Apprentissage connexionniste

Nathalie Villa-Vialaneix, Stéphane Canu

Table des matières

Chapitre 7

Apprentissage connexionniste

Nathalie Villa-Vialaneix et Stéphane Canu

Ce chapitre est une introduction à l'*apprentissage connexionniste*, qui est un type d'apprentissage inspiré des neurosciences et de la psychologie cognitive. On retrouve également ces méthodes sous le nom générique de *réseaux de neurones* ou *réseaux de neurones artificiels*. Celles-ci se sont peu à peu éloignées des sciences cognitives dont elles sont issues et ont été appropriées par les communautés de la statistique, de l'intelligence artificielle et du « machine learning ».

La famille des réseaux de neurones est très vaste et comprend des modèles très divers avec des buts très différents. Ces méthodes sont utilisées à des fins exploratoires (méthodes non supervisées comme les cartes auto-organisatrices dites cartes de Kohonen, Kohonen [1995, 2014]) ou bien pour de la prédiction. Elles possèdent le point commun d'être basées sur la combinaison de modèles très simples, généralement appelés *neurones* (ou *neurones formels*). Ces neurones possèdent plusieurs formes différentes, sont combinés de façons diverses et sont entraînés de manières différentes, ce qui donne à la famille des réseaux de neurones sa variété.

Dans ce chapitre, nous nous restreindrons à la présentation d'une des méthodes les plus courantes utilisée en apprentissage, les perceptrons multicouches. Après avoir présenté les origines de la méthode et sa définition formelle (sections 7.1 et 7.2), nous étudierons les propriétés statistiques de ces approches (section 7.3) puis nous présenterons les principes de base de la mise en œuvre de leur apprentissage (section 7.4). Ensuite, dans la section 7.5, nous illustrerons quelques-uns des principes présentés dans le chapitre sur un jeu de données simulées très simple. Enfin, une introduction à l'apprentissage profond (« *deep learning* »), qui est une utilisation récente des perceptrons à des problèmes d'apprentissage complexes et de grande taille, est présentée dans la section 7.6.

Nous renvoyons le lecteur intéressé par plus de détails sur les réseaux de neurones aux ouvrages de référence [Bishop, 1995; Ripley, 1996], qui donnent une

vision d'ensemble assez diverse des méthodes neuronales. En particulier, dans un cadre supervisé, des variantes des réseaux de neurones y sont présentées. Parmi celles-ci, on peut citer, les *réseaux à fonctions de base radiale* (« RBF networks », Powell [1987]) ou les réseaux basés sur des modèles de mélange de densités (« mixture density networks »). L'utilisation d'approches bayésiennes pour l'apprentissage de réseaux de neurones y est également abordée : ces méthodes offrent des solutions intéressantes pour le choix des hyper-paramètres ou de l'architecture ainsi que dans le domaine de l'apprentissage actif (« active learning », MacKay [1992]) pour diriger la collecte de nouvelles données au cours de l'apprentissage. Également, les chapitres dédiés aux réseaux de neurones des ouvrages [Devroye *et al.*, 1996; Györfi *et al.*, 2002] présentent des résultats assez complets sur les propriétés statistiques de ces méthodes. Enfin, Bishop [2006] consacre également un chapitre aux réseaux de neurones, sous un angle plus orienté « machine learning » en présentant divers aspects de la mise en œuvre de leur apprentissage.

7.1 Origine et définition des perceptrons

Les réseaux de neurones ont été motivés, à l'origine, par la volonté de modéliser le fonctionnement du cerveau. En particulier, le principe de base des réseaux de neurones est basé sur les découvertes issues des premières études sur le cerveau qui montrèrent que le système nerveux central est composé de cellules aux propriétés remarquables, les *neurones*. Chacun de ces neurones a un fonctionnement très basique mais l'architecture complexe de leur organisation leur permet de traiter des tâches complexes. Un neurone est constitué de trois parties : le noyau, à l'intérieur du corps cellulaire, qui est le centre vital du neurone, les dendrites, qui reçoivent les informations provenant d'autres neurones par l'intermédiaire de flux électriques excitateurs ou inhibiteurs. Ces influx électriques entrant sont traités par le neurone et si le signal total reçu est considéré comme suffisamment fort, le neurone déclenche, via son axone, une réponse adéquate sous la forme d'un flux électrique sortant. L'axone possède jusqu'à un millier de ramifications, les synapses, qui permettent la connexion avec les dendrites des autres neurones.

Une des caractéristiques remarquables des neurones est leur capacité d'adaptation et de mémorisation. En effet, les connexions évoluent constamment : certaines synapses dégénèrent, d'autres se stabilisent. Au milieu du $20^{\text{ème}}$ siècle, Hebb [1949] fait l'hypothèse que l'évolution des connexions est soumise à la règle suivante :

> « *Quand une cellule A excite, par son axone, une cellule B et que, de manière répétée et persistante, elle participe à la génèse d'une impulsion dans B, un processus de croissance ou un changement métabolique a lieu dans l'une ou dans les deux cellules de telle sorte que l'efficacité de A à déclencher une impulsion dans B est, parmi*

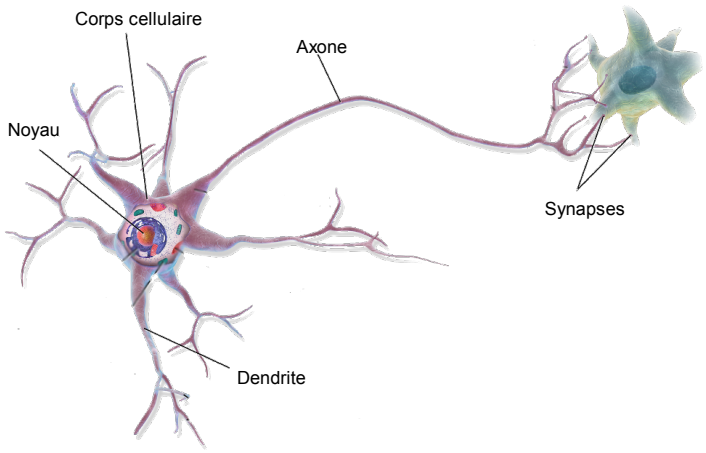

Figure 7.1 : *Représentation schématique d'un neurone biologique. Image adaptée d'un travail de BruceBlaus, Source : Wikimedia Commons, « Blause_0657_MultipolarNeuron.png ».*

les autres cellules qui ont cet effet, accrue. »

La modélisation de ce principe de base a conduit à la conception de neurones artificiels. Le premier modèle de neurone artificiel fût introduit dans l'article de McCulloch and Pitts [1943] : il est schématisé dans la figure 7.2. Pour une valeur $\mathbf{x} = \left(x^1, \ldots, x^p\right) \in \mathbb{R}^p$, la sortie du neurone s'écrit

$$f(\mathbf{x}) = \mathbb{1}_{\left\{\sum_{j=1}^{p} w_j x^j + w_0 \geqslant 0\right\}},$$

où $-w_0 \in \mathbb{R}$ est le *seuil d'activation* du neurone (w_0 est appelé aussi le *biais*) et les $(w_j)_{j \geqslant 1}$ sont les *poids*. Le neurone est donc un simple prédicteur binaire basé sur une combinaison linéaire des variables d'entrée et sur un seuil. Dans Rosenblatt [1958, 1962], le psychologue américain Frank Rosenblatt, souvent considéré comme un précurseur du connexionnisme, utilisa ce modèle pour des tâches de reconnaissance de forme. Il lui donna le nom de *perceptron* et proposa un algorithme d'apprentissage dont il prouva la convergence. Des perspectives historiques plus complètes sont présentées dans le chapitre **??** du présent ouvrage.

7.2 Perceptrons multi-couches

Le modèle du perceptron reste très limité dans sa forme initiale. En particulier, l'approche reste linéaire et ne peut résoudre même des problèmes triviaux de discrimination binaire non linéaire (comme le problème dit du « X-OR », voir

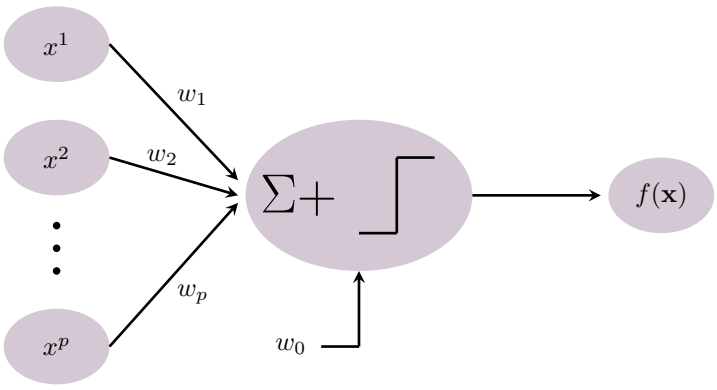

Figure 7.2 : *Neurone de McCulloch et Pitts.*

Minsky and Papert [1969] qui prouvent que le perceptron ne peut résoudre le problème de séparation des classes $\mathcal{C}_1 = \{(0,0),(1,1)\}$ et $\mathcal{C}_2 = \{(0,1),(1,0)\}$). Ces limitations ont été levées par la définition du *perceptron multi-couches* [LeCun, 1985; Rumelhart and Mc Clelland, 1986]. Celui-ci combine des neurones élémentaires sur des couches successives dont le but est de propager l'information. L'apprentissage de tels modèles est rendu possible par l'utilisation du principe de la *rétro-propagation du gradient* (voir section 7.4).

Dans la suite de la section, nous nous focaliserons sur l'étude du perceptron à une couche cachée, même si les travaux récents (datant des années 2010) montrent l'intérêt de combiner un grand nombre de couches cachées (c'est l'*apprentissage profond*, ou « deep learning », qui est discuté plus en avant dans la section 7.6).

Un perceptron à une couche cachée est schématisé par la figure 7.3. Pour une valeur $\mathbf{x} = \left(x^1, \ldots, x^p\right) \in \mathbb{R}^p$, la sortie du perceptron s'écrit

$$f(\mathbf{x}) = h_0\left(\sum_{l=1}^{Q} w_l^{(2)} h_l\left(\mathbf{x}^\top \mathbf{w}_l^{(1)} + w_l^{(0)}\right) + w_0^{(2)}\right), \tag{7.1}$$

avec Q le nombre de neurones sur la couche cachée et $\forall\, l = 1, \ldots, Q$, $w_l^{(0)}, w_l^{(2)} \in \mathbb{R}$ et $\mathbf{w}_l^{(1)} \in \mathbb{R}^p$. h_l $(l = 0, \ldots, Q)$ sont les *fonctions de transfert* ou *fonctions d'activation*, qui peuvent prendre diverses formes (voir figure 7.4) :

- identité : $z \in \mathbb{R} \mapsto z$;
- fonction de transfert à seuil : $z \in \mathbb{R} \mapsto \mathbb{1}_{\{z \geqslant 0\}}$;
- logistique sigmoïde : $z \in \mathbb{R} \mapsto \frac{1}{1+\exp(-z)}$;

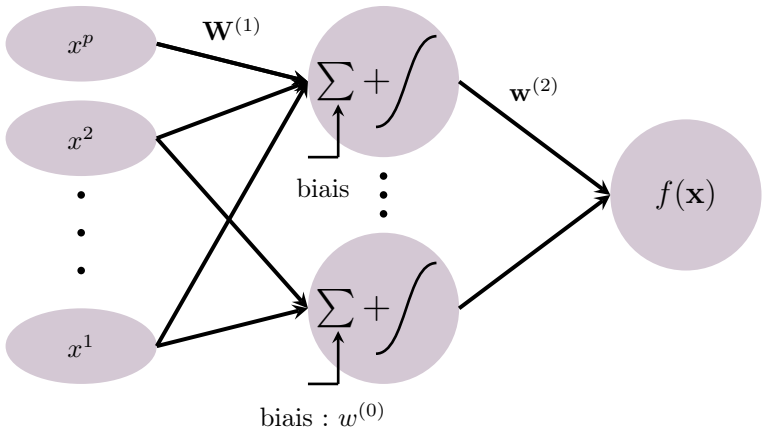

Figure 7.3 : *Perceptron à une couche cachée.*

- identité rectifiée (ReLu) : $z \in \mathbb{R} \mapsto \max(0,z)$;
- ...

En pratique, la logistique sigmoïde est le choix standard pour h_l avec $l \geqslant 1$ et le choix pour h_0 dépend de la nature du problème : pour les problèmes de régression, h_0 est souvent la fonction identité alors que pour les problèmes de classification, les fonctions privilégiées sont la fonction de transfert à seuil et la logistique sigmoïde qui permet d'obtenir des sorties semblables à celles d'une régression logistique (en particulier, dans ce cas, $f(\mathbf{x}) \in [0,1]$).

De nombreux autres choix de fonctions de transfert existent, que l'on regroupe généralement sous le terme générique de *sigmoïde*. Dans la suite, on appellera sigmoïde une fonction $h : \mathbb{R} \to \mathbb{R}$ telle que h est croissante et $\lim_{z\to+\infty} h(z) = 1$, $\lim_{z\to-\infty} h(z) = 0$ (en particulier, la fonction identité n'est donc pas une sigmoïde).

7.3 Propriétés statistiques

Cette section présente quelques propriétés statistiques des perceptrons multicouches. En particulier, nous nous intéressons à deux propriétés fondamentales :

- la propriété d'*approximation universelle* qui donne une propriété de den-

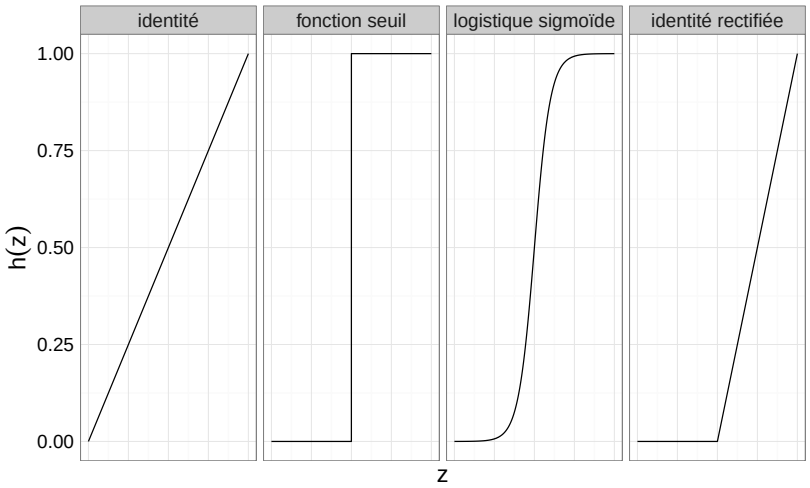

Figure 7.4 : *Quelques exemples de fonctions de transfert classiques.*

sité des perceptrons à une couche cachée dans l'ensemble des fonctions régulières ;

- les propriété de *consistance* des perceptrons à une couche cachée que nous décrirons dans le cadre de la discrimination binaire pour simplifier notre propos.

7.3.1 Approximation universelle

De nombreux travaux ont traité de la richesse d'approximation de la famille des fonctions de la forme de l'équation (7.1). On renvoie le lecteur intéressé à Pinkus [1999] pour une revue des travaux dans ce domaine. En particulier, Hornik [1991] donne un résultat de densité des réseaux de neurones sur les compacts de l'ensemble des fonctions continues de $\mathbb{R}^p$ dans $\mathbb{R}$ et Hornik [1993] généralise ce résultat aux fonctions de $L_r(\mu)$ où μ est une mesure finie sur $\mathbb{R}^p$. Stinchcombe [1999] donne une version de ces résultats pour des espaces vectoriels arbitraires.

De manière formelle, on peut formuler la propriété d'approximation universelle en introduisant

$$\mathcal{P}^Q(h) = \left\{ \mathbf{x} \in \mathbb{R}^p \mapsto \sum_{l=1}^{Q} w_l^{(2)} h\left(\mathbf{x}^\top \mathbf{w}_l^{(1)} + w_l^{(0)}\right) + w_0^{(2)} : w_l^{(2)}, w_l^{(0)} \in \mathbb{R}, \mathbf{w}_l^{(1)} \in \mathbb{R}^p \right\}$$

et $\mathcal{P}(h)$ l'espace engendré par $\bigcup_{Q\in\mathbb{N}} \mathcal{P}^Q(h)$. On a alors le résultat suivant :

Théorème 7.1 (Pinkus [1999]) *Si h est une fonction continue alors $\mathcal{P}(h)$ est dense dans l'ensemble des fonctions continues de $\mathbb{R}^p$ dans $\mathbb{R}$ pour la topologie de la convergence uniforme sur les compacts si et seulement si h n'est pas un polynôme.*

Cela signifie que, si h est une fonction continue non polynomiale alors, pour toute fonction continue $g : [0,1]^p \to \mathbb{R}$ et pour tout $\epsilon > 0$, il existe une fonction $f \in \mathcal{P}(h)$ telle que :

$$\sup_{\mathbf{x}\in[0,1]^p} |f(\mathbf{x}) - g(\mathbf{x})| < \epsilon.$$

Des versions différentes de ce théorème existent pour des fonctions h qui sont des sigmoïdes (non nécessairement continues) : par exemple, Devroye *et al.* [1996] donnent un résultat similaire pour $\mathcal{P}(h)$ avec h une sigmoïde quelconque.

Enfin, comme souligné par Pinkus [1999], il faut ajouter que l'on ne peut approcher de la même manière les fonctions continues avec l'ensemble $\mathcal{P}^Q(h)$ pour un Q donné. Au contraire, des bornes inférieures d'approximation existent, indépendantes du choix de h, pour la qualité de l'approximation fournie par $\mathcal{P}^Q(h)$.

7.3.2 Consistance

La propriété d'approximation universelle est une propriété importante pour prouver la consistance (dans le sens introduit dans le chapitre **??** ; sauf mention contraire, les notations seront reprises de ce chapitre).

Pour simplifier le propos, nous nous placerons dans le cadre de la classification binaire. De manière plus précise, nous supposerons connu un échantillon d'apprentissage $D_n = (X_i,Y_i)_{1\leqslant i\leqslant n}$ qui sont des réalisations i.i.d. d'un couple de variables aléatoire (X,Y) de loi P dans lequel $X \in \mathbb{R}^p$ et $Y \in \{0,1\}$. On notera, enfin, $\mathcal{F}^Q(h)$ l'ensemble des prédicteurs

$$f : \mathbf{x} \in \mathbb{R}^p \mapsto \begin{cases} 0 & \text{si } \psi(\mathbf{x}) \leqslant 1/2, \\ 1 & \text{sinon} \end{cases} \qquad \text{pour un } \psi \in \mathcal{P}^Q(h).$$

Rappelons que, dans ce cadre, le risque associé à f est $\mathcal{R}_P(f) = \mathbb{P}(f(X) \neq Y)$. La preuve de consistance, dans ce cadre-ci, peut être trouvée dans Devroye *et al.* [1996].

Tout d'abord, la propriété d'estimation universelle des perceptrons permet de démontrer que l'erreur d'approximation de cette famille de prédicteurs est proche de l'erreur optimale :

Théorème 7.2 (Devroye *et al.* [1996]) *Si h est une sigmoïde quelconque alors, pour toute loi P*

$$\lim_{Q\to+\infty} \inf_{f\in\mathcal{F}^Q(h)} \mathcal{R}_P(f) - \mathcal{R}_P^* = 0.$$

Pour contrôler l'erreur d'estimation des perceptrons à une couche cachée, des bornes inférieures et supérieures de leur dimension de Vapnik-Chervonenkis ont été démontrées dans [Baum, 1988; Baum and Haussler, 1989] dans le cadre où h est la fonction de transfert à seuil :

Théorème 7.3 (Baum [1988]; Baum and Haussler [1989]) *Soit h la fonction de transfert à seuil alors*

$$2\left\lfloor \frac{Q}{2} \right\rfloor \leqslant V(\mathcal{F}^Q(h)) \leqslant (2Qp + 4Q + 2)\log_2(e(Qp + 2Q + 1)).$$

Notons que la borne supérieure de la dimension de Vapnik-Chervonenkis est directement liée au nombre de paramètres (poids) du réseau, $Qp + 2Q + 1$: les grandes valeurs de Q donnent des perceptrons avec une grande capacité d'approximation et donc un risque accru de sur-apprentissage.

Farago and Lugosi [1993] en déduisent la consistance universelle de ce type de perceptrons lorsque l'on minimise l'erreur empirique de classification (les résultats de la section **??** permettent également d'obtenir le même résultat) :

Théorème 7.4 (Farago and Lugosi [1993]) *Soit h la fonction de transfert à seuil et soit $\widehat{f}_{\mathcal{F}^{Q_n}(h)}$ le minimiseur de l'erreur empirique de classification $\widehat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^n \mathbb{1}_{\{f(X_i)\neq Y_i\}}$ sur l'ensemble des fonctions, f, de $\mathcal{F}^{Q_n}(h)$. Alors, si $Q_n \to +\infty$ et $(Q_n \log n)/n \xrightarrow{n\to+\infty} 0$, on a*

$$\lim_{n\to+\infty} \mathcal{R}_P\left(\widehat{f}_{\mathcal{F}^{Q_n}(h)}\right) = \mathcal{R}_P^*.$$

D'autres résultats de consistance existent pour des fonctions sigmoïdes plus générales dans le cadre de la classification binaire. Une approche, consiste à minimiser l'erreur empirique L_r ($r \geqslant 1$) plutôt que l'erreur de classification empirique, puis à utiliser le classifieur « plug-in » associé (voir section **??** ou Devroye *et al.* [1996]). Dans le cadre de la régression, des résultats de consistance variés pour l'erreur quadratique moyenne et des vitesses de convergence ont été prouvés dans [White, 1990, 1991; Barron, 1994; McCaffrey and Gallant, 1994].

7.4 Apprentissage

Jusqu'ici, nous avons présenté les perceptrons multi-couches comme un ensemble de prédicteurs non paramétriques et nous avons étudié quelques-unes de leurs propriétés statistiques. Celles-ci sont basées sur l'optimisation d'une erreur empirique (l'erreur de classification empirique dans les résultats présentés dans la section 7.3 ou l'erreur quadratique empirique dans d'autres travaux). Or, même dans le cas de l'erreur quadratique, l'apprentissage de paramètres (les poids $\mathbf{w} = \left(\mathbf{w}^{(0)},\mathbf{W}^{(1)},\mathbf{w}^{(2)}\right) \in \mathbb{R}^{Qp+2Q+1}$) optimaux est complexe car dès

que les fonctions h_k de l'équation (7.1) ne sont pas la fonction identité, l'expression de l'erreur n'est pas une fonction quadratique ou convexe des poids et il n'existe aucune solution analytique au problème de minimisation.

Dans cette section, nous nous placerons dans le cadre de la recherche des poids $\mathbf{w}$ qui minimisent l'erreur quadratique empirique

$$\widehat{\mathcal{R}}_n(\mathbf{w}) = \sum_{i=1}^{n} [f_{\mathbf{w}}(X_i) - Y_i]^2 \tag{7.2}$$

pour $f_{\mathbf{w}}$ comme dans l'équation (7.1) avec $(h_k)_{k=0,\ldots,Q}$ des fonctions sigmoïdes continues et dérivables. Ce problème est décrit pour un cadre de régression $Y \in \mathbb{R}$ ou de classification binaire $Y \in \{0,1\}$. Dans ce dernier cas, l'optimisation des poids est souvent traitée comme dans le cas de la régression, par optimisation de l'erreur quadratique empirique, en prenant pour h_0 la fonction logistique sigmoïde et en utilisant la règle de classification finale par « plug-in »

$$\widehat{f}(\mathbf{x}) = \begin{cases} 0 & \text{si } f_{\mathbf{w}}(\mathbf{x}) \leqslant 1/2, \\ 1 & \text{sinon} \end{cases} .$$

7.4.1 Apprentissage par descente de gradient

Une approche classique pour minimiser une fonction arbitraire est l'approche par descente de gradient : les poids sont initialisés à une valeur arbitraire $\mathbf{w}(0)$ et mis à jour de manière itérative dans la direction du gradient de l'erreur en $\mathbf{w}(t)$ au pas de temps t courant :

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mu(t)\nabla_{\mathbf{w}}\widehat{\mathcal{R}}_n(\mathbf{w}(t)),$$

où $\mu(t) > 0$ est connu sous le nom de *taux d'apprentissage*. La méthode peut être appliquée de deux manières différentes : soit les n observations sont utilisées pour calculer $\nabla_{\mathbf{w}}\widehat{\mathcal{R}}_n(\mathbf{w}(t))$ à chaque itération (cette approche est appelée « batch »), soit le gradient est évalué à chaque itération par une seule observation i prise au hasard dans $\{1, \ldots, n\}$ (cette approche est appelée *approche en ligne* ou *approche stochastique*). Dans ce dernier cas, on décompose l'erreur quadratique en

$$\widehat{\mathcal{R}}_n(\mathbf{w}) = \sum_{i=1}^{n} E_i(\mathbf{w}) \tag{7.3}$$

et la mise à jour à l'étape t, lors de laquelle l'observation i est tirée, devient

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mu(t)\nabla_{\mathbf{w}}E_i(\mathbf{w}(t)).$$

L'approche « batch » converge vers un minimum local (pour un bon choix du taux d'apprentissage) mais la convergence peut être assez lente. Des méthodes plus efficaces ont été proposées dans un cadre « batch » qui utilisent des dérivées d'ordres plus élevés [Gill *et al.*, 1981; Fletcher, 1987; Nocedal and Wright, 1999]. L'approche stochastique a montré son efficacité pour l'apprentissage de réseaux de neurones lorsque n est grand [LeCun *et al.*, 2001] .

7.4.2 Principe de rétro-propagation du gradient

Dans le contexte de l'apprentissage des poids optimaux du perceptron, les approches basées sur le gradient ont été particulièrement développées car il existe une méthode rapide pour obtenir celui-ci : la rétro-propagation du gradient [Rumelhart and Mc Clelland, 1986]. Celle-ci est basée sur un principe de transfert de messages (« message passing ») itératif dans lequel l'information est transmise des sorties vers les entrées. Dans la suite, nous présentons la rétro-propagation du gradient dans le cadre d'un algorithme stochastique de descente du gradient mais le même principe peut-être appliqué pour d'autres types d'algorithme basés sur des calculs de dérivées d'ordres divers et pour des architectures de réseaux de neurones plus complexe que le perceptron à une couche cachée.

Rappelons tout d'abord que le terme d'erreur lié à l'observation i de l'équation (7.3) peut s'écrire :

$$E_i(\mathbf{w}) = \left[h_0\left(\sum_{l=1}^{Q} w_l^{(2)} h_l\left(X_i^\top \mathbf{w}_l^{(1)} + w_l^{(0)} \right) + w_0^{(2)} \right) - Y_i \right]^2 .$$

On introduit alors les notations :

- $\forall\, l = 1, \ldots, Q$, $a_l^{(1)} = X_i^\top \mathbf{w}_l^{(1)} + w_l^{(0)}$ est la combinaison linéaire de la couche d'entrée obtenue dans le neurone l de la couche cachée ;
- $\forall\, l = 1, \ldots, Q$, $z_l = h_l(a_l^{(1)})$ est la sortie du neurone l de la couche cachée ;
- $a^{(2)} = \sum_{l=1}^{Q} w_l^{(2)} z_l + w_0^{(2)}$ est la combinaison linéaire obtenue dans le neurone de sortie.

Les dérivées partielles de $E_i(\mathbf{w})$ par rapport aux différents poids sont obtenues de la couche de sortie vers la couche d'entrée :

1. $\forall\, l = 1, \ldots, Q$,

$$\begin{aligned} \frac{\partial E_i}{\partial w_l^{(2)}} &= \frac{\partial E_i}{\partial a^{(2)}} \times \frac{\partial a^{(2)}}{\partial w_l^{(2)}} \\ &= \delta^{(2)} \times z_l \qquad \text{avec } \delta^{(2)} = \frac{\partial E_i}{\partial a^{(2)}}, \end{aligned} \tag{7.4}$$

où $\frac{\partial E_i}{\partial a^{(2)}} = \frac{\partial [h_0(a^{(2)}) - Y_i]^2}{\partial a^{(2)}} = 2h_0'(a^{(2)}) \times [h_0(a^{(2)}) - Y_i]$;
de même, $\frac{\partial E_i}{\partial w_0^{(2)}} = \delta^{(2)}$;

2. $\forall\, l = 1, \ldots, Q, \forall\, j = 1, \ldots, p,$

$$\begin{aligned} \frac{\partial E_i}{\partial w_{lj}^{(1)}} &= \frac{\partial E_i}{\partial a_l^{(1)}} \times \frac{\partial a_l^{(1)}}{\partial w_{lj}^{(1)}} \\ &= \delta_l^{(1)} \times X_{ij} \qquad \text{avec } \delta_l^{(1)} = \frac{\partial E_i}{\partial a_l^{(1)}}, \end{aligned} \tag{7.5}$$

où $\delta_l^{(1)}$ est obtenu à partir de $\delta^{(2)}$ par

$$\begin{aligned} \delta_l^{(1)} &= \frac{\partial E_i}{\partial a^{(2)}} \times \frac{\partial a^{(2)}}{\partial a_l^{(1)}} \\ &= \delta^{(2)} \times w_l^{(2)} h_l'(a_l^{(1)}) \ ; \end{aligned} \tag{7.6}$$

3. enfin, $\forall\, l = 1, \ldots, Q,$

$$\begin{aligned} \frac{\partial E_i}{\partial w_l^{(0)}} &= \frac{\partial E_i}{\partial a_l^{(1)}} \times \frac{\partial a_l^{(1)}}{\partial w_l^{(0)}} \\ &= \delta_l^{(1)}. \end{aligned} \tag{7.7}$$

L'algorithme de descente de gradient stochastique peut donc être mis en œuvre de la manière décrite dans l'algorithme 7.1, en alternant une étape « forward » (calcul des valeurs des différents neurones de la couche d'entrée vers la couche de sortie) et une étape « backward » (calcul des gradients partiels par rapport aux poids de la couche de sortie vers la couche d'entrée). En pratique, cet algorithme converge vers un minimum local et il peut donc être pertinent d'effectuer l'apprentissage plusieurs fois avec des initialisations différentes pour obtenir un optimum de bonne qualité.

7.4.3 L'apprentissage en pratique

En pratique, la mise en œuvre de l'apprentissage du perceptron requiert d'une part, d'initialiser les poids et, d'autre part, d'arrêter l'algorithme à un moment opportun.

Une solution classique pour l'initialisation des poids consiste à choisir des paramètres « moyens » : une fois les variables d'entrées centrées et réduites, il est courant d'initialiser les poids selon une distribution gaussienne d'écart type $\frac{1}{\sqrt{p}}$ pour les poids $w_{lj}^{(1)}$ et d'écart type $\frac{1}{\sqrt{Q}}$ pour les poids $w_l^{(2)}$ (la valeur de l'écart type est une fonction du nombre de neurones d'entrées correspondant aux poids).

L'apprentissage peut être stoppé selon divers critères qui sont liés à une borne sur le temps de calcul (dans ce cas, on fixe alors directement le nombre d'itérations T de l'algorithme), à une valeur cible de l'erreur ou bien à une

Algorithme 7.1 Descente de gradient stochastique par rétro-propagation

Initialiser les poids **Produit : $\mathbf{w} \in \mathbb{R}^{Qp+2Q+1}$**
Pour $t = 1 \to T$ **Faire**
  Bloc *Étape « forward »*
    Calculer $a_l^{(1)}$, $\forall\, l = 1, \ldots, Q$
    Calculer z_l, $\forall\, l = 1, \ldots, Q$
    Calculer $a^{(2)}$
  Fin
  Tirer une observation $i \in \{1, \ldots, n\}$
  Bloc *Étape « backward »*
    Calculer $\delta^{(2)}$
    Calculer $h_l'(a_l^{(1)})$, $\forall\, l = 1, \ldots, Q$
    Équation (7.6) : calculer $\delta_l^{(1)}$, $\forall\, l = 1, \ldots, Q$
    Équation (7.4) : calculer $\frac{\partial E_i}{\partial w_l^{(2)}}$, $\forall\, l = 1, \ldots, Q$
    Équation (7.5) : calculer $\frac{\partial E_i}{\partial w_{lj}^{(1)}}$, $\forall\, l = 1, \ldots, Q$ et $\forall\, j = 1, \ldots, p$
    Équation (7.7) : calculer $\frac{\partial E_i}{\partial w_l^{(0)}}$, $\forall\, l = 1, \ldots, Q$
  Fin
  Mise à jour des poids

$$\mathbf{w} \leftarrow \mathbf{w} - \mu(t) \nabla_{\mathbf{w}} E_i(\mathbf{w})$$

Fin Pour
Produit : $\mathbf{w} \in \mathbb{R}^{Qp+2Q+1}$

diminution de l'évolution de l'erreur, $\widehat{\mathcal{R}}_n(\mathbf{w}(t)) - \widehat{\mathcal{R}}_n(\mathbf{w}(t+1))$, au cours de l'algorithme. En pratique, le troisième critère est fréquemment utilisé, combiné avec une limite sur le nombre maximum d'itérations autorisées.

7.4.4 Éviter le sur-apprentissage

La complexité de la méthode est directement liée au nombre de neurones sur la couche cachée, Q, qui est un hyper-paramètre important à calibrer lors de l'apprentissage. Les méthodes classiques (validation croisée, comme décrite dans le chapitre **??**, ou bien des approches par bootstrap, basées sur une erreur « out-of-bag » comme présentées dans le chapitre **??**) peuvent être utilisées pour réaliser cette optimisation.

Pour une valeur de Q fixée assez grande, on peut aussi choisir de contrôler la complexité de la fonction de prédiction estimée en utilisant d'autres heuristiques :

- la technique dite de l'*arrêt prématuré* (« early stopping ») consiste à séparer les données en un jeu de données d'apprentissage (sur lequel l'ap-

prentissage des poids est effectué) et un jeu de données de validation sur lequel l'erreur de prédiction est calculée. L'algorithme d'apprentissage est alors arrêté dès que l'erreur de validation croît ;

- la méthode de régulation des poids (« weight decay ») consiste à pénaliser l'erreur quadratique empirique de l'équation (7.2) par une pénalité dépendant des poids. Dans la forme la plus simple, elle est équivalente à une regression ridge, ce qui conduit à minimiser en $\mathbf{w}$

$$\widehat{\mathcal{R}}_n(\mathbf{w}) + \lambda \mathbf{w}^\top \mathbf{w}, \tag{7.8}$$

  mais des variantes plus efficaces existent ;

- la méthode par injection de bruit consiste à modifier les vecteurs d'entrées par ajout d'un bruit aléatoire qui est différent à chaque présentation du vecteur.

7.5 Exemples

Dans ce chapitre, nous illustrons quelques-uns des concepts présentés au-dessus en utilisant le logiciel libre R[1]. L'implémentation utilisée est celle du package **nnet** [Ripley, 1996]. Cette implémentation contient plusieurs options standard, notamment la possibilité de choisir :

- le nombre de neurones sur la couche cachée,
- la valeur initiale des poids (par défaut, ils sont initialisés selon une loi uniforme dans $[-0.5,0.5]$ ou $\left[-\frac{1}{\max_i |X_{ij}|}, \frac{1}{\max_i |X_{ij}|}\right]$ si $\max_i |X_{ij}|$ est « grand »),
- le nombre maximum d'itérations T de l'algorithme d'apprentissage (par défaut celui-ci est égal à 100) ainsi que les seuils de l'erreur et de la décroissance de l'erreur pour contrôler l'arrêt de l'algorithme d'apprentissage,
- la valeur de la fonction d'activation de la couche de sortie, h_0 de l'équation (7.1), qui peut être l'identité ou bien la logistique sigmoïde,
- la valeur de la fonction d'erreur qui est, par défaut, l'erreur quadratique moyenne, mais peut, dans le cas où la fonction h_0 est la logistique sigmoïde, être choisie différemment (entropie, fonction log-linéaire, ... qui sont plus particulièrement recommandés lorsque le problème à traiter est en classification),
- le paramètre de régularisation, λ, comme dans l'équation (7.8).

[1] `https://www.R-project.org`

Les hyper-paramètres de l'algorithme (le nombre de neurones sur la couche cachée et le paramètre de régularisation λ) peuvent être calibrés en utilisant la méthode générique `tune` du package **e1071** qui permet d'optimiser un hyper-paramètre selon une approche par validation simple, validation croisée ou par bootstrap .

L'influence des hyper-paramètres sur l'apprentissage des réseaux de neurones est illustrée sur un jeu de données simulées très simple : celui-ci permet de comprendre de manière intuitive les concepts de base. Il correspond à la génération de 15 points i.i.d. selon le modèle suivant :

$$X \sim \mathcal{U}[0,1] \qquad \text{et} \qquad Y \sim \sin(2\pi X) + \epsilon \text{ où } \epsilon \sim \mathcal{N}(0,0.3). \tag{7.9}$$

Ces données sont reproduites dans la figure 7.5 ainsi que la fonction cible à estimer $f : \ x \in \mathbb{R} \mapsto \sin(2\pi x)$.

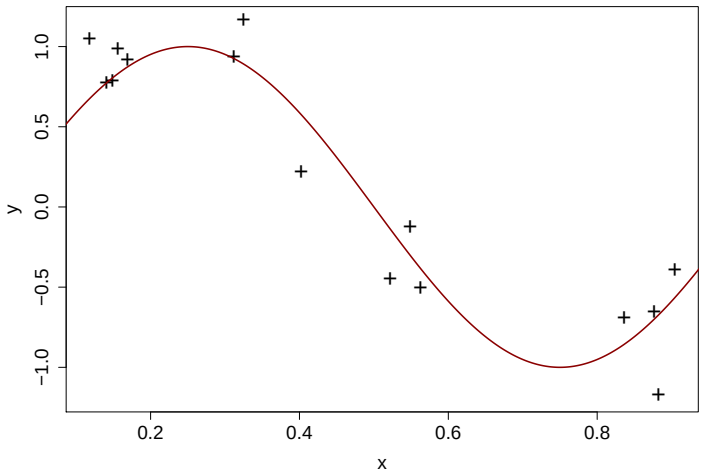

Figure 7.5 : *15 observations i.i.d. générées selon le modèle de l'équation (7.9) et fonction cible à estimer* $f : \ x \in \mathbb{R} \mapsto \sin(2\pi x)$.

Une fonction de régression est estimée à l'aide d'un perceptron à une couche cachée. Dans toute la suite, ce perceptron est entraîné par approximation de la minimisation de l'erreur quadratique et la fonction de transfert de la couche de sortie est l'identité.

Les scripts ayant permis l'obtention des résultats présentés dans ce chapitre ainsi que les sorties commentées de ces scripts sont disponibles à `http://www.nathalievilla.org/teaching/jes2016`. Le script contient, outre l'exemple inclus dans ce chapitre, un cas d'étude sur un jeu de données réel.

7.5.1 Choix de Q

L'influence du nombre de neurones sur la couche cachée, Q, est tout d'abord étudiée en faisant varier la valeur de Q entre 1 et 10 (pour $\lambda = 0$). Pour chaque valeur de Q, 10 perceptrons sont entraînés et celui avec la plus petite erreur empirique, $\widehat{\mathcal{R}}_n(\mathbf{w})$, est conservé. On notera $\widehat{f}_Q$ le perceptron retenu pour Q neurones sur la couche cachée. Deux quantités sont calculées :

1. $\forall\, Q \in \{1, \ldots, 10\}$, $\widehat{\mathcal{R}}_n(\mathbf{w},Q)$ désignera l'erreur empirique de $\widehat{f}_Q$. Celle-ci sera appelée *erreur d'apprentissage* ;
2. $\forall\, Q \in \{1, \ldots, 10\}$, $\widehat{\mathcal{R}}_T(\mathbf{w},Q)$ désignera l'erreur quadratique moyenne de $\widehat{f}_Q$ pour un ensemble, τ de 1000 points uniformément répartis dans [0,1]. Celle-ci sera appelée *erreur de test* et correspond à une estimation de la quantité

$$\int_{x\in[0,1]} \left(\widehat{f}_Q(x) - f(x)\right)^2 dx = \mathbb{E}\left[\left(\widehat{f}_Q(X) - f(X)\right)^2\right],$$

puisque X est uniforme sur [0,1].

La figure 7.6 représente, pour diverses valeurs de Q, la fonction $\widehat{f}_Q$ obtenue et la figure 7.7 montre l'évolution des erreurs d'apprentissage selon Q. La première illustre bien le phénomène de sous/sur-apprentissage : lorsque Q a une valeur faible ($Q = 1$ par exemple) la fonction estimée est très simple et ne réussit pas à approximer correctement la fonction cible f. Lorsque le nombre de neurones sur la couche cachée devient grand (à partir de $Q = 5$ par exemple), la fonction estimée tend à sur-apprendre en s'adaptant aux irrégularités du jeu de données d'apprentissage.

La figure 7.7 confirme ce phénomène : l'erreur d'apprentissage est presque constamment décroissante lorsque Q augmente, le modèle s'adaptant de plus en plus précisément aux données d'apprentissage. L'erreur de test, plus élevée, atteint, pour sa part, un minimum pour $Q = 3$.

7.5.2 Choix de λ

Comme expliqué dans la section 7.4.4, une des stratégies permettant d'éviter le sur-apprentissage consiste à choisir Q suffisamment grand et à utiliser une pénalisation des poids comme dans l'équation (7.8). Dans cette partie, nous choisissons $Q = 10$ et faisons varier λ dans $\{10^{-1}, 10^{-2}, \ldots, 10^{-10}, 0\}$. Le même principe que dans la partie précédente, est utilisé pour l'apprentissage du perceptron et la fonction obtenue est notée $\widehat{f}_\lambda$.

La figure 7.8 représente, pour les diverses valeurs de λ, la fonction[2] $\widehat{f}_\lambda$ et la figure 7.9 montre l'évolution des erreurs d'apprentissage selon $-\log_{10}(\lambda)$ pour les valeurs de $\lambda > 0$.

[2]On notera que la fonction $\widehat{f}_\lambda$ obtenue pour $\lambda = 0$ n'est pas identique à la fonction $\widehat{f}_Q$

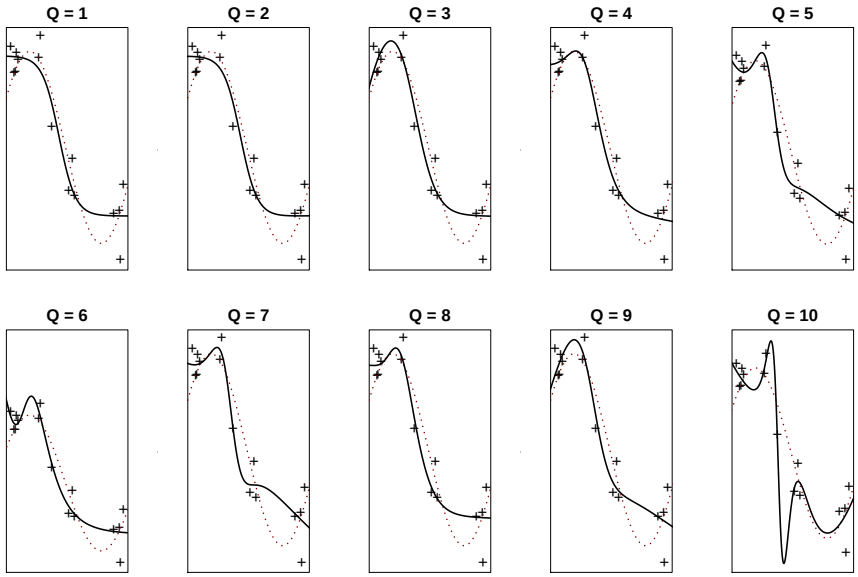

Figure 7.6 : *Ensemble d'apprentissage comme dans la figure 7.5 (croix), fonction à estimer f (en pointillés) et fonction $\widehat{f}_Q$ selon Q (en traits pleins).*

Les valeurs faibles de λ correspondent aux solutions avec une grande complexité qui présentent de nombreuses irrégularités locales correspondant aux données observées alors que les plus grandes valeurs de λ fournissent une solution trop simple, presque linéaire. Un bon compromis semble être trouvé pour $\lambda = 10^{-3}$, ce que confirme aussi l'évolution des erreurs de test et apprentissage. Nous signalons au passage que la valeur minimale de l'erreur de test $\widehat{\mathcal{R}}_T(\mathbf{w},\lambda)$ pour $Q = 10$ et les différentes valeurs de λ testées est inférieure à la valeur minimale de l'erreur de test $\widehat{\mathcal{R}}_T(\mathbf{w},Q)$ pour $\lambda = 0$ et les différentes valeurs de Q testées (environ 0.0399 contre 0.0450). Ainsi, dans cet exemple, une bonne calibration du paramètre de régularisation s'avère plus efficace qu'une bonne calibration de Q pratiquée sans régularisation.

Finalement, le package **e1071** est utilisé pour calibrer par validation croisée le paramètre λ pour $Q = 10$. La figure 7.10 montre l'évolution de la valeur de l'erreur de validation croisée (V-fold avec $V = 10$) en fonction de $-\log_{10}(\lambda)$ pour les valeurs de $\lambda > 0$ et confirme qu'un minimum est bien atteint pour $\lambda = 10^{-3}$.

---

obtenue dans la partie précédente pour la valeur $Q = 10$: ce phénomène est dû au fait que l'apprentissage du perceptron ne retourne qu'un minimum local, qui peut varier assez fortement d'une initialisation à l'autre. Même 10 initialisations différentes n'assurent pas, y compris sur ce cas simple, une stabilité de la solution retournée.

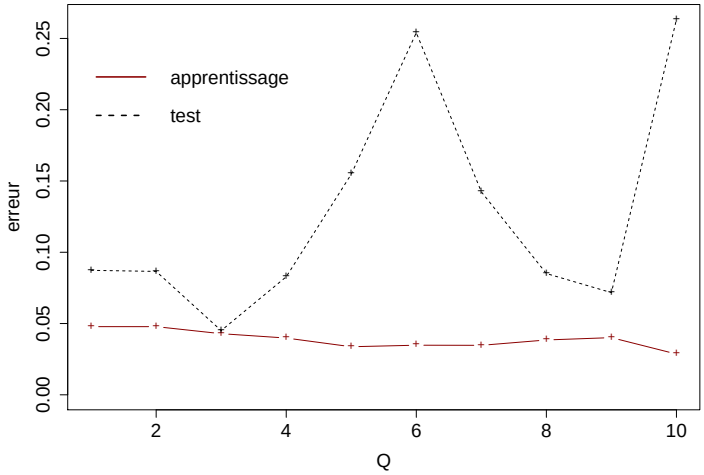

Figure 7.7 : *Évolution des erreurs d'apprentissage ($\widehat{\mathcal{R}}_n(\mathbf{w},Q)$, en traits pleins) et de test ($\widehat{\mathcal{R}}_T(\mathbf{w},Q)$, en pointillés) selon Q.*

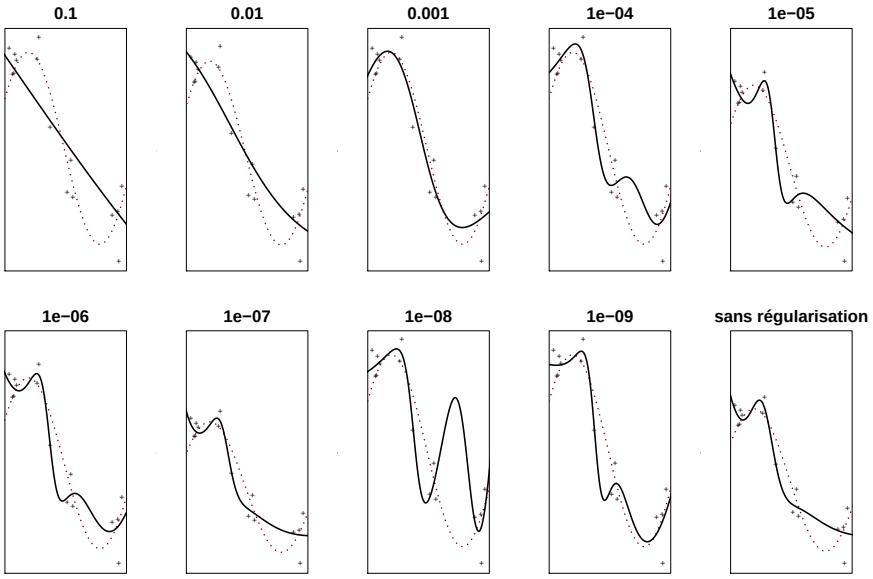

Figure 7.8 : *Ensemble d'apprentissage comme dans la figure 7.5 (croix), fonction à estimer f (en pointillés) et fonction $\widehat{f}_\lambda$ selon λ (en traits pleins).*

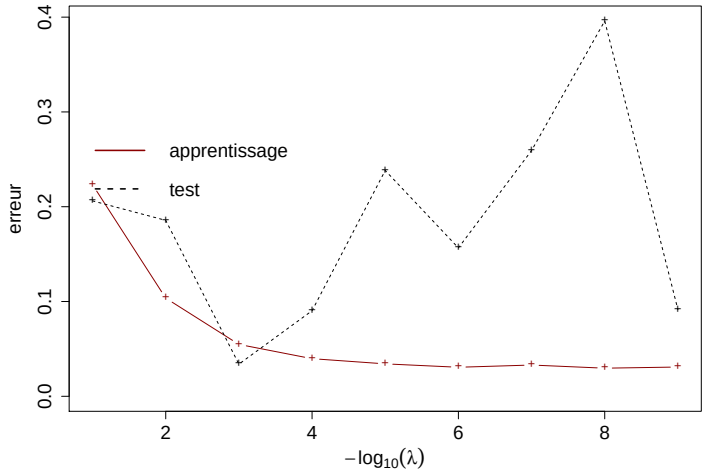

Figure 7.9 : *Évolution des erreurs d'apprentissage ($\widehat{\mathcal{R}}_n(\mathbf{w},\lambda)$, en traits pleins) et de test ($\widehat{\mathcal{R}}_T(\mathbf{w},\lambda)$, en pointillés) selon λ.*

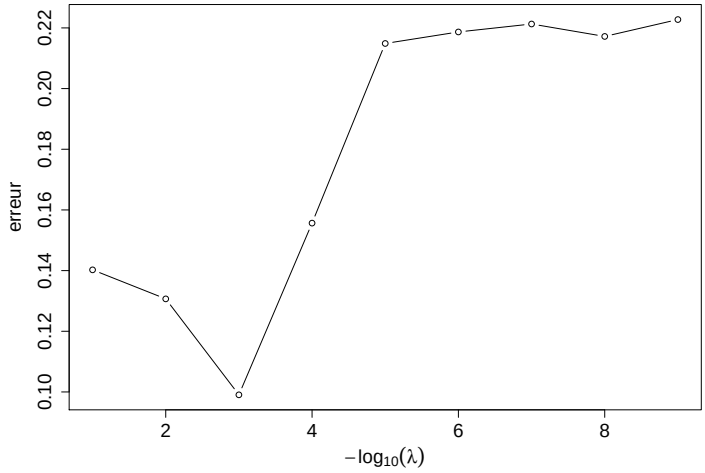

Figure 7.10 : *Évolution de l'erreur de validation croisée en fonction du paramètre de régularisation $-\log_{10}(\lambda)$ pour $Q = 10$.*

7.6 Réseaux de neurones à architectures profondes

7.6.1 *AlexNet*

Si la recherche scientifique subit indéniablement des effets de mode, celle de l'apprentissage profond (ou *deep learning*) a surpris par sa force et son ampleur. Comme le titrait le Monde en 2015, « *cette technologie d'apprentissage, basée sur des réseaux de neurones artificiels, a complètement bouleversé le domaine de l'intelligence artificielle en moins de cinq ans* »[3]. Cette rupture technologique, si rupture il y a, peut être datée de la publication en 2012 des résultats obtenus par Krizhevsky *et al.* [2012] au défi de reconnaissance d'objets « *ImageNet Large Scale Visual Recognition Challenge* »[4] (ILSVRC) [Russakovsky *et al.*, 2015]. Ce défi, conçu pour évaluer la capacité de passage à l'échelle des différents algorithmes de reconnaissance d'objets dans des images, propose chaque année depuis 2010 une compétition construite autour de la base d'images publique *ImageNet*. Les données extraites d'*ImageNet* pour le défi de 2012 comprenaient 1,2 million d'images standardisées étiquetées appartenant à 1 000 classes différentes (de l'ordre de 27 gigabytes de données). La figure 7.11 présente quelques exemples d'images de cette extraction d'*ImageNet* et la hiérarchie des classes associées.

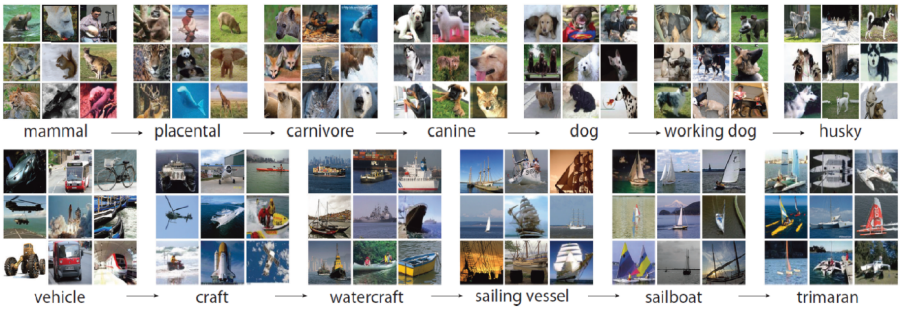

Figure 7.11 : *Exemples d'images issues de la base d'exemples ImageNet.*

Le modèle proposé par Krizhevsky *et al.* [2012] pour traiter cette masse de données est présenté figure 7.12. C'est un réseau de neurones à convolution de neuf couches baptisé *AlexNet*, comprenant près de soixante millions de paramètres à régler. L'apprentissage de ce réseau (l'estimation de ces soixante millions de paramètres) est réalisé par rétro-propagation du gradient régularisé et ne prend *que* près d'une semaine de calcul grâce à l'utilisation de cartes

[3] `http://www.lemonde.fr/pixels/article/2015/07/24/comment-le-deep-learning-revolutionne-l-intelligence-artificielle_4695929_4408996.html`

[4] `http://www.image-net.org`

graphiques (GPU) particulièrement adaptées à ce type de calcul.

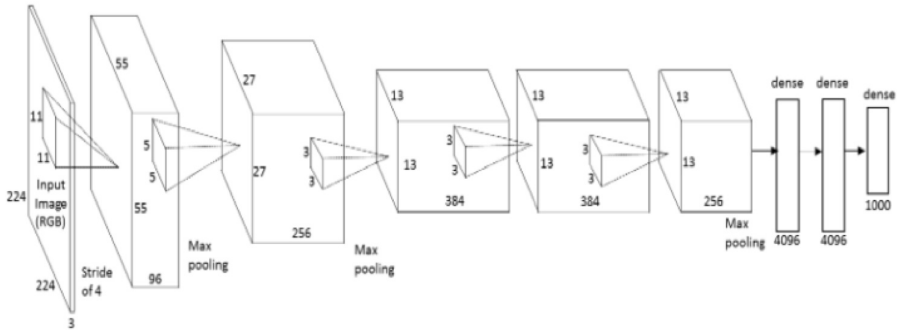

Figure 7.12 : *Architecture de l'*AlexNet*, le réseau gagnant de la compétition ILSVRC 2012 [d'après Krizhevsky* et al.*, 2012].*

Ce qui a frappé les esprits c'est non seulement la qualité des résultats obtenus (le modèle proposé a gagné la compétition) mais aussi la différence de performances avec les autres approches proposées. *AlexNet* a atteint une erreur de l'ordre de 15% alors que les autres modèles ont plafonné à 26%. C'est à cette époque que cette même association entre un grand nombre d'exemples, un modèle de réseau de neurones plus grand et des moyens de calcul adaptés (GPU) a été popularisée sous le nom de « *deep learning* » (apprentissage profond). Cette approche a permis d'obtenir d'excellents résultats dans de nombreuses autres compétitions et dans le domaine de la reconnaissance vocale, qui reste d'ailleurs le premier domaine d'application industrielle de l'apprentissage profond [LeCun *et al.*, 2015]. En 2015, ce même type de réseau profond a été utilisé dans *AlphaGo*, le premier programme ayant battu un joueur professionnel au jeu de Go.

7.6.2 L'architecture des réseaux de neurones profonds

D'après Yann LeCun, c'est pour rompre avec la mauvaise image des réseaux de neurones que le terme « deep learning » a été adopté. Dans son esprit, il n'y a pas de différence conceptuelle entre les réseaux de neurones multicouches proposés dans les années 90 et les réseaux profonds d'aujourd'hui mais simplement une différence d'échelle (le réseau contient plus de couches cachées). De manière plus précise, un réseau profond est juste un réseau de neurones avec plus de trois couches. Pourtant, à y regarder de plus près, les choses ne sont pas si simples.

Si les réseaux profonds permettent de passer à l'échelle lorsque le nombre d'exemples disponibles est important et que le problème à résoudre est suffisamment complexe (tout en étant réalisable), les temps de calcul sont clairement un goulet d'étranglement et cela impacte l'architecture à mettre en œuvre. De

ce fait, la fonction d'activation choisie est souvent de type ReLU (voir page 5), car elle permet d'entrainer les réseaux plus rapidement. Des études précédentes ont aussi montré l'importance, pour obtenir de bonnes performances, de l'utilisation de couches « à convolution »[5] (non totalement connectées), suivies de sous échantillonnages (« *local max pooling* ») et d'une normalisation locale des réponses des neurones voisins, surtout dans les premières couches d'un réseau. L'ordonnancement et la nature des couches à utiliser sont aussi importants.

Il faut aussi déterminer, pour un problème donné, le nombre de couches à utiliser. Lors de sa présentation à la conférence « *Extract Data* » en 2015, Andrew Ng répond à la question en postulant que, pour résoudre un problème complexe, il faut beaucoup de données et une architecture suffisamment grande pour les intégrer. Il suggère d'augmenter simultanément le nombre d'exemples et le nombre de couches du réseau tant que les performances ne sont pas satisfaisantes.

Cette course au nombre de couches a été jusqu'à la proposition d'une architecture à 152 couches [dite « extrêmement profonde » He *et al.*, 2016] dépassant les performances humaines sur le défi LSVRC[6]. La figure 7.13 montre l'évolution année par année des performances et du nombre de couches des meilleures solutions à ce défi.

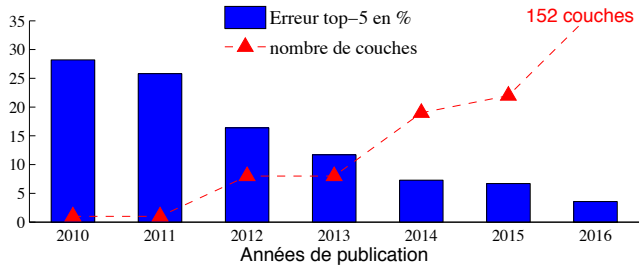

Figure 7.13 : *Évolution, ces dernières années, des meilleures performances sur la base ImageNet [d'après He* et al.*, 2016].*

La nature des couches a aussi évolué. En 2015, Szegedy *et al.* publiaient les détails de GoogLeNet, le réseau ayant gagné le challenge ILSVRC l'année précédente (les résultats de ce réseau sont reportés à l'abscisse 2015 de la figure 7.13). La principale innovation de cette architecture est l'introduction de couches multi-convolutionelles appelées modules d'*inception*. L'idée est de remplacer les convolutions simples par trois au quatre convolutions effectuées en parallèle à différentes échelles (1×1, 3×3, $5 \times 5 \ldots$), combinées (ou non) à des blocs de convolution 1×1 et à une couche de sous échantillonnage.

[5] Une couche de neurones est dite « à convolutions » si chacune de ses unités est activée à travers un même filtre linéaire localisé. Cela revient à estimer un masque local que l'on va appliquer et donc convoluer avec toutes les unités de la couche précédente.

[6] Pour plus de détails sur une estimation des performances humaines voir la table 9 dans [Russakovsky *et al.*, 2015].

Pour plus de détails sur ces différentes architectures, on pourra se reporter à `culurciello.github.io/tech/2016/06/04/nets.html`.

Un autre aspect intéressant des réseaux de neurones profonds est le caractère « générique » des représentations qu'ils proposent. Un argument souvent mis en avant est la capacité de ces réseaux à apprendre automatiquement de bonnes caractéristiques discriminantes [Bengio *et al.*, 2013]. L'importance de ce point est telle que la principale conférence du domaine s'intitule « *International Conference on Learning Representations* ». Chaque couche du réseau profond est associée à un traitement relativement simple permettant d'obtenir une représentation légèrement plus abstraite que la précédente. La figure 7.14 illustre ce point sur un réseau à convolution de type réseau *AlexNet*.

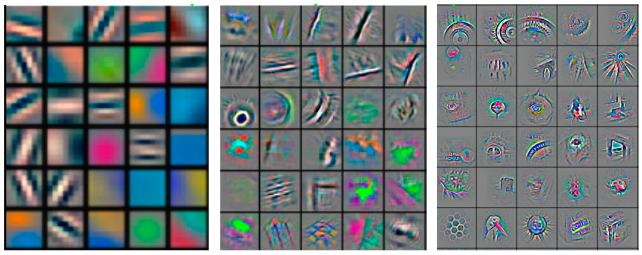

Figure 7.14 : *Évolution des représentations dans différentes couches successives d'un réseau de type* AlexNet *[extraite d'un tutoriel de Yann Le Cun d'après Zeiler and Fergus, 2014].*

Cette propriété de représentation permet de faire du transfert d'apprentissage et d'utiliser des réseaux comme *AlexNet*, déjà entrainés sur un problème, pour en résoudre un autre. La pratique consiste à utiliser le même type d'entrées (les trois canaux RGB des images couleur dans le cas d'*AlexNet*) et à ne modifier que la dernière couche. Ainsi, par exemple, pour discriminer des images satellites (jamais vues par *AlexNet*), on peut présenter ces images au réseau et utiliser les sorties de l'avant dernière couche comme nouvelles caractéristiques, comme entrées pour un autre algorithme d'apprentissage (SVM par exemple).

Notons qu'au delà des réseaux à convolution, il existe de nombreuses autres architectures de réseaux profonds, souvent présentées en deux grandes familles. Il s'agit des modèles profonds discriminants, comme les réseaux à convolutions ou les réseaux récurrents, et des modèles génératifs/non supervisés, comme les machine de Boltzmann (RBM), les réseaux de croyances (DBN), ou les auto-encodeurs. Pour plus de détails on pourra consulter le livre de Goodfellow *et al.* [2016].

7.6.3 L'apprentissage des réseaux de neurones profonds

L'apprentissage des réseaux de neurones profonds a aussi ses spécificités et dans la nouvelle édition de leur livre sur l'apprentissage des réseaux de neurones, Orr and Müller [2003] consacrent plusieurs chapitres au thème « *big learning and deep neural network* ».

Les réseaux profonds se caractérisent par une sur-paramétrisation extrême. Leur apprentissage va donc nécessiter l'utilisation de plusieurs techniques permettant de lutter contre le sur-apprentissage. Un premier moyen est l'augmentation artificielle du nombre d'exemples en utilisant différents types de modifications des données disponibles. Dans le cas d'images, on peut utiliser des translations, des rotations ou même la convolution avec certains types de bruit.

Le « *dropout* » [Srivastava *et al.*, 2014] est une autre technique de régularisation utilisée avec succès pour entraîner *AlexNet*. Cette méthode consiste, chaque fois qu'une entrée est présentée, à annuler avec une chance sur deux la réponse de chacun des neurones cachés. Les neurones qui sont ainsi ignorés ne contribuent ni au calcul de la sortie ni à la rétro-propagation. À chaque étape de l'apprentissage c'est donc un réseau avec une architecture différente qui est utilisé, tous ces réseaux partageant les mêmes poids. Lors de la phase de test, tous les neurones sont utilisés, mais en divisant leurs sorties par deux, ce qui est une approximation raisonnable de la moyenne géométrique des distributions de prédiction produites par l'ensemble des réseaux appris par ce mécanisme.

L'empilement des auto-encodeurs est une autre technique proposée pour accélérer l'apprentissage des réseaux de neurones profonds [Vincent *et al.*, 2010]. L'idée ici est d'apprendre successivement chacune des couches cachées en utilisant en entrée et en sortie les mêmes données. C'est ce qui est appelé l'auto-encodage, problème dont l'analyse en composantes principales est sans doute la version la plus simple.

Une difficulté bien documentée, liée à la rétropropagation du gradient dans les réseaux profonds, est le phénomène de disparition du gradient (« *gradient vanishing* »), dû à la présence d'un grand nombre de couches. En effet, une longue série de compositions non-linéaires a tendance à diminuer les gradients et donc à perdre le signal d'erreur. Pour contrer ce phénomène, He *et al.* [2016] ont proposé l'architecture des réseaux résiduels (« *residual networks* » ou « *ResNet* ») qui consiste simplement à additionner les sorties d'une couche c avec celles de couches précédentes $c-2$ ou $c-3$. Ce faisant, le signal de gradient est préservé pendant la rétropropagation ce qui permet de construire un réseau extrêmement profond (152 couches) et d'obtenir les meilleurs résultats sur LSVRC en 2015.

Il n'en reste pas moins que, malgré quelques résultats récents [Mallat, 2016], la justification théorique de ces méthodes reste à établir.

Enfin, pour mettre en œuvre l'apprentissage profond, il existe aujourd'hui plusieurs environnements logiciels conçus pour faciliter l'utilisation et l'entraî-

nement de ces réseaux, avec ou sans GPU. Parmi ceux-ci, les plus populaires[7] sont TensorFlow, Theano (avec Lasagne), une de leur surcouche appelée Keras, Caffe et Torch.

[7] `http://www.github.com/aymericdamien/TopDeepLearning`

Bibliographie

Barron, A. [1994]. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, **14**, 115–133.

Baum, E. [1988]. On the capabilities of multi-layer perceptrons. *Journal of Complexity*, 193–215.

Baum, E. and Haussler, D. [1989]. What size net gives valid generalization ? *Neural Computation 1*, 151–160.

Bengio, Y., Courville, A. and Vincent, P. [2013]. Representation learning : A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, **35**(8), 1798–1828.

Bishop, C. [1995]. *Neural Networks for Pattern Recognition*. Oxford University Press, New York, USA.

Bishop, C. [2006]. *Pattern Recognition and Machine Learning*. Springer-Verlag, New York, NY, USA.

Devroye, L. P., Györfi, L. and Lugosi, G. [1996]. *A Probabilistic Theory of Pattern Recognition*, volume 31 of *Applications of Mathematics (New York)*. Springer-Verlag, New York. ISBN 0-387-94618-7.

Farago, A. and Lugosi, G. [1993]. Strong universal consistency of neural network classifiers. *IEEE Transactions on Information Theory*, **39**(4), 1146–1151.

Fletcher, R. [1987]. *Practical Methods of Optimization*. Wiley, 2nd edition.

Gill, P., Murray, W. and Wright, M. [1981]. *Practical Optimization*. Academic Press.

Goodfellow, I., Bengio, Y. and Courville, A. [2016]. Deep Learning. Book in preparation for MIT Press, URL `http://www.deeplearningbook.org`.

Györfi, L., Kohler, M., Krzyżak, A. and Walk, H. [2002]. *A Distribution-free Theory of Nonparametric Regression*. Springer Series in Statistics. Springer-Verlag, New York. ISBN 0-387-95441-4.

He, K., Zhang, X., Ren, S. and Sun, J. [2016]. Deep Residual Learning for Image Recognition. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, .

Hebb, D. [1949]. *The Organization of Behavior*. Wiley, New York.

Hornik, K. [1991]. Approximation capabilities of multilayer feedfoward networks. *Neural Networks*, **4**(2), 251–257.

Hornik, K. [1993]. Some new results on neural network approximation. *Neural Networks*, **6**(8), 1069–1072.

Kohonen, T. [1995]. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Science*. Springer.

Kohonen, T. [2014]. *MATLAB Implementations and Applications of the Self-Organizing Map*. Unigrafia Oy, Helsinki, Finland.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. [2012]. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

LeCun, Y. [1985]. Une procédure d'apprentissage pour réseau à seuil assymétrique. In *Proceedings of Cognitiva 85 (À la Frontière de l'Intelligence Artificielle, des Sciences de la Connaissance et des Neurosciences)*, Paris, France.

LeCun, Y., Bengio, Y. and Hinton, G. [2015]. Deep learning. *Nature*, **521**(7553), 436–444.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. [2001]. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, 306–351. IEEE Press.

MacKay, D. [1992]. Information-based objective functions for active data selection. *Neural Computation*, **4**(4), 590–604.

Mallat, S. [2016]. Understanding deep convolutional networks. *Phil. Trans. R. Soc. A*, **374**(2065), 20150203.

McCaffrey, D. and Gallant, A. [1994]. Convergence rates for single hidden layer feedforward networks. *Neural Networks*, **7**(1), 115–133.

McCulloch, W. S. and Pitts, W. [1943]. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, **5**(4), 115–133.

Minsky, M. and Papert, S. [1969]. *Perceptrons : an Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Nocedal, J. and Wright, S. [1999]. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, NY, USA.

Orr, G. B. and Müller, K.-R. [2003]. *Neural networks : tricks of the trade*. Springer.

Pinkus, A. [1999]. Approximation theory of the MLP model in neural networks. *Acta Numerica*, **8**, 143–195.

Powell, M. [1987]. Radial basis functions for multivariate interpolation : a review. In J. Mason, M. Cox (editors), *Algorithms for Approximation*, Oxford University Press, Oxford, UK.

Ripley, B. [1996]. *Pattern Recognition and Neural Networks*. Cambridge University Press.

Rosenblatt, F. [1958]. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological Review*, **65**, 386–408.

Rosenblatt, F. [1962]. *Principles of Neurodynamics : Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, DC, USA.

Rumelhart, D. and Mc Clelland, J. [1986]. *Parallel Distributed Processing : Exploration in the MicroStructure of Cognition*. MIT Press, Cambridge, MA, USA.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. and Fei-Fei, L. [2015]. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, **115**(3), 211–252.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. [2014]. Dropout : a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, **15**(1), 1929–1958.

Stinchcombe, M. [1999]. Neural network approximation of continuous functionals and continuous functions on compactifications. *Neural Network*, **12**(3), 467–477.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. [2015]. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1–9.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.-A. [2010]. Stacked denoising autoencoders : Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, **11**(Dec), 3371–3408.

White, H. [1990]. Connectionist nonparametric regression : multilayer feedforward networks can learn arbitraty mappings. *Neural Networks*, **3**, 535–549.

White, H. [1991]. Nonparametric estimation of conditional quantiles using neural networks. In *Proceedings of the 23rd Symposium of the Interface : Computing Science and Statistics*, 190–199. American Statistical Association, Alexandria, VA, USA.

Zeiler, M. D. and Fergus, R. [2014]. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, 818–833. Springer.

Index