



réseaux neuronaux artificiels

l'apprentissage d'une machine

Da Silva Gameiro Henrique

Table des matières

1	Introduction	2
1.1	Définition du Machine learning	2
1.2	Types d'apprentissage	2
1.3	L'histoire du machine learning en bref:mettre des images de personnes	3
2	Premier modèle: le perceptron	5
2.1	Principe général du perceptron	5
2.2	Algorithme d'apprentissage du perceptron	6
2.3	Exemple d'apprentissage du perceptron	7
2.4	Exemple d'utilistaion du perceptron avec des portes logiques	8
2.5	Algorithme de correction des poids et biais	8
2.6	Obtention de la règle d'apprentissage du perceptron	9
2.7	régression logistique(activation sigmoid	10
3	Les réseaux de neurones artificiels (ou ANN pour artificial neural network)	11
3.1	problème des classificateurs linéaires à une couche	11
3.2	les perceptrons multi-couches	11
3.3	définition d'un réseau de neurone à L couche	12
3.4	backpropagation ou rétropropagation des gradients/erreurs	12
3.5	mise à jour des paramètres dans un réseau de neurones	13

3.6	exemple d'apprentissage d'un réseau de neurone avec backpropagation	13
3.7	optimisation avec mini-batch	14
3.8	Mesure de la performance d'un modèle	15
3.9	les hyper-paramètres	17
3.10	les types d'optimisation	17
3.11	régularisation	17
3.12	data augmentation et overfitting	17
3.13	Fonctions d'activation	18
3.14	les fonctions de coût ou pertes	19
3.15	pandas ou comment manipuler la data(les données)	19
3.16	tensorflow et keras	19
4	deeplearning	21
5	prédire la météo avec le machine learning: comparaison de performance avec SGD, tanh, adagrad, prendre des screenshots ...	22
6	Les CNN(convolutional neural network)	23
6.1	principes	23
6.2	convolution	23
6.3	couche de pooling	24
6.4	dernières couches d'un CNN	25
6.5	les hyper-paramètres d'un CNN	25
6.6	exemple de reconnaissance avec cifar10	25
7	ML et marketing,journal,...	27

1. Introduction

1.1 Définition du Machine learning

Le machine learning ou apprentissage automatique est un sous-ensemble de l'intelligence artificielle(A.I) selon l'encyclopédie Larousse, l'A.I. se définit comme «l'ensemble de théories et de techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence »[14]. Le machine learning fait donc parti de cette ensemble.

Comme ce domaine a beaucoup évolué au cours des années, mais l'idée était déjà définie en 1959 par Arthur Samuel:«Le machine learning est le domaine d'étude qui donne à l'ordinateur la capacité d'apprendre sans être explicitement programmé»(traduit de l'anglais).

Tom Mitchell définit aussi l'apprentissage d'un algorithme de machine learning en 1998 «On programme un algorithme informatique pour qu'il apprenne d'une expérience E selon une certaine fonction T(task) et une mesure de performance P, si sa performance aux fonctions T, mesurée par P, augmente avec l'expérience E »

On peut illustrer cette définition par un exemple: dans le cas d'un algorithme qui apprendrait à jouer à un jeu, le E serait l'expérience d'avoir joué ou vu quelqu'un joué, le T serait la fonction de joueur à ce jeu et P la probabilité de gagner une partie. Ces notions sont présentes aussi pour parler des différents concepts dans un réseau de neurone, mais avec un nom différent.

Le E sera l'ensemble d'entraînement D et le P sera l'erreur ou le delta entre ce qu'on cherche à obtenir et ce qu'on prédit.[16]

1.2 Types d'apprentissage

Pour résoudre un problème de Machine learning, il faut d'abord déterminer le type d'algorithme ou type de réseau de neurone.

Il existe 2 grandes catégories d'apprentissages: supervisé et non-supervisé

Dans un algorithme d'apprentissage supervisé, nous utilisons des exemples d'entraînement avec un ensemble de départ et d'arrivée. A partir de ces exemples, l'algorithme va apprendre; c'est comme si par de la "data" nous apprenions à la machine à par exemple reconnaître des images.

En apprentissage supervisé, il existe encore des sous-catégories(parfois combinés):

Tous d'abord, les algorithmes de régression: ceux-ci prédisent une valeur future en continu à partir de résultats passés comme un prix d'un appartement par exemple.

Chapter 1. Introduction

La 2ème catégorie est la classification: dans ces algorithmes, les valeurs d'ensemble d'arrivées sont des classes comme des chiffre ou un type d'image. Le but est que l'algorithme trouve la classe correspondante à un élément de l'ensemble de départ. Pour ce faire, on utilise une fonction linéaire ou quadratique pour faire une séparation entre classes.

Dans les problèmes d'apprentissage non-supervisé, il n'y a pas d'ensemble d'arrivée dans nos exemples d'entraînement: nous ne connaissons pas à l'avance le résultat que doit obtenir l'algorithme et on peut ainsi résoudre des problèmes sans savoir à quoi le résultat devrait ressembler.

Leurs utilisations est très vaste les plus basiques sont exemple de trouver des liens entre des éléments de notre data et les grouper comme le fait d'identifier des groupes d'amis utilisé par facebook ou lorsqu'on fait une recherche sur un moteur de recherche, trouver des sites en liens avec ce qu'on recherche. Justement, le terme utilisé pour évaluer l'efficacité de ce type d'algorithme, c'est-à-dire le degrés de ressemblance entre ce qu'on cherche à obtenir et ce qu'on obtient s'appelle "fitness".[14]

1.3 L'histoire du machine learning en bref:mettre des images de personnes

Aussi surprenant que cela puisse paraître, les premiers réseaux de neurones artificiels ont été théorisés avant la création des premiers ordinateurs. En effet, le moteur premier de l'avancer du machine learning a été les neurosciences dans un premier temps(est les toujours en partie).

Ce sont les neuroscientifiques Walter Pitts et Warren McCulloch qui ont eu l'idée d'appliquer les mathématique aux réseaux de neurones pour reproduire leur fonctionnement dans leur travail de séminaire "A Logical Calculus of Ideas Immanent in Nervous Activity". Leur idée est de reproduire la pensée avec des calculs mathématiques.

Alan Turing ,théoricien de l'ordinateur et de la programmation, voyait déjà, en 1950, l'impact considérable que pourrait avoir le machine learning dans le futur. Il a créé aussi le célèbre test de Turing établissant un seuil où il considère que l'algorithme est assez intelligent pour penser comme une humain. Le test consiste en une simple conversation entre un humain et une intelligence artificielle puis avec un humain. L'intelligence réussit le test si l'humain ne peut pas discerner lequel des deux est un ordinateur.Certains pensent que ce test a déjà été passé aujourd'hui par certains algorithmes, mais c'est un sujet très débattu(notamment avec la démonstration de google duplex).

En 1952, Arthur Samuel crée un programme jouant aux dames. Ce dernier apprend par lui-même et devient meilleur de partie en partie selon ses erreurs. Cette exemple a mis en évidence un des principes du machine learning qui est l'apprentissage automatique de l'intelligence.

Le premier perceptron est créé,en 1957, par Frank Rosenblatt. C'est la base des réseaux de neurones artificiels encore aujourd'hui. Il consiste en un seul neurone, mais possède déjà les bases d'un réseau de neurone. Cette découverte a produit un grand engouement pour le domaine. Cependant, les limitations du perceptron va étouffer cette élan¹ et il a fallu attendre d'autre découvertes pour que le machine learning prenne la place qu'il a aujourd'hui.

La théorie va passer à la pratique en 1965 avec Alexey Ivakhnenko qui va créer le premier réseau artificiel fonctionnel. Il a aussi développé des réseaux à plusieurs couches jusqu'à 8.

En 1980, le premier réseau à convolution(CNN²) est inventé ,selon le cortex des animaux, pour pouvoir faire de la reconnaissance d'image.

En 1982 est inventé le premier réseau à neurones récurrents par John Hopfield. Les neurones de ce type d'algorithmes sont proches des neurones biologiques puisqu'ils ont une plus grand mémoire que ceux des autre type de réseaux artificiels.

David Rumelhart, Ronald J. Williams et plus particulièrement Geoffrey Hinton ont développé la backpropagation en 1986. Cette découverte a permis d'augmenter considérablement la performance des réseaux et a été un catalyseur pour le machine learning.

¹il y a eu 2 périodes appelées "AI winter"(1974-80 et 1987-93) durant lesquelles l'engouement pour l'intelligence était moindre à cause de blocages technologique

²cf. chapitre sur les CNN

Chapter 1. Introduction

En 1989, le français Yann LeCun a mis au point un algorithme permettant de lire des chiffres écrits à la main en utilisant les CNN et la backpropagation. Cette application deviendra un étalon de mesure de la performance d'un réseau.

Dans cette même période, le Q-learning (plus tard apprentissage par renforcement) est découvert par Christopher Watkins. Le principe est de corriger le réseau par récompenses lorsque le réseau fait un bon choix. Le but est donc de prendre des actions maximisant la récompense plutôt que de minimiser l'erreur.[3], [17]

2. Premier modèle: le perceptron

2.1 Principe général du perceptron

Le principe du perceptron est de modéliser une décision par une fonction linéaire et un seuil (Threshold en anglais) pour pouvoir classer des entrées x_1, x_2, \dots, x_n et qu'elle formule une hypothèse $hw(x)$ qui doit s'approcher d'une sortie y voulue [1], [4], [10]

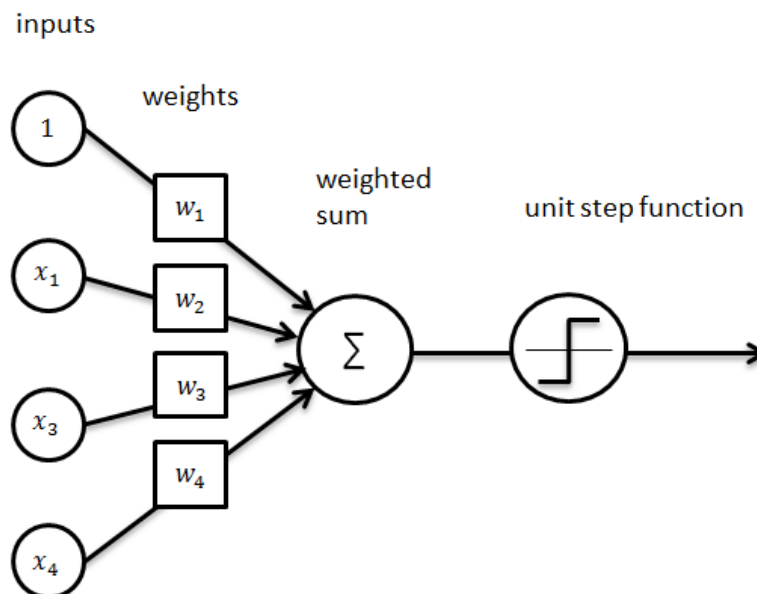


Figure 2.1: perceptron

Chapter 2. Premier modèle: le perceptron

$$hw(x) = \text{Threshold}(W \cdot x) \quad (2.1)$$

où $\text{Threshold}(z) = 1$ si $z \geq 0$, sinon $\text{Threshold}(z) = 0$: c'est la prise de décision, l'activation d'un neurone.

L'activation, dans le perceptron, est une fonction linéaire. On peut donc se représenter le fonctionnement d'un perceptron par une fonction linéaire c'est à dire une droite : celle-ci sépare les éléments en entrée en 2 classes. Le perceptron est donc un classificateur linéaire qui ne peut traiter que les problèmes transformable linéairement(cf.chapite 3.1).

Plus généralement, l'activation est en fait inspiré du cerveau. Un neurone biologique ne s'allume que si il reçoit un signal électrique dépassant un certains seuil. Pour notre perceptron, c'est exactement ce principe qui est utilisé. z correspond au produit matricielle des vecteurs-colonnes entrés(x_1, x_2, \dots, x_n) et poids(w_1, w_2, \dots, w_n).

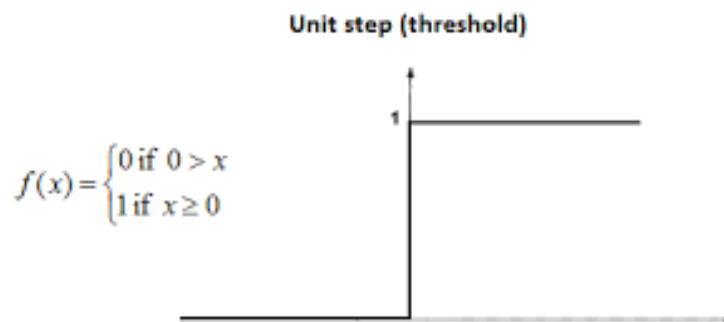


Figure 2.2: fonction threshold

Le poids d'une connexion,appelé aussi poids synaptique, représente l'influence d'une connexion sur un neurone, c'est-à-dire la valeur d'un input x_n par rapport à l'output y . On pourrait représenter ceci dans un exemple concret pour prendre une décision:imaginons qu'on doive décider d'assister à un concert on pourrait prendre par exemple 4 facteurs qui influenceraient notre décision: x_1 le groupe qui joue à ce concert, x_2 le prix du billet, x_3 les personnes qui vont nous accompagner et x_4 la date du concert. Imaginons que le groupe qui joue nous importe peu(w_1),que nous sommes en difficulté financière(w_2),que nous voulons absolument y aller en grand nombre(w_3) et que nous sommes en vacances. Nous remarquons que certaines variables ont une plus grand variables : x_2 et x_3 sont plus important et par conséquent la valeur de w_2 et w_3 en saura d'autant plus grand que l'impact de x_2 et x_3 sur notre décision.

Le but de l'algorithme est de trouver les bonnes valeurs pour le vecteur de poids W : c'est le paramètre, ce qui va changer au cours des entraînements. On utilise cependant aussi un biais b qui sert à faire varier le seuil pour trouver le bon,le biais représente la difficulté d'un neurone à s'activer,doit-il s'activer fréquemment ?(comme un neurone biologique), plus il est élevé,plus un neurones s'active facilement et, au contraire, plus il tend vers les négatifs plus l'activation du neurone sera difficile, le biais peut être considéré comme 1 neurone ajouté en plus à chaque couche non influencé par la couche précédente :

$\text{Threshold}(z) = 1$ si $z \geq -b \Rightarrow \text{Threshold} = -b$ va donc aussi varier au cours de l'entraînement cf <https://www.quora.com/What-is-bias-in-artificial-neural-network>

2.2 Algorithme d'apprentissage du perceptron

L'algorithme d'apprentissage doit adapter les paramètres W et b pour faire en sorte que la prédiction $hw(x)$ soit la bonne valeur pour l'ensemble d'entraînement.

Chapter 2. Premier modèle: le perceptron

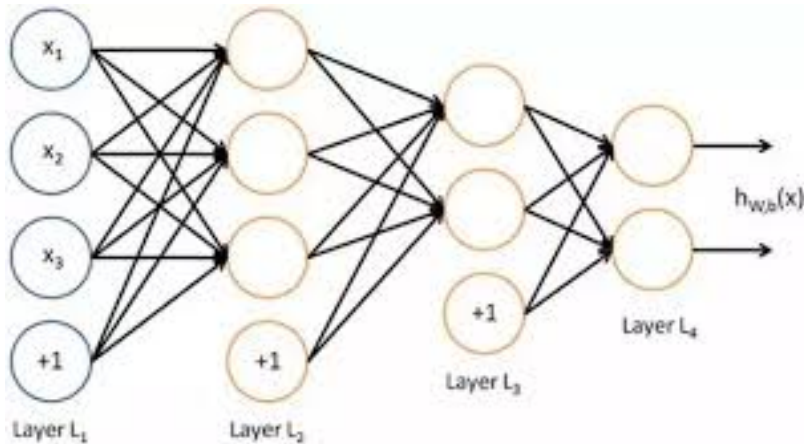


Figure 2.3: réseau de neurones avec biais

R le +1 représente le biais

1) Pour chaque paire $(x_t, y_t) \in D$ (x_t = input, y_t = target output et D = ensemble d'entraînement):

- calculer $hw(x_t) = \text{Threshold}(w \cdot x_t + b)$
- si $y_t \neq hw(x_t)$: $w_i \leftarrow w_i + \eta(y_t - hw(x_t)) \cdot x_{t,i}$

2) Retourner à l'étape 1 (itération) jusqu'à ce qu'on atteigne un critère d'arrêt comme un nombre maximal d'itération ou un seuil d'erreur satisfaisant.

-La mise à jour des poids est appelée la règle d'apprentissage du perceptron. Son obtention sera détaillé dans la suite du document.

-Le coefficient η est le taux d'apprentissage et est lié au gradient.

Dans la pratique ce calcul se fera sous la forme matricielle pour une plus grande efficacité:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} + (b) = \begin{pmatrix} hw(x_1) \\ hw(x_2) \\ \vdots \\ hw(x_n) \end{pmatrix} \approx \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

2.3 Exemple d'apprentissage du perceptron

	x_t	y_t
Ensemble d'entraînement $D[11]$:	$[2, 0]$	1
	$[0, 3]$	0
	$[3, 0]$	0
	$[1, 1]$	1

initialisation des paramètres ex: $\eta = 0.1, W = [0, 0], b = 0.5$:

1ère itération avec (x_1, y_1) :

- $h(x_1) = \text{Threshold}(W \cdot x_1 + b) = \text{Threshold}([0, 0] \cdot [2, 0] + 0.5) = \text{Threshold}(0.5) = 1$:

Chapter 2. Premier modèle: le perceptron

- $h(x_1) = y_1 \Rightarrow$ pas de mise à jour de W et b

2ème itération avec (x_2, y_2) :

- $-h(x_2) = \text{Threshold}(W \cdot x_2 + b) = \text{Threshold}(0.5) = 1$:
- puisque $h(x_2) \neq y_2$, on met à jour W et b :
 $w \leftarrow w + \eta(y_2 - h(x_2)) \cdot x_2 \leftrightarrow [0, 0] + 0.1 \cdot (0 - 1) \cdot [0, 3] = [0, -0.3]$
 $b \leftarrow b + \eta(y_2 - h(x_2)) \cdot 1 = 0.5 + 0.1(0 - 1) = 0.4$

3ème itération avec (x_3, y_3) et $W = [0, -0.3]$ et $b = 0.4$:

- $-h(x_3) = \text{Threshold}(W \cdot x_3 + b) = \text{Threshold}(0.4) = 1$, puisque $h(x_3) \neq y_3$, mise à jour de W et b :
 $w \leftarrow w + \eta(y - h(x_3)) \cdot x_3 \leftrightarrow [0, -0.3] + 0.1(0 - 1) \cdot [3, 0] = [-0.3, -0.3]$
 $b \leftarrow 0.4 + 0.1(0 - 1) = 0.3$

4ème itération avec (x_4, y_4) et $W = [-0.3, -0.3]$, $b = 0.3$

- $-h(x_4) = \text{Threshold}(W \cdot x_4 + b) = \text{Threshold}(-0.3) = 0$, puisque $h(x_4) \neq y_4$ mise à jour de W et b :
 $w = [-0.2, -0.2]$
 $b = 0.4$

... \rightarrow itérations jusqu'au critère d'arrêt.


2.4 Exemple d'utilisation du perceptron avec des portes logiques

Les portes logiques constituent la base d'un ordinateur : elles permettent d'effectuer des opérations ou instructions avec des bytes ayant deux états possibles (0 ou 1)

Fonctionne pour OR,... mais pas XOR pas lin sép. On doit utiliser un perceptron multicouche (MLP) avec 3 neurones ce qui va donner à plus grande échelle un NN de type feed forward \neq recurrent appelé réseau convolutif utilisé pour la reconnaissance d'image

2.5 Algorithme de correction des poids et biais

Trouver les bons paramètres consiste en un problème d'optimisation: trouver le minimum d'une erreur. Cette erreur est la distance entre l'output ciblée et la prédiction: $\text{delta} = y_t - h(w \cdot x_t)$

 Nb Cette distance est aussi souvent appelée coût ou encore perte

Cette fonction dépend des variables pour $h(x_t)$: W et b , ce sont donc plus d'une variable qu'il faut minimiser et donc il faut considérer l'utilisation de la dérivée partielle et le gradient ∇ .

Le gradient nous donne la direction de l'augmentation la plus grande de la fonction. Prendre $-\nabla$ avec un coefficient (le taux d'apprentissage η) permet de diriger la fonction erreur vers un minimum local (pas forcément le bon) et ainsi de minimiser cette fonction. Ce processus est appelé descente de gradient.

En revanche, calculer le gradient pour tout l'ensemble d'entraînement et ensuite un gradient serait en vérité beaucoup trop long si l'ensemble d'entraînement est grand, car on devrait calculer la moyenne des dérivées sur tous les exemples d'entraînement. Pour cela, on utilise la descente de gradient stochastique le but est de sélectionner quelques exemples d'entraînement aléatoirement pour obtenir un "mini-batch" il s'avère que calculer le gradient moyen pour ce "mini-batch" revient à trouver une approximation proche de la vraie valeur du gradient.

Chapter 2. Premier modèle: le perceptron

Cette approximation ne pose pas de problème en machine learning car nous ne cherchons pas à avoir une réponse exacte mais connaître une direction vers laquelle on veut que nos variables tendent. Ensuite, après avoir une correction avec le 1er "mini-batch", on en prend un 2ème et ainsi de suite jusqu'à avoir utilisé tout les exemples d'entraînement. Toute cette procédure constitue ce qu'on appelle une "epoch" et donc ce processus de "mini-batch" recommence pour chaque "epoch". La progression de l'erreur peut être représenté par une courbe appelé courbe d'apprentissage c'est le taux d'erreur par itération.



Figure 2.4: courbe d'apprentissage

2.6 Obtention de la règle d'apprentissage du perceptron

On utilise le gradient avec les dérivées partielles pour avoir une direction de mise à jour des paramètres[12], [13]:

$$\nabla = \frac{\partial}{\partial w_i} \text{Loss}(y_t, hw(x_t)) = \frac{\partial}{\partial w_i} (y_t - hw(x_t)) \cdot w \cdot x_t = -(y_t - hw(x_t))x_{t,i}$$

Comme le gradient donne la direction d'augmentation la plus grande de la perte, on va dans la direction opposée $-\nabla$ pour la mise à jour des poids. A noter que pour la backpropagation, seule les dérivées partielles selon les poids sont nécessaires, alors le gradient consiste seulement en la dérivée partielle de la fonction selon les poids. :

$$w_i \leftarrow w_i - \eta \frac{\partial}{\partial w_i} \text{Loss}(y_t, hw(x_t)) \forall i$$

En remplaçant le gradient avec le gradient qu'on a obtenu plus haut on obtient la règle d'apprentissage du perceptron:

$$w_i \leftarrow w_i + \eta (y_t - hw(x_t)) \cdot x_{t,i} \forall i$$

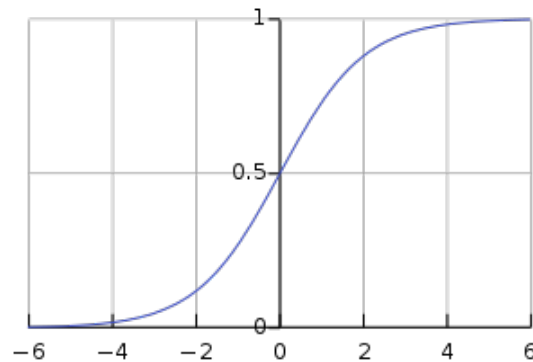
Chapter 2. Premier modèle: le perceptron

2.7 régression logistique(activation sigmoid)

Le problème d'utiliser la fonction Threshold comme activation est que cette fonction n'est pas dérivable partout, elle est discontinue; ceci peut mener à des approximations.[6]

Il y a donc d'autre fonction d'activation est celui qui va nous intéresser dans ce chapitre est sigmoid:

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$



Cette fonction à l'avantage de ne pas être discontinue. Elle converti l'entrée x en probabilité
L'idée de ce système de classification est de se baser sur la probabilité d'appartenir à une classe:

$$p(y = 1|x) = hw(x) = Sigmoid(w \cdot x) = \frac{1}{1 + \exp^{-w \cdot x}}$$

A partir de cette probabilité, l'algorithme choisit la classe la plus probable.

3. Les réseaux de neurones artificiels (ou ANN)

3.1 problème des classificateurs linéaires à une couche

Les perceptrons à une couche fonctionnent très bien pour les problèmes dit séparable linéairement, c'est-à-dire les cas où les différents objets regroupés en classe (ex: pour les fonctions logiques 0 est une classe et 1 une autre) puissent être séparés par une fonction linéaire comme c'est le cas pour les fonctions logiques ET, OU, et même implique.[7]

schéma:

Cependant, ce même modèle du perceptron ne fonctionne pas pour des problèmes qui ne paraissent à première vue pas plus compliqué; c'est le cas pour XOR.

En effet, la différence du où exclusif et qu'on ne peut pas séparer les classes d'objets par une droite : il n'est non-séparable linéairement

schéma:

Le perceptron de base ne peut donc par conséquent pas apprendre XOR, Ce problème est cependant fréquent. En effet, la plus part des problèmes ne sont pas séparable linéairement.

3.2 les perceptrons multi-couches

Pour résoudre des problèmes non-linéaires, une des premières solutions trouvées a été de créer des perceptrons à plusieurs couches ou MLP (*multi-layer perceptron*)

L'idée derrière ces systèmes est de transformer l'entrée pour la rendre linéairement séparable avec une première couche et ensuite une deuxième couche qui fonctionne comme les perceptrons vus précédemment.

Pour le cas du XOR, une des solutions serait d'avoir 2 neurones sur la 1ère couche: un premier qui va transformer l'entrée X_1 par $ET(x_1, x_2)$ et un 2ème neurone qui va transformer x_2 par $OU(x_1, x_2)$. Ainsi, le problème sera linéairement séparable

schéma:

Dans ce modèle on a donc un neurone qui va apprendre OU un autre ET qui seront reliés à un autre neurone sur une 3ème couche qui va apprendre XOR ex code:

Ce processus est encore assez primitif d'utiliser plusieurs couches de neurones pour rendre le problème séparable linéairement est pourtant la base de ce qui va nous intéresser plus particulièrement dans ce travail à savoir les réseaux de neurones artificiels

3.3 définition d'un réseau de neurone à L couche

Le but d'un réseau de neurone est, comme pour le perceptron, d'apprendre des poids pour pouvoir classer, mais les couches intermédiaires("hidden layer") permettent de rendre le problème linéairement séparable ou, plus basiquement, juxtaposer des couches de neurones permet d'avoir une corrélation plus évidente entre l'entrée et la sortie.

Par exemple, dans le cas de la reconnaissance de nombres, on peut imaginer que chaque couche décomposerait les nombres en parties et essaierait de reconnaître des patrons dans les pixels actifs pour pouvoir les faire correspondre à un nombre selon les parties constituant l'image d'entrée. Cependant, ce n'est pas ce que fait réellement l'algorithme, mais il suit ce principe.

Il existe plusieurs notations pour un réseau de neurones, l'une d'entre elle numérote les couches de haut en bas selon les couches. Les poids sont notés $w_{j,k}$ (j et k représente les 2 neurones que la liaison relie), a_j est l'activité d'un neurone j et in_j l'activité du neurone j avant l'activation :

$$a_j = \text{Sigmoid}(in_j)$$

NB.: $in_j = z$ notations: j
k

3.4 backpropagation ou rétropropagation des gradients/erreurs

Dans le perceptron à une seule couche, calculer l'erreur de la dernière couche consistait en la différence entre la cible y et notre prédiction $hw(x)$.

Cependant, dans ce cas-ci il n'y a plus une seule couche et cette méthode ne fonctionne pas pour calculer l'erreur des neurones des couches cachées(ou hidden layer)

Pour cela on utilise la dérivée en chaîne(ou dérivée d'une fonction $f(g(x))$),

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(x)}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x} \Rightarrow \frac{\partial f(x)}{\partial x} = \sum_i \frac{\partial f(x)}{\partial g_i(x)} \cdot \frac{\partial g_i(x)}{\partial x}$$

elle nous permet de calculer une dérivée à partir d'un résultat intermédiaire. Ainsi, on peut calculer la dérivée d'un neurone à une couche L à partir de la dérivée des neurones connectés à ce dernier à la couche suivante L+1.[8]

Ce processus de retour en arrière s'appelle "backpropagation".

Elle constitue la 3ème étape de l'apprentissage et est un mouvement dans le sens contraire du "feed forward" et elle débute par le calcul de l'erreur c'est-à-dire la différence entre le résultat ciblé et celui obtenu.

La backpropagation consiste donc à se servir de l'écart entre notre prédiction et le résultat ciblé et de propager cette erreur dans l'autre sens. Avec les poids des différents liens entre les neurones, on peut calculer l'erreur de chaque neurone(trouver le gradient) et ainsi mettre à jour les paramètres.

La dérivée partielle appliquée au NN donne la formule suivante:

$$\frac{\partial \text{Loss}}{\partial in_j} = \frac{\partial \text{Loss}}{\partial a_j} \cdot \frac{\partial a_j}{\partial in_j} = \left(\sum_k \frac{\partial \text{Loss}}{\partial ink} \cdot \frac{\partial ink}{\partial a_j} \right) \cdot \frac{\partial a_j}{\partial in_j} = \left(\sum_k \frac{\partial \text{Loss}}{\partial ink} \cdot w_{j,k} \right) \cdot a_j(1 - a_j)$$

remarques:

-le j et k fait référence à la notation des neurones et des poids vue au-dessus: le j est le numéro d'un neurone à la couche L(couche actuelle) et k est le numéro du neurone relié au neurone j à la couche L+1(couche suivante)

- $a_j = \text{activation}(in_j)$ où $in_j = W \cdot xn + b$ pour un j-ème neurone

- le premier terme entre parenthèse est la dérivée en chaîne de $\frac{\partial \text{Loss}}{\partial a_j}$ celui-ci signifie qu'on itère à travers les neurones à la couche k(L+1)

Chapter 3. Les réseaux de neurones artificiels (ou ANN pour artificial neural network)

- la dérivée partielle $\frac{\partial ink}{\partial a_j} = w_{j,k}$ et $\frac{\partial a_j}{\partial in_j} = a_j(1 - a_j)$ soit la dérivée de la fonction sigmoid($\frac{df}{dx} = x(1 - x)$)
- pour calculer l'erreur à la dernière couche, il peut être judicieux de mettre la différence au carré pour éviter les valeurs négatives. Ce calcul s'appelle (MSE¹), cependant on peut aussi utiliser la valeur absolue (à noter qu'il existe encore d'autre méthode pour calculer le delta).

3.5 mise à jour des paramètres dans un réseau de neurones

Comme nous sommes toujours dans un problème d'optimisation, la mise à jour des paramètres pour le réseau de neurone se fait de la même façon en principe que le perceptron : on se sert des erreurs de chaque neurone calculé avec la backpropagation nous donnant la direction minimisant l'erreur de la fonction. La formule générale est donc

$$w_{i,j} \leftarrow w_{i,j} + \alpha \cdot a_i \cdot \Delta[j]$$

. $\Delta[j]$ est l'erreur pour chaque neurone j que nous avons vu ci-dessus avec la backpropagation c'est-à-dire pour rappel:

$$-\Delta[j] = \frac{\partial}{\partial in_j} Loss(yt - hw(xt))$$

et

$$a_i = \frac{\partial}{\partial w_{i,j} \cdot in_j}$$

La formule complète est donc:

$$w_{i,j} \leftarrow w_{i,j} - \alpha \cdot \frac{\partial}{\partial in_j} Loss(yt - hw(xt)) \cdot \frac{\partial}{\partial w_{i,j} \cdot in_j}$$

à savoir que $\frac{\partial}{\partial in_j} Loss = sigmoid(z)(1 - sigmoid(z))$ comme vu plus haut.

3.6 exemple d'apprentissage d'un réseau de neurone avec backpropagation

Nous allons reprendre la même donnée que pour le perceptron donnée:

propagation avant:

neurone j_3 : $ak_3^2 = sigmoid(z^3) = \sigma(2 \cdot 2 + 0 \cdot 1.5) = 0,982$

neurone j_4 : $ak_4 = sigmoid(z) = \sigma(2 \cdot (-1) + 0 \cdot (1)) =$

neurone j_5 : $ak_5 = sigmoid(z) = \sigma((-2) \cdot 0,982 + 3 \cdot (0,119)) = 0,167$

neurone j_6 : $ak_6 = sigmoid(z) = \sigma(1 \cdot 0,982 + 2.5 \cdot (0,119)) = 0,782$

neurone j_7 : $ak_7 = sigmoid(z) = \sigma((-2) \cdot 0,167 + 2 \cdot (0,782)) = 0.773$

backpropagation:

$$\Delta_{j7} = y - a = 1 - 0,773 = 0,227$$

$$\Delta_{j6} = \sigma(in_j)(1 - \sigma(in_j)) \cdot \sum_k w_{j,k} \cdot \Delta[k]$$

$$= 0,782 \cdot (1 - 0,782) \cdot 2 \cdot 0,227 = 0,077$$

$$\Delta_{j5} = 0,167 \cdot (1 - 0,167) \cdot -2 \cdot 0,227 = -0,063$$

¹pour mean squared error traduit en erreur quadratique moyenne en français

²pour rappel le neurone k est le neurone à la couche $L+1$ par rapport à j

³ $z = \sum_j w_{j,k} \cdot a_j$

Chapter 3. Les réseaux de neurones artificiels (ou ANN pour artificial neural network)

$$\Delta_{j4} = 0,119 \cdot (1 - 0,119) \cdot (2,5 \cdot 0,077 + 3 \cdot (-0,063)) = 0,000366$$

$$\Delta_{j3} = 0,982 \cdot (1 - 0,982) \cdot (1 \cdot 0,077 + (-2) \cdot (-0,063)) = 0,00358$$

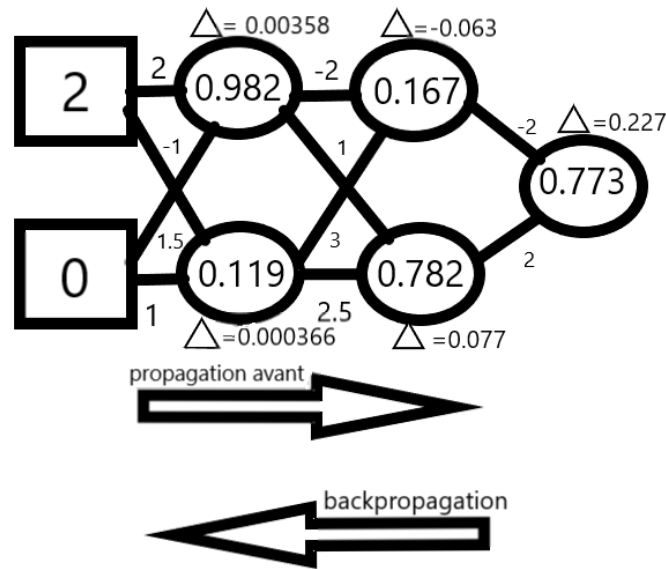


Figure 3.1: exemple de réseau de neurones

mise à jour:

$$w_{1,3} \leftarrow w_{i,j} + \alpha \cdot a_i \cdot \Delta[j] = 2 + 0,9 \cdot 2 \cdot 0,00358 = 2,0064$$

$$w_{1,4} \leftarrow (-1) + 0,9 \cdot 2 \cdot 0,000366 = -0,99934$$

$$w_{2,3} \leftarrow 1,5 + 0,9 \cdot 0 \cdot 0,00358 = 1,5$$

$$w_{2,4} \leftarrow 1 + 0,9 \cdot 0 \cdot 0,000366 = 1$$

$$w_{3,5} \leftarrow (-2) + 0,9 \cdot 0,982 \cdot (-0,063) = -2,0556$$

$$w_{3,6} \leftarrow 1 + 0,9 \cdot 0,982 \cdot 0,077 = 1,068$$

$$w_{4,5} \leftarrow 3 + 0,9 \cdot 0,119 \cdot (-0,063) = 2,993$$

$$w_{4,6} \leftarrow 2,5 + 0,9 \cdot 0,119 \cdot 0,077 = 2,508$$

$$w_{5,7} \leftarrow (-2) + 0,9 \cdot 0,167 \cdot 0,227 = -1,965$$

$$w_{6,7} \leftarrow 2 + 0,9 \cdot 0,782 \cdot 0,227 = 2,159$$

propagation avant...

Cette boucle s'applique soit jusqu'à ce que l'erreur ait atteint un certain seuil ou soit jusqu'à ce qu'on ait atteint un certain nombre de boucles. Ces calculs peuvent donc être fait à la main, mais sont très longs.

3.7 optimisation avec mini-batch

Nous avons vu qu'un problème de Machine learning tel que nous l'avons vu consiste en un problème d'optimisation. En effet, le réseau de neurone peut être vu comme une fonction avec beaucoup de variables différentes dont on doit trouver le minimum à l'aide du gradient.

Cependant, l'ensemble d'entraînement D est souvent très grand dans un problème de Machine learning, ainsi calculer le gradient moyen pour tout l'ensemble d'entraînement serait beaucoup trop long dans certains cas. Ainsi, il faut nous contenter de calculer le gradient moyen à partir d'une partie de l'ensemble d'entraînement.

Chapter 3. Les réseaux de neurones artificiels (ou ANN pour artificial neural network)

Ces données sont sélectionnées de manière aléatoire est constitue un mini-batch.

Le gradient n'est dans ce cas pas une valeur exacte mais l'expérience nous montre que la différence entre prendre 100 exemples d'entraînement et en prendre 10000, ce qui demenderait 100 fois plus de temps de calcul, ne modifie pas grandement le résultat.

De plus dans beaucoup de cas les valeurs dans l'ensemble d'entraînement sont redondants, c'est-à-dire que l'on retrouvera à plusieurs reprise les mêmes données ce qui fait que l'approximation de cette technique est souvent proche de la vraie valeur du gradient.

En somme, il existe 2 manières de résoudre le problème d'optimistaion :

- soit la technique classique qui consiste à calculer le gradient pour tout l'ensemble d'entraîenemnt en prenant un grand "batch"
- soit la variante que nous avons présenté ici appelée descente du gradient stochastique plus rapide.

Si l'on décide d'utiliser la seconde méthode, la taille du "mini-batch" sera, tout comme le learning rate, l'un des paramètre qu'il faudra régler pour obtenir un meilleur algorithme.

Pour déterminer la taille du "mini-batch" il faut prendre en compte plusieurs principes:

- plus le "mini-batch" est grand plus le gradient sera précis.
- comme les gradients de chaque exemple d'entraînement sont souvent calculer simultanément, la mémoire de l'hardware peut déterminer la taille du "mini-batch"
- Il est préférable, surtout dans le cas de l'utilisation de la carte graphique pour le calcul, d'avoir un multiple de 2 voir 16.[5]

Il est important aussi de sélectionner ce mini-batch aléatoirement est donc de mélanger les données dans le cas où ils seraient ordonnées d'une quelconque façon.

A noter que la descente de gradient stochastique n'est pas la seul méthode d'optimisation, il existe par exemple

3.8 Mesure de la performance d'un modèle

Lorsqu'on veut mesurer la performance de notre modèle, on serait tenter de regarder l'erreur dans notre apprentissage sur notre ensemble d'entraînement. Cependant, se serait le confronter à des exemples dont l'algorithme connaît la réponse, pourtant lorsqu'on utilise un algorithme en général on ne connaît pas la réponse. Il est donc nécessaire d'utiliser des exemples nouveaux ne faisait pas partit de l'ensemble d'entraînement et dont la réponse n'est pas connue pour mesurer la performance d'un modèle, c'est-à-dire mesurer la capacité de notre algorithme à fonctionner dans des exemples nouveaux(appelé aussi généralisation).[5], [9]

On va donc séparer nos données pour en avoir une partie appelée ensemble de test permettant de mesurer la performance du modèle.

De plus, la courbe d'erreur obtenue avec l'ensemble d'entraînement n'est pas toujours la même que la courbe de l'erreur lors de l'apprentissage. Effectivement, lorsqu'on augmente le nombre d'itérations l'erreur forcément va diminuer. Cependant, ça ne veut pas dire que le modèle devient meilleur pour autant ! car la diminution de l'erreur peut provenir du fait qu'il garde en mémoire l'ensemble entraînement dont il connaît la réponse ou s'adapte spécifiquement à ces cas précis et ses particularités pour réduire l'erreur(l'augmentation du nombre de neurone augmente la mémoire ce qui va aussi produire ce phénomène)

Comme on peut le comprendre augmenter le nombre d'itérations ou de neurones à ce niveau ne le rendra pas meilleur dans ce cas : c'est ce qu'on appelle le surapprentissage(ou overfitting en anglais).

On peut repérer ce cas en comparant la courbe d'erreur sur l'ensemble d'entraînement et celle sur l'ensemble de test : il y a surapprentissage lorsque la courbe d'erreur d'apprentissage diminue alors que l'erreur sur la courbe d'erreur sur l'ensemble test augmente.

Il faut donc augmenter le nombre d'itérations et de neurones tant que les 2 courbes diminuent jusqu'à atteindre un point de la fonction entre le surapprentissage et un point où on peut encore augmenter ces paramètre, car le

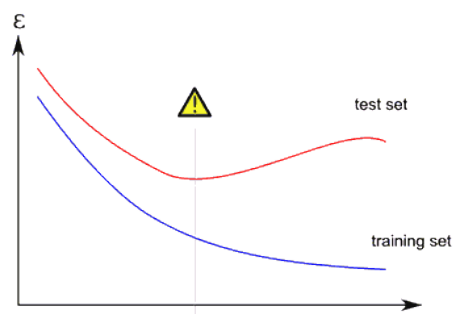


Figure 3.2: représentation du sur et sous-apprentissage

réseau est en sous-apprentissage.

Une des techniques pour éviter l'overfitting est le "dropout", cette technique consiste à ignorer aléatoirement une partie des neurones à une certaine couche lors de l'apprentissage, ce qui permet de réduire la capacité du réseau à garder en mémoire les exemples. En effet, dans un réseau de neurone de type "feed forward" les neurones sont connectés à beaucoup d'autres et les neurones peuvent devenir dépendent les uns des autres et ne plus fonctionner individuellement lorsque le nombre d'epochs est grand.[2] ⁴

On remarque, sur le schéma ci-dessous (figure 3.3), que le nombre de lien est bien plus petit.

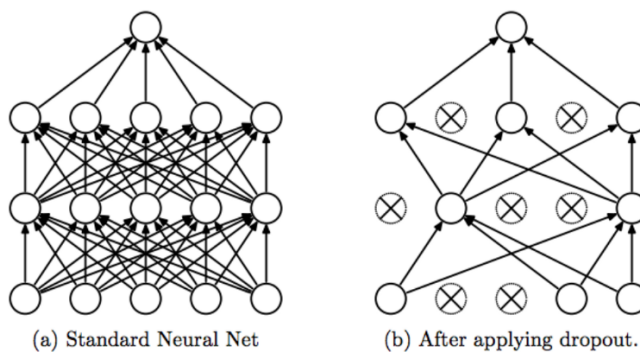


Figure 3.3: différence d'un réseau avec et sans dropout, source: <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334> consulté le 13 août 2018

En conséquence, le nombre d'epochs nécessaire est plus grand mais aussi ils sont plus rapides. Un dropout souvent utilisé est de 0.2 (on ignore 20% des neurones), mais il convient comme avec les autres hyper-paramètres d'en tester plusieurs. Par exemple, un nombre trop élevé pour le dropout peut considérablement réduire la performance d'un réseau de neurones.

Cependant, le meilleur moyen de réduire l'overfitting reste d'avoir plus de data, c'est-à-dire plus d'exemples différents et variés. L'overfitting provenant de l'accoutumance du réseau à des exemples spécifiques mais ne rendant pas le réseau plus performant sur le problème que le réseau doit traiter, l'overfitting peut provenir d'un manque d'exemple et surtout de variété. Ce dernier peut être commencé par une technique appelée data augmentation⁵ qui consiste à transformer les données dans on dispose en les transformant légèrement (utiliser pour les CNN cf. chapitre 10)

⁴cf. <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334>

⁵sources : <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

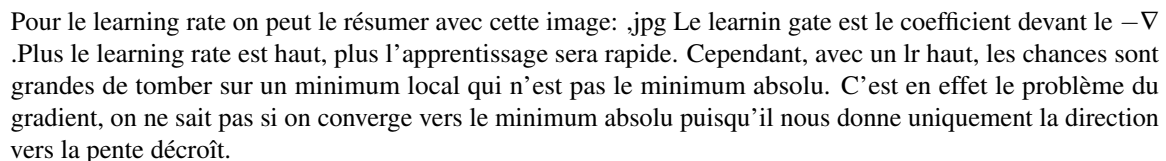
Chapter 3. Les réseaux de neurones artificiels (ou ANN pour artificial neural network)

3.9 les hyper-paramètres

Les hyper-paramètres sont les paramètres définis avant l'apprentissage (selon wikipedia)

Ce sont donc le nombre d'itérations, le learning rate, le biais, les fonctions d'activation ou encore le nombre de couche ou neurone par couche. Choisir les bonnes valeurs pour ses paramètres et les adapter au mesure de l'apprentissage permet de l'améliorer. Ils se différencient d'autres paramètres comme les poids des connections par le fait que ce soit l'utilisateur et non l'algorithme qui décide de leur valeur.

Il faut donc respecter certains principes comme ne pas se baser sur l'erreur lors de l'apprentissage, c'est-à-dire se contenter d'augmenter le nombre d'itérations pour réduire l'erreur ce qui mènerait à du sur-apprentissage. Tout d'abord il faut faire une liste des différents hyper-paramètres à tester (varier le nombre d'itérations, activations,...) puis il faut mesurer la performance de notre modèle sur un ensemble dit de validation pour déterminer le meilleur choix pour les hyper-paramètres. En effet, généralement nos données en environ 80% pour l'apprentissage et 10% respectivement pour la validation (mesure de performance) et le test pour mesurer la performance d'un modèle comme le perceptron ou un réseau de neurones sur une application. [9]

Pour le learning rate on peut le résumer avec cette image:  Le learning rate est le coefficient devant le $-\nabla$. Plus le learning rate est haut, plus l'apprentissage sera rapide. Cependant, avec un lr haut, les chances sont grandes de tomber sur un minimum local qui n'est pas le minimum absolu. C'est en effet le problème du gradient, on ne sait pas si on converge vers le minimum absolu puisqu'il nous donne uniquement la direction vers la pente décroît.

Ainsi, une des techniques utilisée consiste à diminuer le learning rate dans l'apprentissage si l'erreur augmente au lieu de descendre (ce qui veut dire qu'on est tombé sur le mauvais minimum).

voir vidéo HUGO Larcohelle et pdf

3.10 les types d'optimisation

La descente de gradient stochastique n'est pas la seule méthode d'optimisation. Il en existe encore d'autres plus complexes, mais plus performantes et moins problématiques. En effet, contrairement à la descente de gradient stochastique, le taux d'apprentissage va s'adapter au cours de l'apprentissage.

L'algorithme AdaGrad est une amélioration du SGD. Elle adapte les taux d'apprentissage de chaque paramètre individuellement en les rendant inversement proportionnels à la racine de la somme de la valeur au carré de tous les anciens gradients.

La mise à jour d'un paramètre se fait ainsi:

Adam

Adadelta

RMSprop : elle est une amélioration de l'adagrad et est une optimisation très efficace. <https://keras.io/optimizers/> cf <https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6-parapapraphe-2.3>

3.11 régularisation

3.12 data augmentation et overfitting

Définition de overfitting:

Une des techniques d'optimisation pour éviter l'overfitting est le "dropout", cette technique consiste à ignorer aléatoirement une partie des neurones à une certaine pendant le training

3.13 Fonctions d'activation

Soft max, ReLu: regarder vidéo de l'indien fou

articles: <https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions->

Comme nous l'avons vu au chapitre 3.9, la fonction d'activation fait parti des hyper-paramètres. Ces fonctions ont des propriétés différentes les unes des autres et il convient donc de les connaître pour les appliquer aux problèmes appropriés.

Tout d'abord, il faut rappeler que les fonctions d'activations permettent au réseau de neurones de traiter des problèmes non-linéaires(cf. chapitre 3.1). En effet, la fonction d'activation que nous avons pour le perceptron à une couche est linéaire, car une fonction d'activation linéaire limite le réseau à ne pouvoir utiliser qu'une seule couche.

Donc les fonctions d'activation pour un réseau à plusieurs couches doivent être non-linéaire. De plus, il faut qu'elle soit dérivable en tout point, puisque la backpropagation utilise la dérivée de la fonction d'activation.

Il existe beaucoup de fonction d'activation, mais les plus communes sont le sigmoid que nous avons déjà présenté, la tangente hyperbolique et ReLU(pour rectified linear unit)

sigmoid était la fonction la plus utilisé aux débuts du machine learning, car son fonctionnement ressemblant aux neurones d'un cerveau est simple: la fonction renvoie un nombre entre 0 et 1 représentant le degrés d'activation d'un neurone biologique.

Cependant, elle a des problèmes: tout d'abord, lorsque la valeur d'un neurone s'approche de 1, le gradient devient très proche de zéro(comme on l'observe dans l'exemple 3.6) ce qui peut poser des problèmes dans certains cas, car le gradient peut disparaître(la fonction est saturée).

De plus, la fonction sigmoid n'est pas centré autour de zéro et peut ainsi causer des problèmes d'optimisation. En effet, avec le sigmoid, les gradients du poids sont soit tous positifs soit tous négatifs et la mise à jour peut parfois aller trop loin dans une direction ce qui ralentit l'optimisation.

Pour ces problèmes, on préfère utiliser parfois d'autre fonction d'activation. On peut cependant l'utiliser pour la dernière couche. La tangente hyperbolique les images de cette fonction sont comprises entre 0 et 1, elle est donc centré sur 0. La fonction est :

$$f(x) = \frac{2}{1 + \exp(-2x)} - 1$$

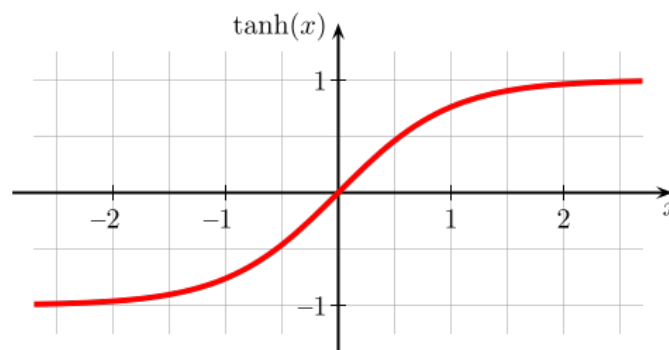


Figure 3.4: représentation de la fonction tangente hyperbolique

6

Elle a toutefois aussi le problème de disparition des gradients,néanmoins, la tangente hyperbolique devrait être utilisée au lieu du gradient.

Chapter 3. Les réseaux de neurones artificiels (ou ANN pour artificial neural network)

La fonction ReLU est devenu plus utilisé depuis ces dernières années. Elle s'écrit:

$$f(x) = \max(0, x) \text{ ou } \begin{array}{lcl} AB & = & 192 \\ C & = & 5896 \\ DEF & = & 0,5 \end{array}$$

L'image est le plus grand de 0 et de X : si $X > 0$: $f(x) = x$ et si $x < 0$: $f(x) = 0$.

représentation:

La fonction est donc linéaire à partir de 0 non-inclus. Cette propriété d'être différente à gauche et à droite l'approche de l'activation du perceptron et rend le réseau plus rapide par la simplicité de la fonction. Aussi, elle n'a pas le problème de disparition du gradient. Elle a tout de même le problème de pouvoir rendre des neurones inactifs : si le gradient d'un neurone avec ReLU est trop grand, lors de la mise à jour des poids, il se peut que ce neurone ne s'active plus du tout et le gradient avec ce neurone sera toujours de 0. Pour cela, il existe une correction de cette fonction appelé leaky ReLU qui à une petite pente dans les négatifs ce qui fait que les poids vont continuer à se corriger, car le gradient devient un petit nombre positif. La fonction. Elle se note :

$$f(x) = \begin{array}{ll} x & \text{si } x > 0 \\ 0,001x & \text{si } x < 0 \end{array}$$

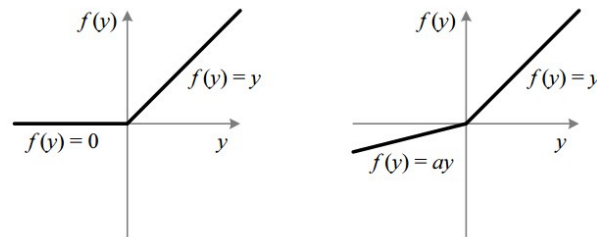


Figure 3.5: représentation de la fonction ReLU à droite et leaky ReLU à gauche

(maxout permet aussi de résoudre ce problème)

C'est donc la fonction qu'on préfère utilisé au lieu du sigmoid ou de la tangente hyperbolique pour les couches cachées, mais pour la couche de sortie⁷, on utilise la fonction softmax pour un problème de classification (qui donne une probabilité d'appartenir à une classe comme le sigmoid⁸) et une fonction linéaire pour la régression puisque on ne veut pas que la valeur change.

3.14 les fonctions de coût ou pertes

3.15 pandas ou comment manipuler la data (les données)

3.16 tensorflow et keras

Tensorflow est un outil développé par google brain (section s'occupant des recherches en deep learning) pour le machine learning. Il est le framework⁹ le plus utilisé dans ce domaine. A noter qu'il en existe d'autres comme pytorch ou encore theano.

⁷la couche de sortie correspond à la dernière couche, c'est-à-dire la sortie

⁸à noter que la fonction sigmoid peut être utilisable pour la classification en dernière couche

⁹définition selon wikipédia : "un framework désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture).

Chapter 3. Les réseaux de neurones artificiels (ou ANN pour artificial neural network)

L'avantage de ces librairies est leur rapidité puisque, bien que fonctionnant avec python, ils sont codé en C++ qui est un langage plus rapide(car plus proche du langage de l'ordinateur) et de plus il suffit de spécifier la structure de notre modèle et les différents paramètres au lieu d'écrire toute les formules vues ci-dessus.

Tensorflow contient aussi des outils permettant d'entraîner notre réseau sur un ensemble d'entraînement et ensuite d'évaluer ses performances.

Au lieu des matrices, on utilise une classe appelé tensor.

faire un MLP avec tensorflow:

keras:

Keras est un outil utilisant tensorflow permettant de définir plus facilement un réseau de neurone couche par couche. Il n'a pas autant de possibilités que avec tensorflow mais est beaucoup plus proche d'une programmation intuitive comme avec python.

exemple de MLP avec keras, cf cahier

Pour créer un réseau de neurone avec Keras et l'entraîner¹⁰, il y a 4 étapes fondamentales: la définition des couches, la définition de fonction d'erreur et la méthode d'optimisation(éventuellement le taux d'apprentissage), l'entraînement et l'évaluation.

```
Ajouter une couche: model.add(Dense(outputmmatrix,activation=,inputshape=()))model.compile(loss=,optimizer=)model.fit(x,yt11,batchsize=,epochs=verbose,callbacks=,validation_split=,validation_data=)model.evaluate(xtest,ytrain,...).
```

```
#Sparerladata : Xtrain,Xval,Ytrain,Yval = train_test_split(Xtrain,Ytrain,test_size = 0.1)
```

¹⁰les informations se trouvent sur le site <https://keras.io/>

¹¹respectivement l'ensemble d'entraînement et le résultat visé(output target)



4. deeplearning

5. prédire la météo avec le machine learning

Prédire la météo est très important dans la vie humaine. Au cours de l'histoire différentes techniques plus ou moins élaborées ont existé. Cependant, malgré la technologie actuelle, les prédictions météo ne sont pas toujours justes.

La difficulté de cette manœuvre vient du fait que la météo est un système comparable avec l'effet papillon des films de sciences fiction. En effet, un petit changement dans l'état initial influence grandement le système : on dit qu'elle a un comportement chaotique. On l'étudie justement en utilisant théorie du chaos.

Cependant, un réseau de neurones pourrait performer mieux que les méthodes habituelles(car l'utilisation de beaucoup de données peut révéler des résultats parfois étonnants)ou sinon mettre en évidence la propriété chaotique de la météo.[18]

Ce modèle utilise la régression, c'est-à-dire un réseau qui prend un ensemble de données en entrée et renvoie un nombre en sortie. C'est le type de réseau utilisé pour faire des prédictions, par exemple, pour prédire le marché boursier.

Le but est d'utiliser les données de température, de précipitation, d'humidité,... des jours précédents, pris grâce à l'API du site météo wunderground, pour prédire la température moyenne d'un jour donné.

[15]

Le code se trouve en annexe.

Résultats:

```
prediction = model.predict(X_val, steps = 1)
mean_error = abs((y_val - prediction))
print("mean absolute error: ")
print(mean_error.mean())
```

```
mean absolute error:
1.4387102
```

error.JPG error.JPG

6. Les CNN(convolutional neural network)

6.1 principes

Un CNN est un type de réseau de neurone utilisé pour la reconnaissance d'image, mais son utilisation peut être large comme apprendre à jouer à jeu par la reconnaissance de patterns ou même reconnaître des sons en utilisant l'intensité des fréquences de sons à travers le temps(ce qui forme un quadrillage comme pour les images) ainsi que pour du texte¹. Ils sont construits de plusieurs couches spécifiques et les neurones représentent des parties distinctives² d'une image que le réseau a détecté comme des yeux pour un visage et des roues pour une voiture. Pour distinguer une roue, le même principe s'applique : l'algorithme recherche des éléments qui reviennent comme une boucle, un rond, un trait. En définitif, il décompose l'image en différentes parties. Ce processus permet de désigner comme roue plusieurs images de roues différentes.

6.2 convolution

La première étape est de transformer les images en une matrice à 2 dimensions contenant les niveaux de gris des pixels ce qui permet de transformer le problème en un problème mathématique qu'un ordinateur peut comprendre. L'image devient un ensemble de carrés(pixels) ayant chacun un nombre correspondant à leur niveau de gris³ par exemple un pixel blanc aura la valeur 1 et un pixel noir 0.

Alors ce qu'on a désigné plus haut comme "bout d'image" appelé "features" sont en fait une combinaison de pixel de valeur similaire comme une diagonale, un trait ou une boucle par exemple. Le but pour le réseau est de trouver ces features et ensuite de les utiliser pour mettre en lien des images. Le processus permettant de retrouver une feature sur une autre image s'appelle mathématiquement le "filtering". Il se compose de 4 étapes : tout d'abord aligner une feature avec une partie de l'image à analyser, ensuite multiplier la valeur des pixels de ce dernier avec ceux de la feature, les additionner et diviser par le nombre total dans la feature.

Sur l'exemple ci- dessus le calcul donne:

$$\frac{1 + 1 - 1 + 1 + 1 + 1 - 1 + 1 + 1}{9} = 0,55$$

¹on peut utiliser ces réseaux pour tout types de données transformable en une sorte d'image avec des pattern et des features

²ces parties distinctifs sont mathématiquement une combinaisons de pixels qui reviennent d'une image à l'autre

³le niveau de gris représente la luminosité d'un pixel https://fr.wikipedia.org/wiki/Niveau_de_gris

Chapter 6. Les CNN(convolutional neural network)

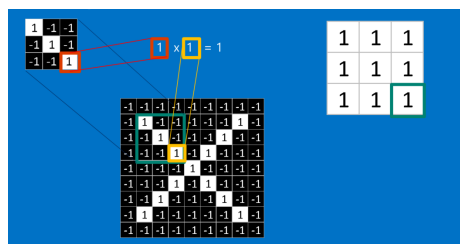


Figure 6.1: exemple de filtering source:http://brohrer.github.io/how_convolutional_neural_networks_work.html

Cette valeur montre à quel point la feature est le bout d'image correspondant. L'algorithme va bouger le filtre à travers l'image et répéter ces étapes pour retrouver la feature sur l'image en question ce qui va donner au finale une grille avec différentes valeurs permettant de voir si on retrouve une forme sur l'image.

Ce processus de bouger le filtre pour retrouver les features s'appelle convolution. Cette grille montre où la

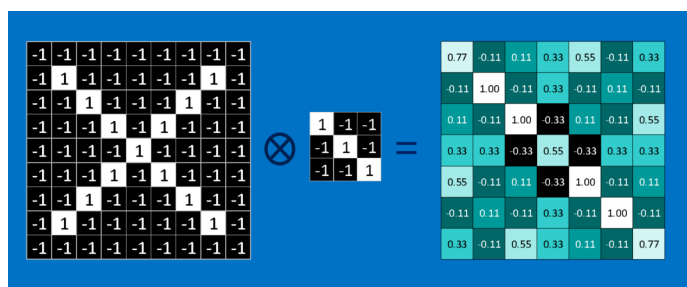


Figure 6.2: exemple de convolution source:http://brohrer.github.io/how_convolutional_neural_networks_work.html

feature apparaît. En effet, les valeurs sont élevés où la diagonale apparaît.

Le but d'une couche de convolution est de répéter cette convolution avec les différents features pour obtenir différentes images filtrées. C'est la transformation d'une image en une pile d'images filtrées. Ce type de couche est le premier élément essentiel d'un CNN.

6.3 couche de pooling

Le 2ème type de couche spécifique aux réseaux neuronaux convolutifs est la pooling layer, elle suit directement la convolution. Le but est de diminuer la pile d'image créée pour chaque features. Pour ce faire il faut prendre un cadre de 2x2 ou 3x3 et un nombre de case à avancer(normalement 2). Le but est de déplacer le cadre de 2 pixels et à chaque fois de répertorier sur une 2ème grille plus petite la valeur la plus grande dans notre. En se déplaçant ainsi à travers la première grille, on obtient une 2ème grille plus petite, mais on y retrouve toujours les features avec les hautes valeurs.

Avec cette opération, la taille de l'image diminue par 2, ce qui est nécessaire car les images ont souvent plusieurs milliers de pixels en taille. De plus, le pooling permet de retrouver toutes les features, car le cadre sélectionne la valeur la plus haute, ce qui fait que la position ne sera pas un problème.

La 3ème technique utilisée pour ces réseaux est la normalisation. Comme lorsqu'on norme un vecteur, on change les valeurs pour qu'elles soient toutes comprises entre 0 et 1. Pour cela, on passe les valeurs des pixels à travers la fonction ReLU.⁴ Avec cette fonction, les nombres négatifs vont se transformer en zéros.

L'ordre des opérations d'un CNN est donc une couche convolutif avec ReLU et le pooling à travers lesquels on

⁴ chapitre sur les fonctions d'activation

Chapter 6. Les CNN(convolutional neural network)

pas une matrice. Cependant, on peut faire des réseaux plus grands(deep learning) et répéter les différentes étapes. C'est ce qu'on appelle le deep stacking: on peut par exemple effectuer 2 convolutions de suite, puis un pooling, puis à nouveau une convolutions etc... Après ces modifications, notre pile d'image est beaucoup plus

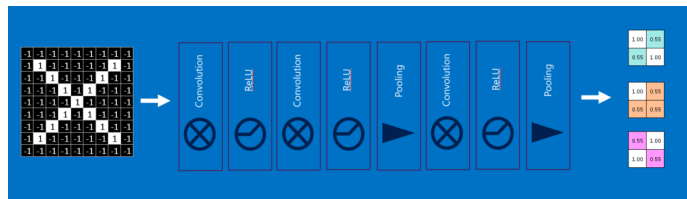


Figure 6.3: exemple de deep stacking source:http://brohrer.github.io/how_convolutional_neural_networks_work.html

petite ce qui fait que les features sont de plus en plus nombreuses et précise.

6.4 dernières couches d'un CNN

Enfin, la dernière couche d'un CNN est appelée une couche entièrement connectée⁵. A cette étape, on retrouve le même principe que les liaisons des réseaux de neurones vu jusqu'à présent. La sortie du réseau est une image à reconnaître comme des chiffres et les processus d'avant nous ont permis de séparer les images à analyser en éléments. Dans cette couche, les poids des connections d'une image et d'une features de cette image sont plus élevés; c'est la fameuse règle de Hebb: « des neurones qui s'excitent ensemble se lient entre eux. » (« cells that fire together, wire together »).

On peut symboliser cette opération par un vote: chaque valeur de notre pile de grille vote par rapport à l'image finale qui leur correspond et les votes(connections) ont plus de poids si une valeur prédit telle ou telle image. Par exemple, une connexion reliant une feature en courbe et le chiffre 8 a plus de poids que cette features relié au chiffre 1.

Ainsi, les valeurs des neurones par rapport aux poids des connections, permettent de prédire, de faire une hypothèse⁶ comme pour les perceptron et l'écart entre ce qui est prédit et ce qui est ciblé, à savoir la vraie image correspondant à l'entrée, nous permet de corriger les paramètres avec la backpropagation.

Ainsi, à partir de cette étape, le réseau fonctionne de la même façon que les réseaux vus ci-dessus donc on peut tout à fait avoir plusieurs couches avec des couches cachées après les couches convolutifs et de pooling.

6.5 les hyper-paramètres d'un CNN

On retrouve les mêmes hyper-paramètres que ceux vu précédemment(activation, biais, nombre de couches cachées,...), mais il y en a aussi de nouvelles avec l'ajout des couches convolutifs et de pooling.

Avec ces couches, il faut prendre en compte le nombre de features, la taille des features pour la convolution et la taille du cadre et le nombre de pixels auquel avance le cadre.

sources: <https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6>
<https://www.youtube.com/watch?v=FmpDIaiMIeA>

6.6 exemple de reconnaissance avec cifar10

L'ensemble d'images cifar-10 est très utilisé pour tester des réseaux CNN. Il regroupe 60'000 images 32x32 en couleur; parmi celles-ci, il y a respectivement 6'000 images d'avions, voitures, oiseaux, chats, cerfs, chiens,

⁵ fully connected layer en anglais

⁶ cette hypothèse peut être vue comme une probabilité d'appartenir à une classe comme pour les perceptrons

Chapter 6. Les CNN(convolutional neural network)

grenouille, chevaux, bateaux et camions⁷.

Le réseau utilisé est un deep CNN avec 3x[2x(Conv2D+relu)+pooling] ce qui permet d'obtenir 85% accuracy.

```
Epoch 93/100
- 50s - loss: 0.5498 - acc: 0.8347 - val_loss: 0.5372 - val_acc: 0.8507
Epoch 94/100
- 48s - loss: 0.5497 - acc: 0.8360 - val_loss: 0.5428 - val_acc: 0.8473
Epoch 95/100
- 41s - loss: 0.5420 - acc: 0.8380 - val_loss: 0.5420 - val_acc: 0.8483
Epoch 96/100
- 41s - loss: 0.5461 - acc: 0.8354 - val_loss: 0.5546 - val_acc: 0.8470
Epoch 97/100
- 41s - loss: 0.5459 - acc: 0.8352 - val_loss: 0.5491 - val_acc: 0.8463
Epoch 98/100
- 41s - loss: 0.5464 - acc: 0.8356 - val_loss: 0.5365 - val_acc: 0.8510
Epoch 99/100
- 41s - loss: 0.5495 - acc: 0.8348 - val_loss: 0.5358 - val_acc: 0.8503
Epoch 100/100
- 41s - loss: 0.5460 - acc: 0.8365 - val_loss: 0.5477 - val_acc: 0.8477
```

Figure 6.4: résultat du CNN

Certains réseau arrive à obtenir 95%+ d'accuracy

⁷à noter qu'il existe une version avec 100 type d'images(cifar-100)



7. MI et marketing,journal,...

Sources

- [1] 3Blue1Brown. *But what *is* a Neural Network? | Deep learning* (cited on page 5).
- [2] Amar Budhiraja. *Dropout in (Deep) Machine learning*. <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5>. [consulté en juillet 2018] (cited on page 16).
- [3] Andrew Fogg. *A History of Deep Learning*. <https://www.import.io/post/history-of-deep-learning/>. [consulté en juillet 2018] (cited on page 4).
- [4] giant_nneural_nnetwork. *Beginner Intro to Neural Networks* (cited on page 5).
- [5] Ian Goodfellow and Yoshua Bengio et Aaron Courville. *Deep Learning*. [consulté en mai 2018]. MIT Press, 2016 (cited on page 15).
- [6] Hugo Larochelle. *Intelligence Artificielle [12.10] : Apprentissage automatique - régression logistique*. <https://www.youtube.com/watch?v=af1j0ogiUak&index=79&list=PL6Xpj9I5qXYGhsvMWM53ZLfwUInzvYWsm>. [consulté en février 2018] (cited on page 10).
- [7] Hugo Larochelle. *Intelligence Artificielle [12.11] : Apprentissage automatique - réseau de neurones*. <https://www.youtube.com/watch?v=ngTF9NxEx7c&list=PL6Xpj9I5qXYGhsvMWM53ZLfwUInzvYWsm&index=80>. [consulté en mars 2018] (cited on page 11).
- [8] Hugo Larochelle. *Intelligence Artificielle [12.12] : Apprentissage automatique - rétropropagation*. <https://www.youtube.com/watch?v=NgwvhX0xBs0&index=81&list=PL6Xpj9I5qXYGhsvMWM53ZLfwUInzvYWsm>. [consulté en avril 2018] (cited on page 12).
- [9] Hugo Larochelle. *Intelligence Artificielle [12.14] : Apprentissage automatique - généralisation*. <https://www.youtube.com/watch?v=B90jPEZ4yuA>. [consulté en juin 2018] (cited on pages 15, 17).
- [10] Hugo Larochelle. *Intelligence Artificielle [12.4] : Apprentissage automatique - perceptron*. <https://www.youtube.com/watch?v=L7WNYvbvGBc&index=73&list=PL6Xpj9I5qXYGhsvMWM53ZLfwUInzvYWsm>. [consulté en février 2018] (cited on page 5).
- [11] Hugo Larochelle. *Intelligence Artificielle [12.5] : Apprentissage automatique - exemple du perceptron*. <https://www.youtube.com/watch?v=nZjg8DjrWVo&index=74&list=PL6Xpj9I5qXYGhsvMWM53ZLfwUInzvYWsm>. [consulté en février 2018] (cited on page 7).
- [12] Hugo Larochelle. *Intelligence Artificielle [12.8] : Apprentissage automatique - dérivées partielles et gradient*. <https://www.youtube.com/watch?v=Hni08ApX95A&index=77&list=PL6Xpj9I5qXYGhsvMWM53ZLfwUInzvYWsm>. [consulté en février 2018] (cited on page 9).

SOURCES

- [13] Hugo Larochelle. *Intelligence Artificielle [12.9] : Apprentissage automatique - minimisation de perte*. <https://www.youtube.com/watch?v=diiP1ydlyss&index=78&list=PL6Xpj9I5qXYGhsvMWM53ZLfwUInzvYWsm>. [consulté en février 2018] (cited on page 9).
- [14] Encyclopédie Larousse. *définition de l'intelligence artificielle*. http://www.larousse.fr/encyclopedie/divers/intelligence_artificielle/187257. [consulté en février 2018] (cited on pages 2, 3).
- [15] Adam McQuistan. *Using Machine Learning to Predict the Weather*. <https://stackabuse.com/using-machine-learning-to-predict-the-weather-part-1/>. [consulté en juin 2018] (cited on page 22).
- [16] Andrew Ng. *Machine Learning*. <https://www.coursera.org/learn/machine-learning/>. [consulté en février 2018] (cited on page 2).
- [17] *Timeline of machine learning*. https://en.wikipedia.org/wiki/Timeline_of_machine_learning. [consulté en juillet 2018] (cited on page 4).
- [18] Wikipédia. *Théorie du chaos*. https://fr.wikipedia.org/wiki/Théorie_du_chaos. [consulté en août 2018] (cited on page 22).