids W de chaque réseau.

La descente de gradient permet d'évaluer l'impact d'un poids sur l'erreur et donc de s'améliorer.

Lorsque l'on regarde l'erreur commise par notre perceptron sur une propagation, on peut évaluer l'impact qu'a eu un poids en particulier sur cette erreur. Grace à cette information, on peut également évaluer si augmenter ou diminuer ce poids va améliorer ou détériorer notre erreur.

Réseaux de neurones convolutifs (CNN)

Définition



Un réseau de neurones convolutifs (ConvNet / CNN) est un algorithme d'apprentissage en profondeur qui peut intégrer une image d'entrée, attribuer une importance (poids et biais) à divers aspects / objets de l'image et être capable de les différencier les uns des autres.

Dans les méthodes de base, les filtres du CNN sont conçus à la main ; toutefois avec un entraînement suffisant, le ConvNet est capable d'apprendre ces filtres/caractéristiques.

L'architecture d'un ConvNet est analogue à celle du modèle de connectivité des neurones dans le cerveau humain et s'inspire de l'organisation du cortex visuel.

Les neurones individuels ne répondent aux stimuli que dans une région restreinte du champ visuel appelée champ récepteur. Une collection de ces champs se chevauchent pour couvrir toute la zone visuelle.

Un ConvNet est capable de capturer avec succès les dépendances spatiales et temporelles dans une image grâce à l'application de filtres appropriés. L'architecture s'ajuste mieux au jeu de données d'image en raison de la réduction du nombre de paramètres impliqués et de la réutilisabilité des poids. En d'autres termes, le réseau peut être formé pour mieux comprendre la sophistication de l'image.

Le rôle du ConvNet est de réduire les images à une forme plus facile à traiter, sans perdre les caractéristiques essentielles à une bonne prédiction. Ceci est important lorsque nous devons concevoir une architecture qui soit, non seulement bonne pour l'apprentissage des fonctionnalités, mais également évolutive pour des jeux de données volumineux.

Fonctionnement

Le ConvNet décrit quatre opérations principales dans l'image présente en définition ci-dessus :

- Convolution
- Non linéarité (ReLU)
- Regroupement ou sous-échantillonnage
- Classification (couche entièrement connectée)

Ces opérations sont les éléments de base de *chaque* réseau de neurones convolutionnels. Comprendre leur fonctionnement est donc une étape importante pour développer une compréhension solide des ConvNets. Nous essaierons de comprendre l'intuition de chacune de ces opérations ci-dessous.

Réseaux de neurones récurrents (RNN)

Définition

Les RNN sont utilisés dans le deep-learning et dans le développement de modèles simulant l'activité des neurones dans le cerveau humain. Ils sont particulièrement puissants dans les cas d'utilisation dans lesquels le contexte est essentiel pour prédire un résultat et se distinguent des autres types de réseaux neuronaux artificiels car ils utilisent des boucles de rétroaction pour traiter une séquence de données qui informe le résultat final, qui peut également être une séquence de données. Ces boucles de rétroaction permettent aux informations de persister. L'effet est souvent décrit comme une mémorisation.

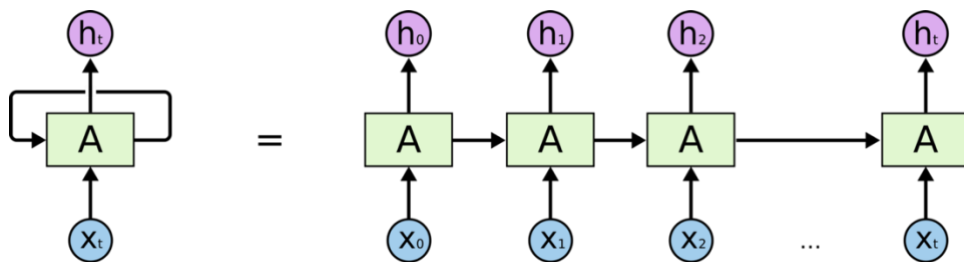
Les cas d'utilisation de RNN ont tendance à être liés à des modèles de NLP (Natural Language Processing) dans lesquels la connaissance de la lettre suivante d'un mot ou du mot suivant d'une phrase dépend des données qui le précèdent.

Comment les réseaux de neurones récurrents apprennent

Les réseaux de neurones artificiels sont créés avec des composants de traitement de données interconnectés, librement conçus de manière à fonctionner tel un cerveau humain.

Ils sont composés de couches de neurones artificiels (nœuds de réseau) capables de traiter les entrées et de les transmettre aux autres nœuds du réseau.

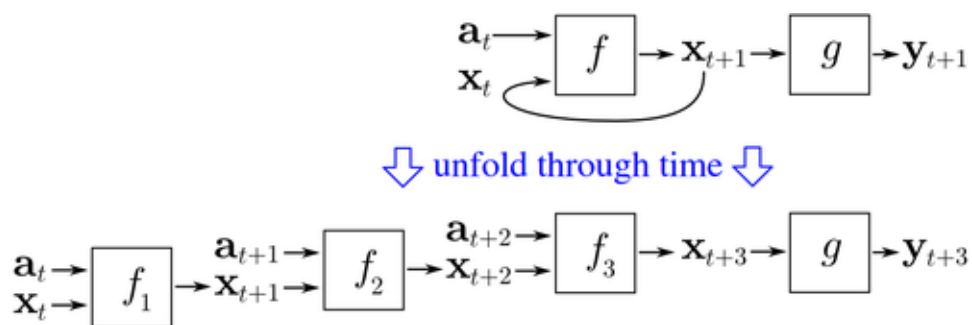
Les nœuds sont connectés par des arêtes ou des poids qui influencent la force du signal et la sortie finale du réseau.



Dans certains cas, les réseaux de neurones artificiels traitent les données dans une seule direction, d'entrée en sortie. Ces réseaux de neurones "Feedforward" incluent des réseaux de neurones à convolution qui sont efficaces dans les systèmes de reconnaissance d'images. Les RNN, cependant, peuvent traiter les données dans les deux directions.

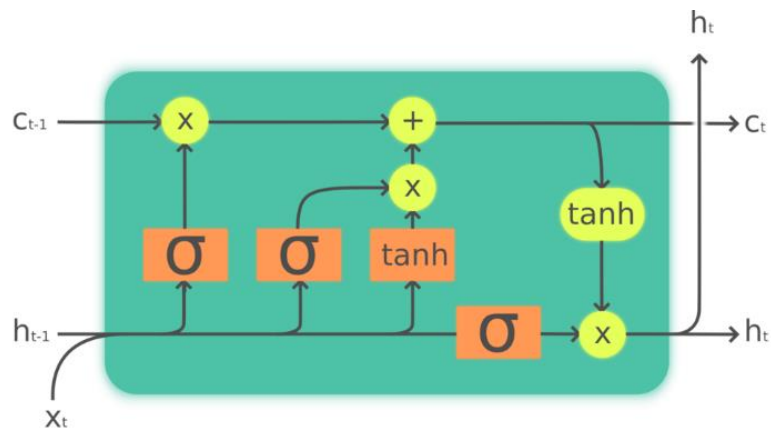
En effet, tout comme les réseaux neuronaux "Feedforward", les RNN peuvent traiter les données de l'entrée initiale à la sortie finale. Mais contrairement aux réseaux neuronaux "Feedforward", les RNN utilisent des boucles de rétroaction telles que la Backpropagation Through Time (BPTT) tout au long du processus de calcul pour relier les données au réseau.

BPTT commence par déployer un réseau de neurones récurrent dans le temps. Le réseau déplié contient k entrées et sorties, mais chaque copie du réseau partage les mêmes paramètres. L'algorithme de rétropropagation est utilisé par la suite pour trouver le gradient de meilleur coût par rapport à tous les paramètres du réseau. Ce qui rend la formation du RNN nettement plus rapide et permet aux RNN de traiter des données séquentielles et temporelles.



Unités de mémoire à court terme

Un problème pour les RNN standard est le problème du vanishing gradient. Une solution à ce problème est d'utiliser un ensemble appelé unités de mémoire à court terme (LSTM) inventés.



Legend:

Layer



Pointwise op



Copy



Les RNN sont composées d'unités de LSTM. Chaque unité LSTM est composée d'une cellule, d'une porte d'entrée, d'une porte de sortie et d'une porte d'oubli. La cellule se souvient des valeurs sur des intervalles de temps arbitraires et les trois portes régulent le flux d'informations entrant et sortant de la cellule.

Cela permet aux RNN de déterminer quelles données sont importantes et doivent être mémorisées et reliées au réseau, et quelles données peuvent être oubliées.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t \circ \sigma_h(c_t)$$

Réseau hybride (mélange de CNN & RNN)

Définition

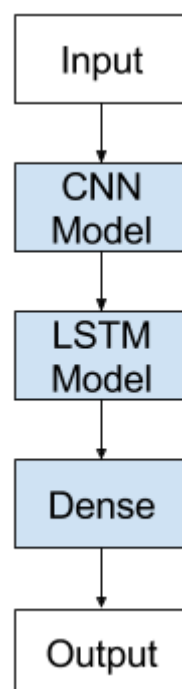
Les CNN sont utilisés dans la modélisation des problèmes liés aux entrées spatiales telles que les images. Les CNN ont fait leurs preuves dans les tâches liées aux images, telles que la vision par ordinateur, la classification des images, la détection d'objets, etc.

Les LSTM sont utilisés dans les tâches de modélisation liées aux séquences et font des prédictions basées sur celles-ci. Les LSTM sont largement utilisés dans les tâches liées à la NL telles que la traduction automatique, la classification de phrases, la génération.

Le réseau de mémoire à court terme à long CNN (CNN-LSTM) est une architecture LSTM spécialement conçue pour les problèmes de prédiction de séquence avec des entrées spatiales, telles que des images ou des vidéos.

Architecture et fonctionnement

L'architecture CNN-LSTM implique l'utilisation de deux sous-modèles : le modèle CNN pour l'extraction de caractéristiques et le modèle LSTM pour l'interprétation des caractéristiques sur l'ordonnancement.



Naturellement, le fonctionnement mathématique de ce modèle n'est autre que celui du CNN dans un premier temps, puis celui du RNN dans un second temps.

Les CNN-LSTM sont généralement utilisés lorsque leurs entrées ont une structure spatiale en entrée, telle que la structure **2D** ou les pixels d'une image ou la structure **1D** de mots dans une phrase, un paragraphe ou un document, et possèdent également une structure temporelle en entrée, telle que l'ordre des images dans une vidéo ou des mots dans le texte, ou nécessitent la génération d'une sortie avec une structure temporelle telle que des mots dans une description textuelle.

Ils sont largement utilisés dans des tâches similaires à la reconnaissance d'activité, à la description d'image, à la description vidéo, etc.

Architectures récentes d'apprentissage en profondeur

Les réseaux résiduels profonds sont apparus comme une famille d'architectures extrêmement profondes présentant une précision accrue et de bons comportements de convergence.

Chacun permet de former avec succès des réseaux profonds en surmontant les limites de la conception de réseau traditionnelle.

La première intuition lors de la conception d'un réseau profond peut consister simplement à empiler bon nombre des blocs de construction typiques tels que des couches convolutives ou entièrement connectées.

Cela fonctionne, mais les performances diminuent rapidement à mesure que le réseau devient profond. Le problème provient de la rétropropagation dans le modèle. Lors de la formation d'un réseau, un signal de gradient doit être propagé vers l'arrière sur le réseau (rétropropagation), de la couche supérieure à la couche inférieure, afin de garantir que le réseau se met à jour correctement.

Avec un réseau traditionnel, ce gradient diminue au fur et à mesure qu'il traverse chaque couche du réseau.

Réseau résiduel (ResNet)

Définition

Un réseau résiduel, ou ResNet, est une architecture de réseau de neurones qui résout le problème de la disparition de gradient (vanishing gradient).

La difficulté de renvoi du signal en arrière est résolue par la mise en place d'un raccourci afin d'aller d'une couche à une autre sans être la suivante et permet ainsi....

Fonctionnement mathématique

Dans un réseau traditionnel, l'activation au niveau d'une couche est définie comme suit :

$$y = f(x)$$

Où $f(x)$ est notre convolution, notre multiplication matricielle, notre normalisation par lots, etc. Lorsque le signal est renvoyé, le gradient doit toujours passer par $f(x)$, ce qui peut poser problème en raison des non-linéarités impliquées.

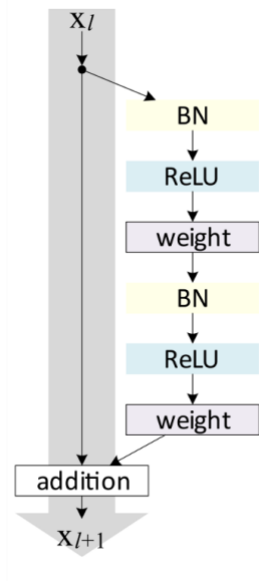
Au lieu de cela, ResNet implémente à chaque couche :

$$y = f(x) + x$$

Le « $+ x$ » à la fin est le raccourci.

Cela permet au dégradé de revenir directement en arrière. En empilant ces couches, le gradient pourrait théoriquement « sauter » sur toutes les couches intermédiaires et atteindre la fin sans être diminué.

Bien que ce soit intuitif, la mise en œuvre réelle est un peu plus complexe. Dans la dernière version de ResNets, $f(x) + x$ prend la forme :



Comparaison des différents model selon leurs cas d'usage

MLP

Utilisation des MLP :

- Jeux de données tabulaires
- Problèmes de prédiction de classification
- Problèmes de prédiction de régression

CNN

Les réseaux de neurones convolutifs, ou CNN, ont été conçus pour mapper des données d'image à une variable de sortie.

Ils se sont avérés si efficaces qu'ils sont la méthode de choix pour tout type de problème de prédiction impliquant des données d'image en entrée. L'avantage d'utiliser les CNN est leur capacité à développer une représentation interne d'une image bidimensionnelle comme expliqué dans la partie explicative des CNN. Cela permet au modèle d'apprendre la position et la mise à l'échelle de variantes de structure dans les données, ce qui est important lorsque vous travaillez avec des images.

Utilisation des CNN :

- Données d'image
- Problèmes de prédiction de classification
- Problèmes de prédiction de régression

RNN

Les RNN en général et les LSTM en particulier ont eu le plus de succès lorsqu'on travaille avec des séquences de mots et des paragraphes, généralement appelés traitements de langage naturel.

Cela inclut à la fois des séquences de texte et des séquences de langage parlé représentées sous forme de série chronologique. Ils sont également utilisés comme modèles génératifs nécessitant une sortie de séquence, non seulement avec du texte, mais également dans des applications telles que la génération d'écriture manuscrite.

Utilisation des RNN :

- Données texte
- Données de la parole
- Problèmes de prédiction de classification
- Problèmes de prédiction de régression
- Modèles génératifs

Afin de valider si ses théories étaient exactes nous avons fait plusieurs tests sur le dataset cifar10 sur les différents modèles afin d'analyser le comportement de chaque modèle en fonction des hyper-paramètres qui lui sont fournis.

Les différentes fonctions utilisées

Introduction

Suite aux présentations des différents modèles utilisés, nous allons expliquer brièvement dans cette partie les différentes fonctions que l'on peut utiliser dans les modèles de machine/deep learning.

Optimiseur

Algorithme qui permet de minimiser la fonction de perte pondérée.

Descente de gradient stochastique (SGD) :

Fonction sur keras :

SGD (lr=int, momentum=int, decay=int, nesterov=False)

- **Learning Rate (Lr)** : Le taux d'apprentissage est un hyper-paramètre qui contrôle à quel point nous ajustons les poids de notre réseau en fonction du gradient de perte. Plus la valeur est basse, plus nous roulons lentement sur la pente descendante. Bien que cela puisse être une bonne idée (en utilisant un taux d'apprentissage faible) pour nous assurer que nous ne manquons aucun minimum local, cela pourrait également signifier que nous allons prendre beaucoup de temps pour converger - surtout si nous restons coincés dans une région de plateau.

Momentum : Momentum prend en compte les gradients passés pour aplanir les étapes de la descente. Il peut être appliqué avec une descente par gradient en batch, une descente en gradient par mini-batch ou une descente par gradient stochastique.

Batch_size : indique la taille du sous-ensemble de votre échantillon d'apprentissage (par exemple, 100 sur 1 000) qui sera utilisée pour former le réseau au cours de son processus d'apprentissage. Chaque lot entraîne le réseau dans un ordre successif, en tenant compte des poids mis à jour provenant des couches du lot précédent.

Return_sequence : indique si une couche récurrente du réseau doit renvoyer la totalité de sa séquence de sortie (c'est-à-dire une séquence de vecteurs de dimension spécifique) à la couche suivante du réseau, ou tout simplement sa dernière sortie, qui est un vecteur unique de la même dimension. Cette valeur peut être utile pour les réseaux conformes à une architecture RNN.

Batch_input_shape : définit que la classification séquentielle du réseau de neurones peut accepter des données d'entrée de la taille de lot définie uniquement, limitant ainsi la création de tout vecteur de dimension variable. Il est largement utilisé dans les réseaux LSTM empilés.

Dropout

Dropout prend la sortie des activations de la couche précédente et définit de manière aléatoire une certaine fraction (taux d'abandon) des activations sur 0, en les annulant ou en les « supprimant ».

C'est une technique de régularisation courante utilisée pour prévenir les surajustements dans les réseaux de neurones. Le taux d'abandon est l'hyper paramètre ajustable qui est ajusté pour mesurer les performances avec différentes valeurs. Il est généralement défini entre 0,2 et 0,5 (mais peut être défini de manière arbitraire).

Le décrochage n'est utilisé que pendant la formation ; Au moment du test, aucune activation n'est abandonnée, mais réduite par un facteur de taux d'abandon. Cela tient compte du nombre d'unités actives pendant la période de test par rapport à la durée de formation.





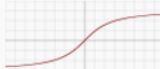




$$\text{new_weight} = \text{existing_weight} - \text{learning_rate} * \text{gradient}$$

Fonctions de perte dans les réseaux de neurones

Les fonctions de pertes utilisées dans Keras :

- Mse: mean_squared_error
- Mae: mean_absolute_error
- Mean_absolute_percentage_error
- Mean_squared_logarithmic_error
- Squared_hinge
- Hinge
- Categorical_hinge
- Logcosh
- Categorical_crossentropy
- Sparse_categorical_crossentropy
- Binary_crossentropy
- Kullback_leibler_divergence
- Poisson
- Cosine_proximity

Activation

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Les métriques

Métriques pour évaluer les algorithmes d'apprentissage automatique en Python

Les mesures que vous choisissez pour évaluer vos algorithmes d'apprentissage machine sont très importantes.

Le choix des métriques influence la manière dont la performance des algorithmes d'apprentissage automatique est mesurée et comparée. Ils influencent la manière dont vous pondérez l'importance des différentes caractéristiques dans les résultats et votre choix ultime de l'algorithme à choisir.

Quelle est la différence entre `categorical_accuracy` et `sparse_categorical_accuracy` in Keras ?

Pour l'exemple suivant :

```
def categorical_accuracy(y_true, y_pred):  
    return K.cast(K.equal(K.argmax(y_true, axis=-1),  
                          K.argmax(y_pred, axis=-1)),  
                  K.floatx())  
  
def sparse_categorical_accuracy(y_true, y_pred):  
    return K.cast(K.equal(K.max(y_true, axis=-1),  
                          K.cast(K.argmax(y_pred, axis=-1), K.floatx()))),  
                  K.floatx())
```

Categorical_accuracy

Vérifie si l'index de la valeur vraie maximale est égal à l'index de la valeur prédite maximale.

Sparse_categorical_accuracy

Vérifie si la valeur vraie maximale est égale à l'index de la valeur prédite maximale.

Confrontation

Donc, `categorical_accuracy` vous devez spécifier votre target (y) en tant que vecteur codé à un seul état (par exemple, dans le cas de 3 classes, il devrait être (0, 1, 0)). `sparse_categorical_accuracy`.

Précision binaire:

```
def binary_accuracy ( y_true , y_pred ):  
    retourner K . moyenne ( K . égale ( y_true , K . rond ( y_pred )), axe = - 1 )  
K.round (y_pred) implique que le seuil est 0.5, tout ce qui est supérieur à 0.5 sera considéré comme correct.
```

Exactitude catégorique :

```
def categorical_accuracy ( y_true , y_pred ):  
    retourner K . coulée ( K . égale ( K . argmax ( y_true , axe = - 1 ),  
K . argmax ( y_pred , axis = - 1 )),  
K . floatx ())  
K.argmax (y_true)  
prend la valeur la plus élevée pour être la prédiction et correspond au jeu de comparaison.
```

Précision catégorique clairsemée :

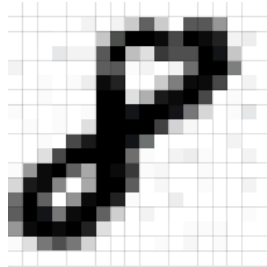
```
def sparse_categorical_accuracy ( y_true , y_pred ):  
    retourner K . coulée ( K . égale ( K . max ( y_true , axe = - 1 ),  
K . coulée ( K . argmax ( y_pred , axe = - 1 ), K . floatx ())),  
K . floatx ())  
Peut-être une meilleure métrique que categorical_accuracy dans certains cas, en fonction de vos données.
```

Précision catégorique maximale :

```
def top_k_categorical_accuracy ( y_true , y_pred , k = 5 ):  
    retourner K . moyenne ( K . in_top_k ( y_pred , K . argmax ( y_true , axe = - 1 ), k ), l'axe  
= - 1 )  
Le top-k est mesuré sur la précision de la prédiction correcte dans les prédictions du top-k. La plupart des articles exposeront l'efficacité des modèles en fonction de la précision du top 5.
```

Expérimentation

Introduction



Une image d'un appareil photo numérique standard aura trois canaux - rouge, vert et bleu - vous pouvez les imaginer sous la forme de trois matrices 2D superposées (une pour chaque couleur), chacune ayant des valeurs de pixels comprises entre 0 et 255.

Canal est un terme conventionnel utilisé pour désigner une certaine composante de l'image.

Pour l'analyse de nos modèles, nous nous sommes penchés sur le dataset Cifar 10.

Un modèle présente de nombreux paramètres différents :

- Le batch_size qui correspond à la taille des « morceaux » de jeu de données passés au modèle lors de l'apprentissage
- Le nombre d'epochs qui correspond au nombre d'itérations qu'effectue le modèle avant de s'arrêter
- Le nombre de couches cachées, ainsi que le nombre de couches spécifiques selon le modèle voulus
- Les fonctions d'activations utilisées pour les couches cachées mais aussi pour la couche de sortie du modèle
- Les dimensions des couches cachées, on parlera aussi de nombre de neurones
- L'optimizer utilisé qui influe sur la manière d'apprendre du modèle
- Les metrics propres à l'accuracy et à la loss
- Le learning rate qui influence la vitesse à laquelle les poids des neurones sont mis à jour.

Méthodologie

La méthode de test se base sur la mise en place d'une table de paramètres dans laquelle, le choix du paramètre pour une fonction est sélectionné de manière aléatoire.

Ci-dessous notre table :

```
lr = random() * (0.1 - 0.0001) + 0.0001
momentum = random() * (0.1 - 0.0001) + 0.0001
decay = lr / epochs

Param = {'input_shape': 3072,
        'input_shape_rnn': (32, 96),
        'input_shape_cnn': (32, 32, 3),
        'lr': lr,
        'hidden_dim': 128,
        'units': 512,
        'unitsSlp': 10,
        'last_units': 10,
        'first_neuron': [4, 8, 16, 32, 64],
        'hidden_layers': [2, 4, 6, 8, 9, 10, 20, 25, 30],
        'kernel_constraint': maxnorm(3),
        'batch_size': (64, 128, 512, 1024, 2048),
        'epochs': epochs,
        'dropout': (0, 0.5, 1),
        'padding': 'same',
        'metrics': ['accuracy'],
        'weight_regularizer': [None],
        'emb_output_dims': [None],
        'shape': ['brick', 'long_funnel'],
        'optimizer': ['adam', 'Nadam', 'RMSprop', SGD(lr=lr, momentum=momentum, decay=decay, nesterov=False)],
        'losses': [mse, logcosh, binary_crossentropy, categorical_crossentropy],
        'activation': [relu, elu, linear],
        'last_activation': [softmax, sigmoid],
        'nb_classes': 10}
```

Prétraitement de la donnée cifar10

Pour le prétraitement des données du dataset cifar10, a été mis en place un prétraitement qui est appliqué sur l'ensemble des modèles, exceptés si dans le modèle une fonction spécifique de prétraitement est présente.

L'original des données de traitement par lots est une matrice de taille 10 000 x 3072.

Le nombre de colonnes, (10000), indique le nombre de données d'échantillon.

Comme indiqué dans l'ensemble de données CIFAR-10 / CIFAR-100, le vecteur de lignes (3072) représente une image couleur de 32 x 32 pixels.

Étant donné que notre projet utilise des modèles différents, pour les tâches de classification, le vecteur de ligne d'origine n'est pas approprié selon le modèle utilisé.

Dans le code du projet (cf aux différentes fonctions de pre-processing dans chaque *fichier.py* des modèles) nous avons utilisé une fonction nommée *preprocess_cifar10*, elle représente la fonction mère de tous les modèle dans la classe *ModelManager.py*.

Cette fonction ensuite est surchargé (overridden) dans chaque classe de modèle.

Nous n'allons pas expliquer tous les prétraitements existants pour le dataset cifar10 mais uniquement celle de *ModelManager.py*

Transformation de la donnée en fonction du model

Pour insérer des données d'image dans un modèle CNN, la dimension du tenseur en entrée doit être soit (largeur, hauteur, num_channel) ou (num_channel, largeur, hauteur). Cela dépend de votre choix (consultez le tensorflow conv2d).

Le vecteur ligne d'une image a exactement le même nombre d'éléments si vous calculez $32 * 32 * 3 == 3072$.

Pour reformater le vecteur ligne sous la forme (largeur x hauteur x num_channel), deux étapes sont nécessaires. La première étape consiste à utiliser la fonction **reshape**, et la deuxième étape consiste à utiliser la fonction **transpose** dans numpy.

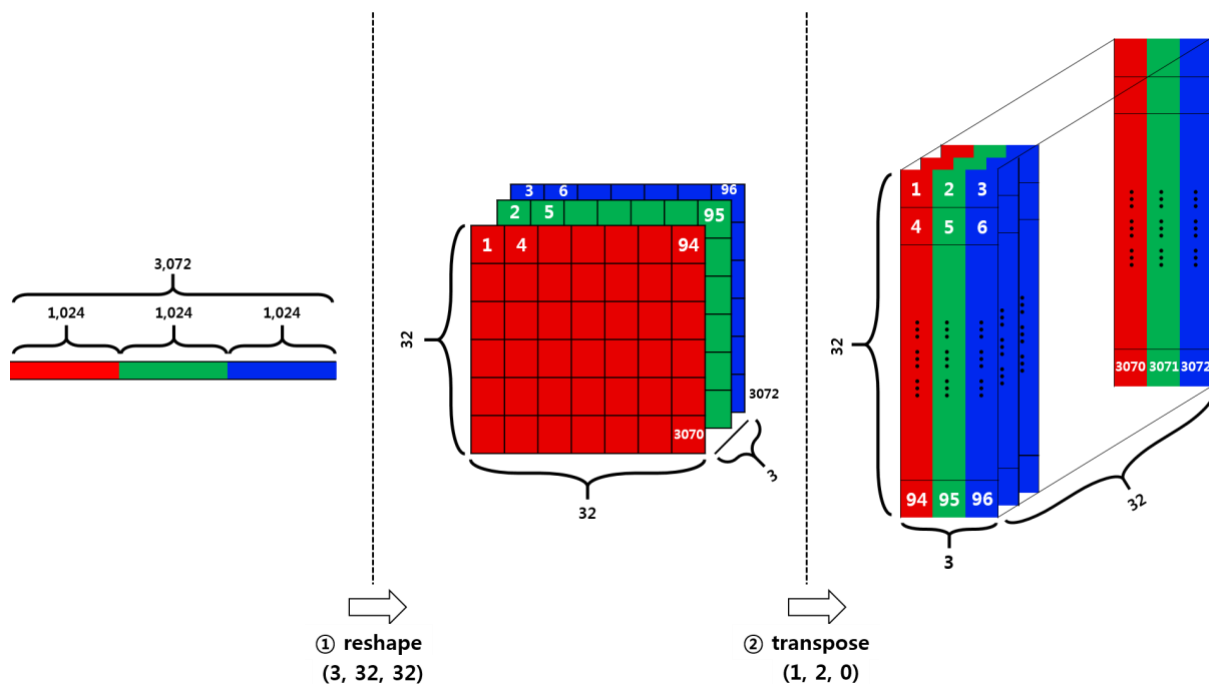
Par définition la fonction *reshapetransforme* un tableau en une nouvelle forme sans modifier ses données. Ici, la phrase sans changer ses données est une partie importante puisque vous ne voulez pas blesser les données. Reshape les opérations doivent être livrées en trois étapes plus détaillées. La direction suivante est décrite dans un concept logique.

Divisez le vecteur de rangée en 3 morceaux, chaque morceau désignant chaque canal de couleur.

- la matrice résultante a (3 x 1024) matrice, ce qui donne un total de (10 000 x 3 x 1024) tenseur.

Divisez les 3 pièces par 32. 32 est la largeur et la hauteur d'une image.

- cela donne (3 x 32 x 32), ce qui donne un tenseur (10000 x 3 x 32 x 32) au total



Nos résultats

Introduction

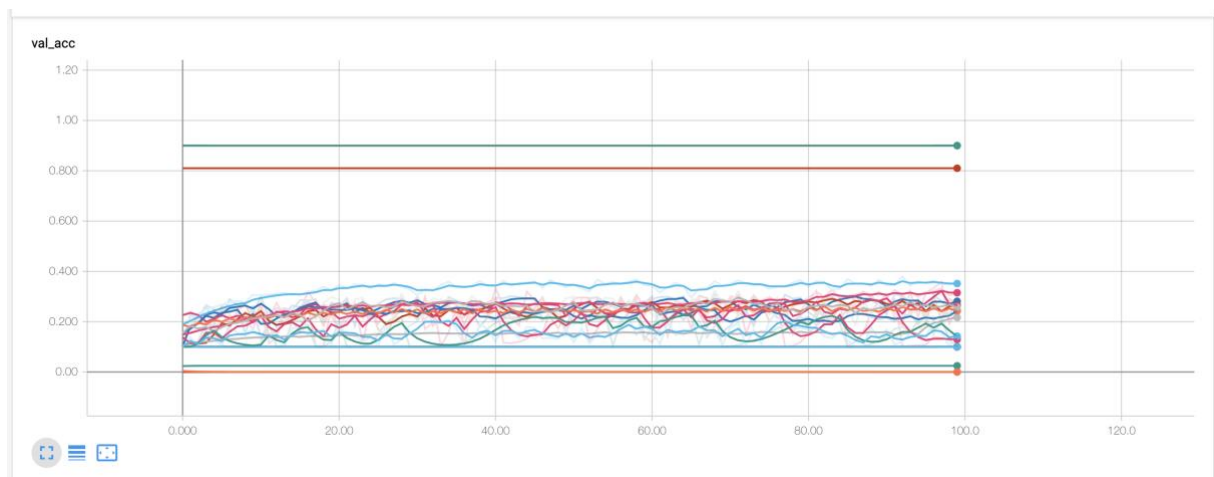
Suite au lancement de nos modèles, nous avons essayé d'interpréter nos résultats en fonction des courbes obtenues dans Tensorboard.

Perceptron Simple

Postulat

Le perceptron simple n'ayant jamais eu pour vocation de classer de telles sources (images), il n'en est pas attendu un résultat probant. Toutefois, il reste intéressant de voir les résultats obtenus.

Résultats



Si on se réfère uniquement à ses résultats, on peut constater que des tests donnent de très bons résultats.

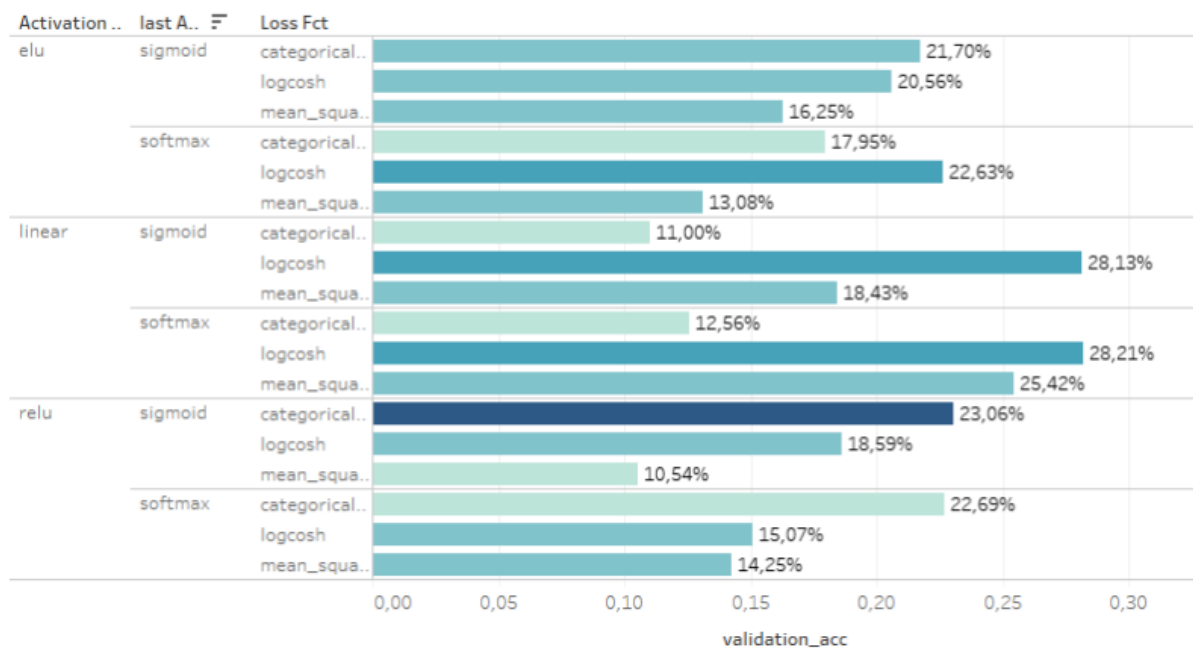
Ce qui est surprenant sachant que le perceptron simple ne peut remplir les conditions nécessaires à classer en 10 étiquettes.

Après vérification, le point commun à ces tests provient du choix de la fonction loss. Ici, la fonction `binary_crossentropy` est utilisée et du fait que celle-ci ne peut être utilisée que sur 2 étiquettes au maximum, les résultats obtenus sont donc erronés.

Sans ces courbes, on constate que le perceptron simple offre une classification fiable à 0,37.

Ce qui pourrait être validé par les schémas suivants :

Hyperparameters influence on Validation Accuracy



On peut voir que pour un modèle ayant comme fonction d'activation « Linear » avec comme fonction d'activation sur sa dernière couche « softmax » ou « sigmoid » et comme fonction de perte « logcosh », on a un modèle qui pourrait être performant, cependant cela pourrait se traduire par un sur-apprentissage.

Conclusion

Le perceptron simple est un modèle qui n'est pas adapté pour le dataset cifar10. Le modèle tombera facilement dans le sur-apprentissage et le sous-apprentissage du au nombre de neurone qu'elle a pour calculer le W optimal.

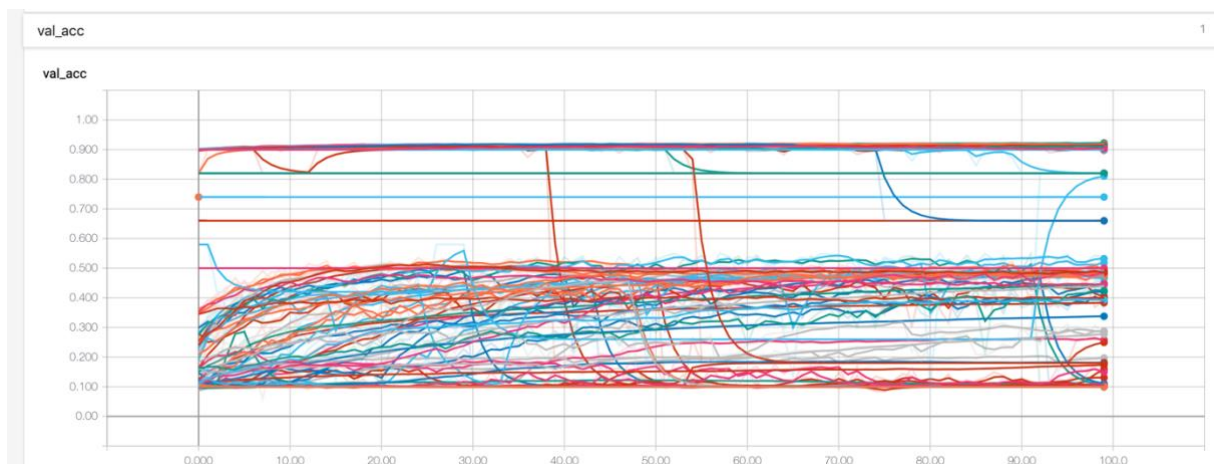
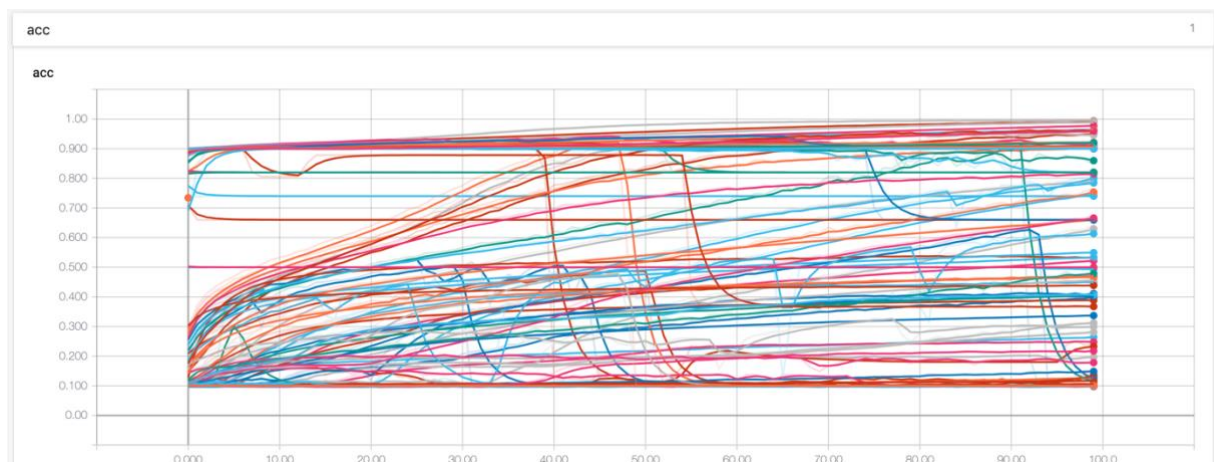
Perceptron Multicouches

Postulat

Tout comme le perceptron simple, nous partons du principe que celui-ci pourrait donner de bons résultats mais il faudra peut-être un nombre de couche et/ou d'épochs très importants.

Le nombre de couche cachées maximum est de 30

Résultats



Nombre de couches cachées : 6

Epochs : 100

Batch size : 1024

Fonction d'activation : Relu

Fonction de perte : Logcosh

Optimizer : RMSprop



Nombre de couches cachées : 4

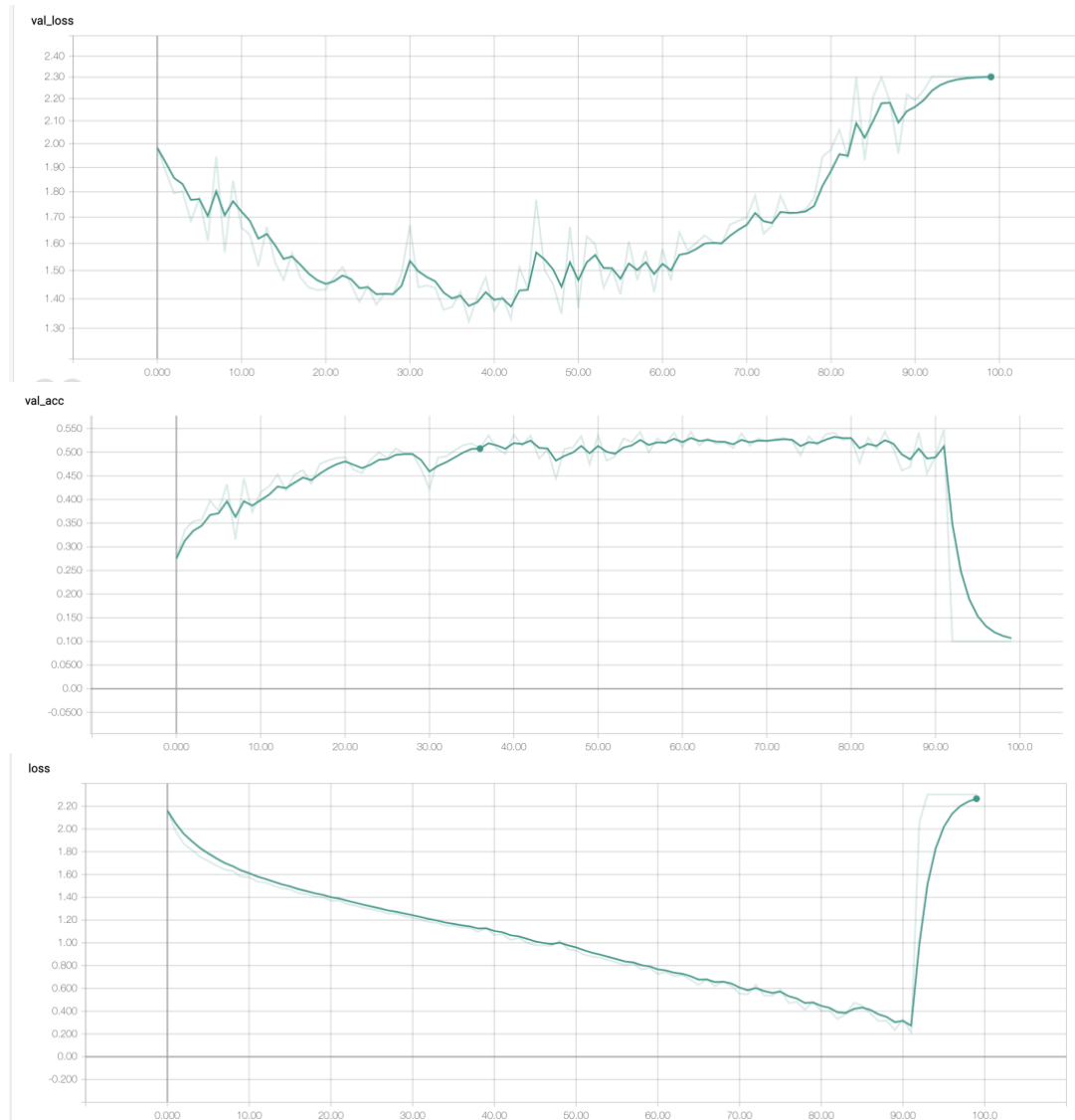
Epochs : 100

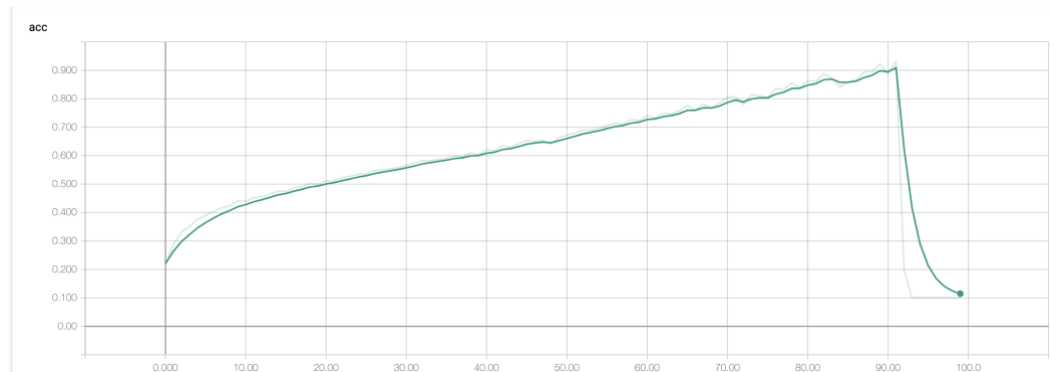
Batch size : 512

Fonction d'activation : Relu

Fonction de perte : categorical_crossentropy

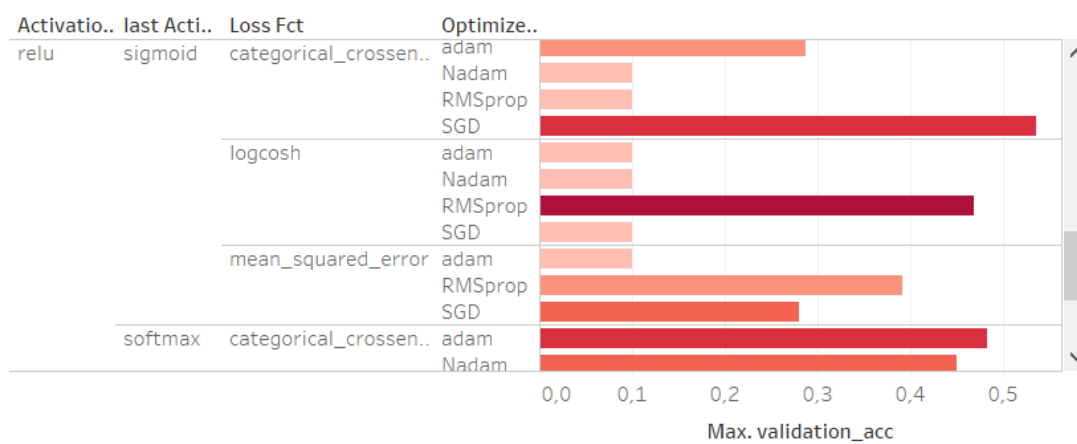
Optimizer : SGD



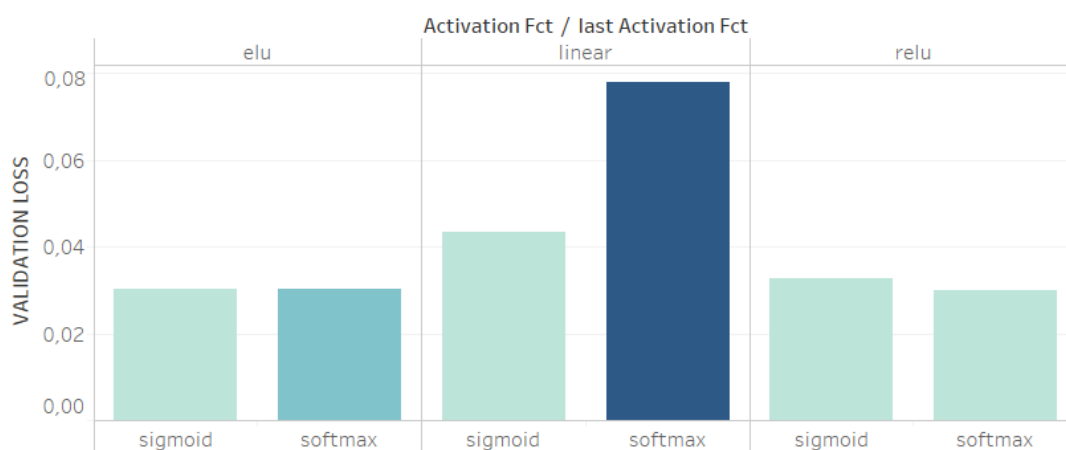


Au niveau du MLP, nous avons aussi pas mal de sur-apprentissage en fonction des combinaisons des fonctions utilisées dans le réseau du neurone du modèle.

Hyperparameters influence on Validation Accuracy



Hyperparameters influence on Validation Loss



Conclusion

Malgré qu'on soit passé sur un modèle multicouche le modèle a du mal à s'adapter avec le dataset. Il présente encore du sur-apprentissage mais également parfois du sous-apprentissage.

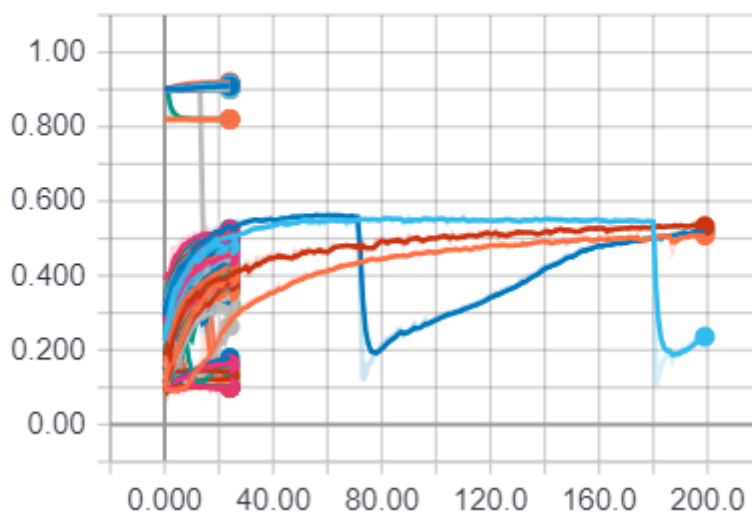
RNN

Postulat

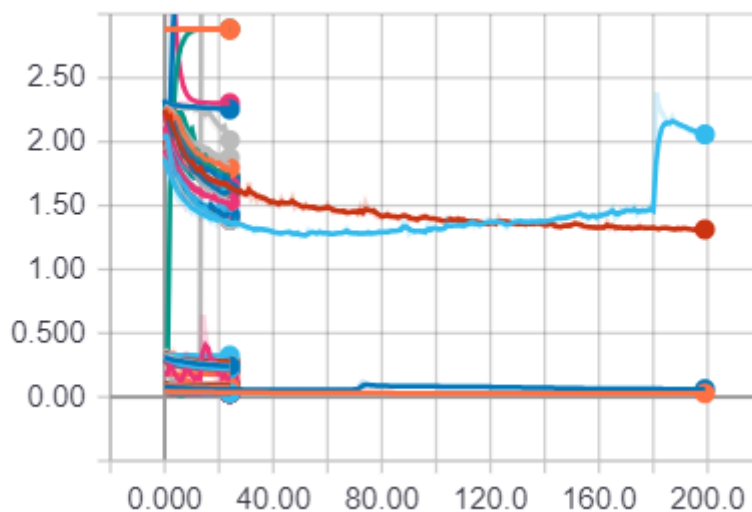
Comme précisé dans la section précédente, les RNN sont généralement utilisés pour l'analyse du langage naturel. De ce fait, il n'en est pas attendu de résultats satisfaisants. Cependant, l'expérimentation de celui-ci sur des images affirmera ou infirmera nos attentes.

Résultats

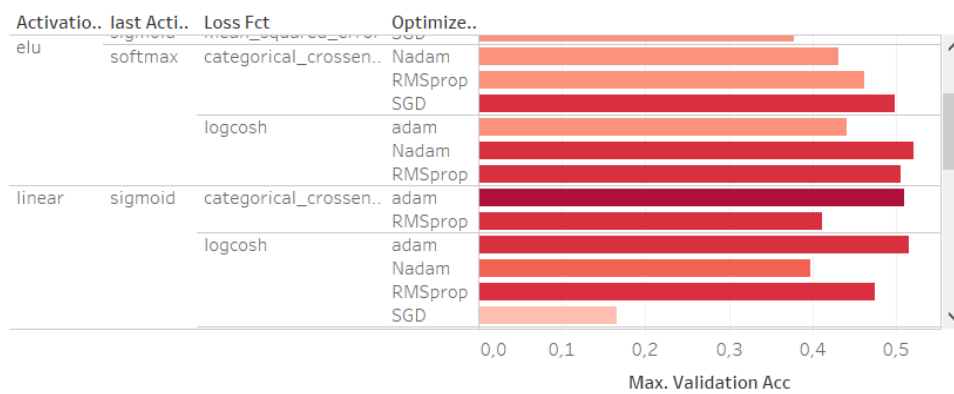
val_acc



val_loss



Hyperparameters influence on Validation Accuracy



Conclusion

Bien que le RNN soit généralement utilisé pour des données textuelles, nous avons obtenu une accuracy de 50% .

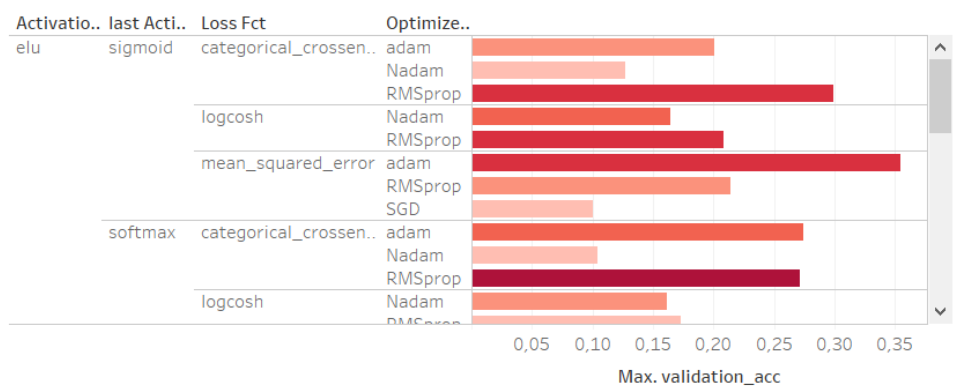
CNN

Postulat

Le CNN est le modèle le plus approprié pour le dataset cifar10. Des résultats à plus de 90% existent. Toutefois, au vu de nos modèles, il semble préférable d'attendre un résultat bien plus faible.

Résultats

Hyperparameters influence on Validation Accuracy



Conclusion

Durant nos tests, le meilleur résultat a été de 71%, ce qui est un des meilleurs résultats obtenus. Il aurait été intéressant de tester sur plus d'epochs afin de voir l'évolution du résultat (en hausse ou en baisse)

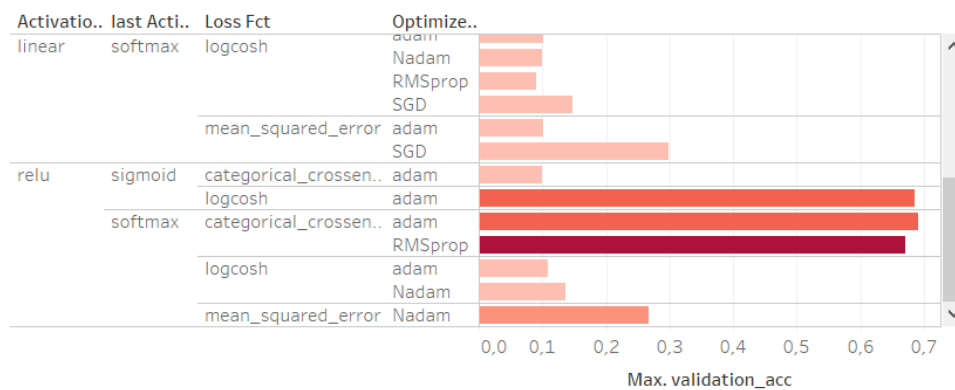
CNN-LSTM

Postulat

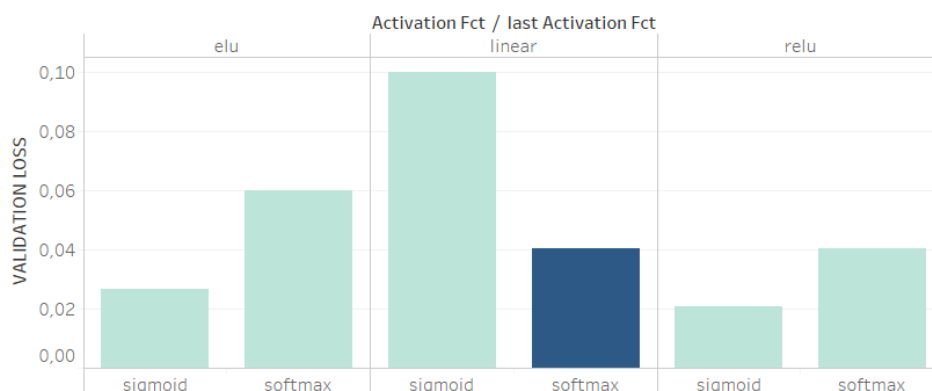
Surpris des résultats du RNN, nous avons pensé à le combiner au CNN. Dans un premier temps, le CNN extrait les caractéristiques des images d'entrée. Dans un second temps, les couches LSTM récupère les données de sortie du CNN (les features) et procède au fitting.

Résultats

Hyperparameters influence on Validation Accuracy



Hyperparameters influence on Validation Loss

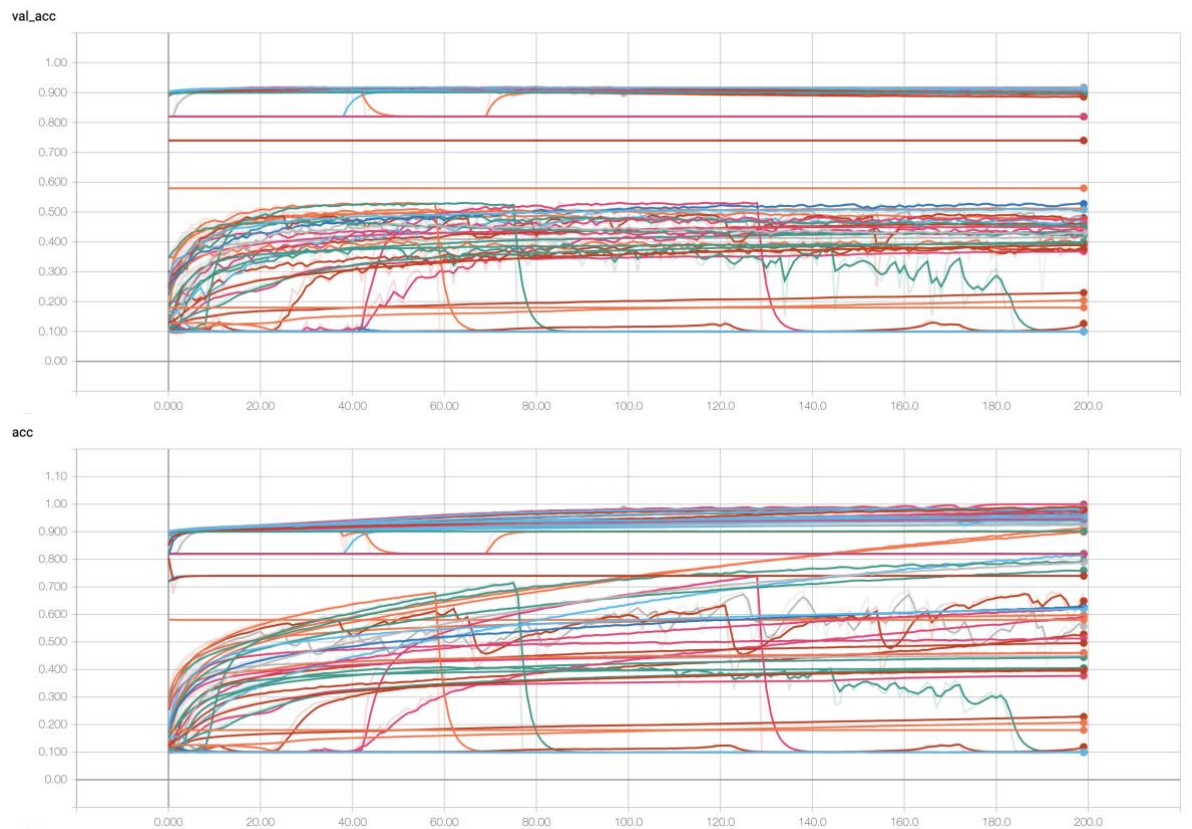


Conclusion

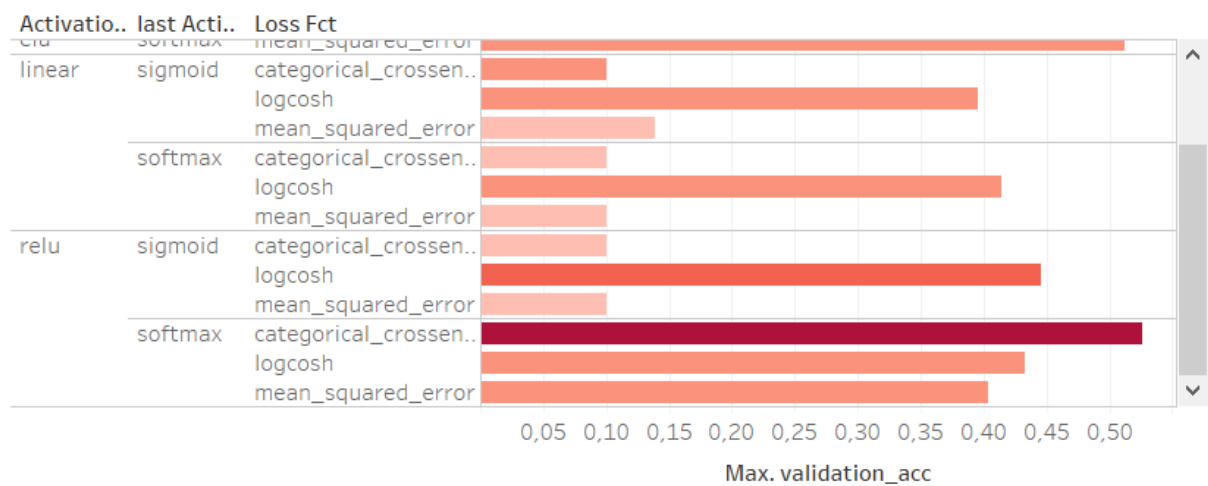
La combinaison CNN-LSTM nous a permis de pousser les performances du RNN et de monter jusqu'à 70% d'accuracy.

Resnets

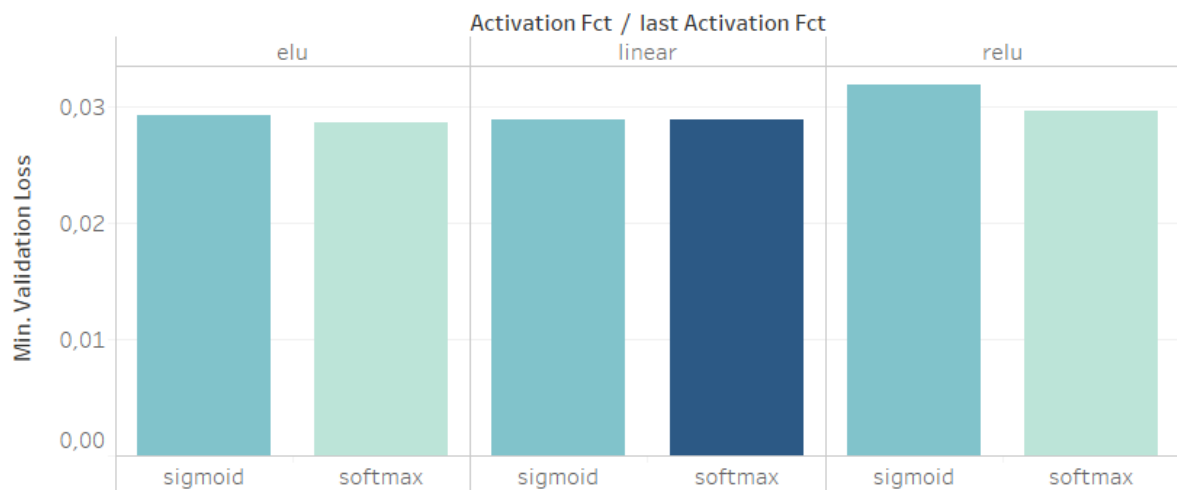
Résultats



Hyperparameters influence on Validation Accuracy



Hyperparameters influence on Validation Loss



Activation: relu

Epoch: 200

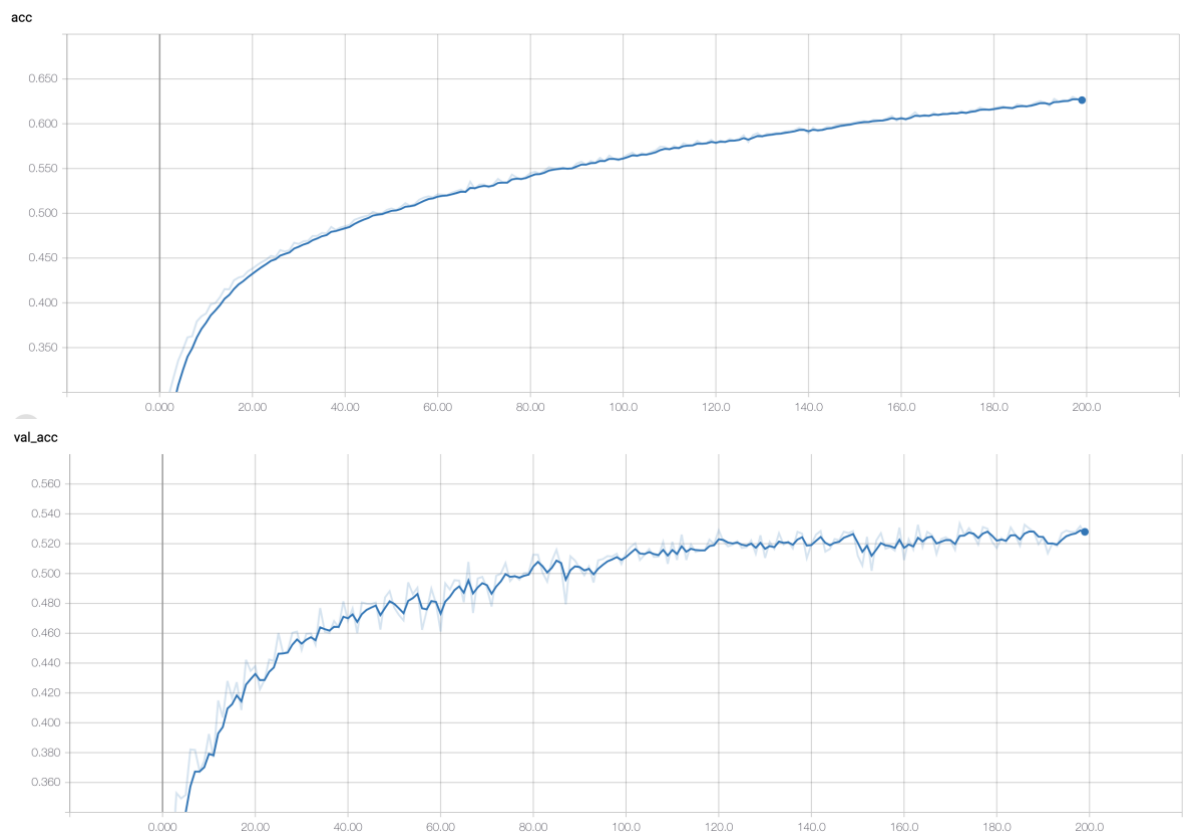
input_shape : 3072

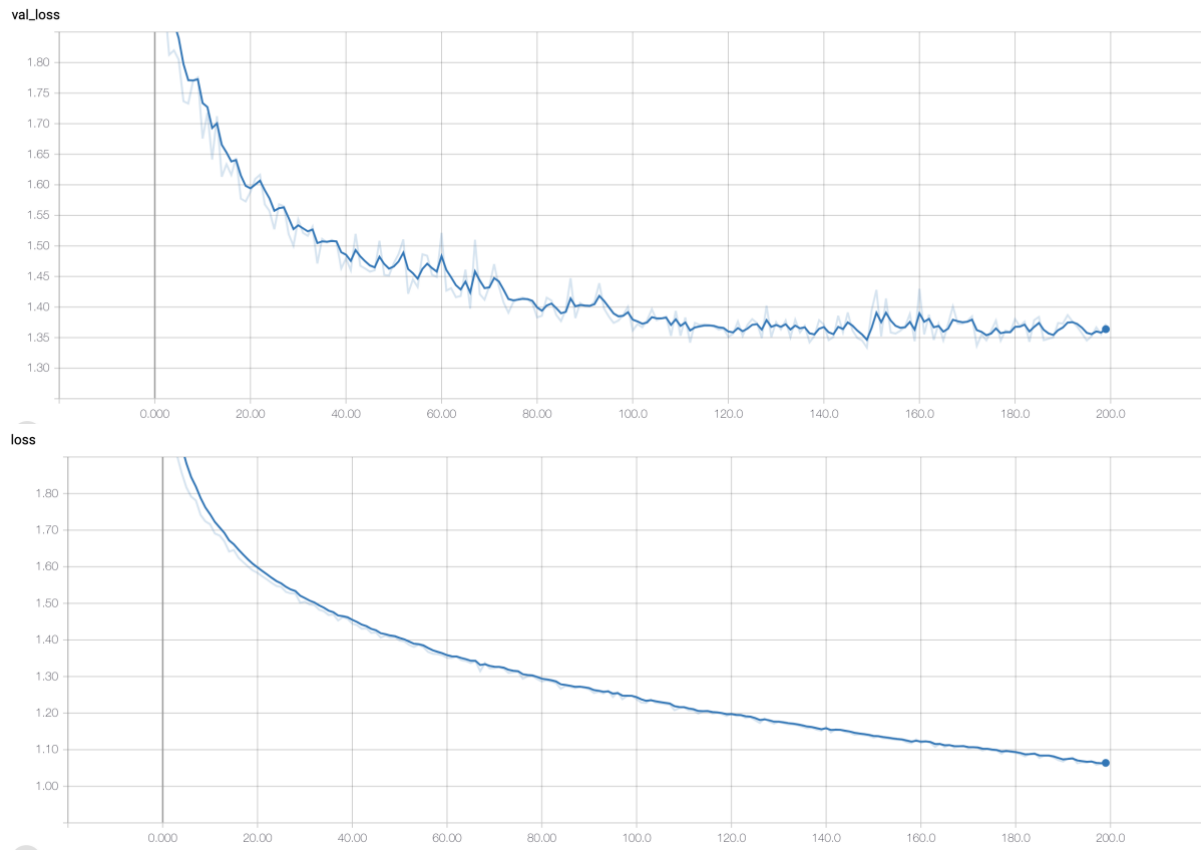
last_activation : softmax

loss_function: categorical_crossentropy

lr : 0.05271792774789524

optimizer: SGD





Le modèle Resnet performe assez bien sur 100 epochs avec comme fonction d'activation relu, dernière fonction sur sa couche de sortie softmax et comme fonction de perte categorical_crossentropy. Avec ces combinaisons on atteint comme valeurs :

Accuracy :0.625

average_acc :0.527

average_loss : 1.372

binary_loss : 1.065

validation_acc : 0.527

validation_loss : 1.372

Conclusion

Le Resnet performe assez bien sur le dataset cependant avec 100 epochs on ne peut pas valider dans la globalité si le modèle est fiable ou pas.

Conclusion

Nous constatons que des tests avec plus d'epochs sur l'ensemble des modèles auraient été plus intéressant et révélateur de l'évolution des modèles.

Toutefois, nous avons une idée plus précise des hyperparamètres qui peuvent être utilisés pour le dataset cifar10 ainsi que des modèles qui peuvent performer.

Par ailleurs, une meilleure intégration de Mlflow aurait d'une précieuse aide dans le filtrage des modèles ainsi que des fonctions qui peuvent être sélectionnées.

Lexique

Algorithme : le perceptron est une suite d'opérations et de calcul = la somme des entrées, leur pondération, la vérification d'une condition et la production d'un résultat d'activation.

Apprentissage : l'algorithme doit être "entraîné", c'est à dire qu'en fonction d'une prédiction voulue, le poids des différentes entrées va évoluer et il faudra trouver une valeur optimale pour chacune.

Supervisé : l'algorithme trouve les valeurs optimales de ses poids à partir d'une base de données d'exemples dont on connaît déjà la prédiction. Par exemple on a une base de données de photos de banane et on "règle" notre algorithme jusqu'à ce que chaque photo (ou presque) soit classé comme *banane*.

Classification : l'algorithme permet de prédire une caractéristique en sortie et cette caractéristique sert à *classer* les différentes entrées entre elles. Par exemple, trouver toutes les bananes dans un panel de photos de fruits.

Binaire : l'algorithme sépare un ensemble de valeurs d'entrée en seulement deux classes différentes.

Linéaire : l'algorithme sépare un ensemble de valeurs de manière linéaire. Prenons l'exemple d'une feuille de papier où vous auriez tracé une diagonale *droite*. Cette diagonale est une séparation linéaire de la feuille de papier. Si vous aviez tracé un rond au milieu de la feuille, la séparation serait considérée comme non linéaire.