Réseaux de neurones

Résumé

Définition et caractéristiques des réseaux de neurones limitée aux perceptrons multicouches spécifiques pour la régression et la classification supervisée. Structure, fonctions de transfert, algorithme d'apprentissage par rétro-propagation du gradient, contrôles du sur-ajustement, introduction à l'apprentissage profond.

Retour au plan du cours

1 Introduction

1.1 Historique

L'*Intelligence Artificielle*, branche de l'Informatique fondamentale s'est développée avec pour objectif la simulation des comportements du cerveau humain. Les premières tentatives de modélisation du cerveau sont anciennes et précèdent même l'ère informatique. C'est en 1943 que Mc Culloch (neurophysiologiste) et Pitts (logicien) ont proposé les premières notions de *neurone formel*. Ce concept fut ensuite mis en réseau avec une couche d'entrée et une sortie par Rosenblatt en 1959 pour simuler le fonctionnement rétinien et tacher de reconnaître des formes. C'est l'origine du *perceptron*. Cette approche dite *connexioniste* a atteint ses limites technologiques, compte tenu de la puissance de calcul de l'époque, mais aussi théoriques au début des années 70.

L'approche connexioniste à *connaissance répartie* a alors été supplantée par une approche *symbolique* qui promouvait les *systèmes experts* à *connaissance localisée* dont L'objectif était d'automatiser le principe de l'expertise humaine en associant trois concepts :

- une *base de connaissance* dans laquelle sont regroupées les connaissances d'experts humains sous forme de propositions logiques élémentaires ou plus élaborées en utilisant des quantificateurs (logique du premier ordre).
- une *base de faits* contenant les observations du cas à traiter comme, par exemple, des résultats d'examens, d'analyses de sang, de salive pour des applications biomédicales de choix d'un antibiotique,
- un *moteur d'inférence* chargé d'appliquer les règles expertes sur la base de faits afin d'en déduire de nouveaux faits jusqu'à la réalisation d'un objectif comme le choix du traitement d'un infection bactérienne.

Face aux difficultés rencontrées lors de la modélisation des connaissances d'un expert humain, au volume considérable des bases qui en découlaient et au caractère exponentiel de la complexité des algorithmes d'inférence mis en jeu, cette approche s'est éteinte avec les années 80. Il a été montré que les systèmes basés sur le calcul des prédicats du premier ordre conduisaient à des problèmes NP complets.

L'essor technologique et quelques avancées théoriques :

- estimation du gradient par rétro-propagation de l'erreur (Hopkins, 1982),
- analogie de la phase d'apprentissage avec les modèles markoviens de systèmes de particules de la mécanique statistique (verres de spin) par (Hopfield, 1982),

au début des années 80 ont permis de relancer l'approche connexioniste. Celle-ci a connu au début des années 90 un développement considérable si l'on considère le nombre de publications et de congrès qui lui ont été consacrés mais aussi les domaines d'applications très divers où elle apparaît. La motivation initiale de simulation du cortex cérébral a été rapidement abandonné alors que les méthodes qui en découlaient ont trouvé leur propre intérêt de développement méthodologique et leurs champs d'applications.

Remis en veilleuse depuis le milieu des années 90 au profit d'autres algorithmes d'*apprentissage machine* ou plutôt statistique : *boosting, support vector machine*..., les réseaux de neurones connaissent un regain d'intérêt et même un énorme battage médiatique sous l'appellation d'apprentissage profond (*deep learning*). La taille des bases de données, notamment celles d'images issues d'internet, associée à la puissance de calcul disponible, permettent d'estimer les millions de paramètres de percetrons accumulant des dizaines voire centaines de couches de neurones aux propriétés très spécifiques. Ce succès médiatique est la conséquence des résultats spectaculaires obtenus par ces réseaux en reconnaissance d'image, jeux de go, traitement du langage naturel...

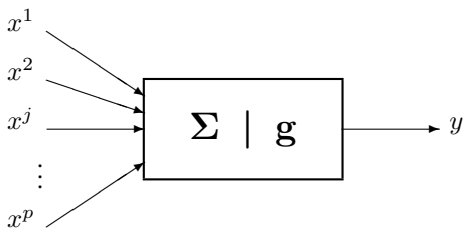


FIGURE 1 – *Représentation d'un neurone formel.*

1.2 Réseaux de neurones

Un *réseau neuronal* est l'association, en un graphe plus ou moins complexe, d'objets élémentaires, les *neurones formels*. Les principaux réseaux se distinguent par l'organisation du graphe (en couches, complets...), c'est-à-dire leur architecture, son niveau de complexité (le nombre de neurones, présence ou non de boucles de rétroaction dans le réseau), par le type des neurones (leurs fonctions de transition ou d'activation) et enfin par l'objectif visé : apprentissage supervisé ou non, optimisation, systèmes dynamiques...

1.3 Neurone formel

De façon très réductrice, un neurone biologique est une cellule qui se caractérise par

- des synapses, les points de connexion avec les autres neurones, fibres nerveuses ou musculaires ;
- des dentrites ou entrées du neurones ;
- les axones, ou sorties du neurone vers d'autres neurones ou fibres musculaires ;
- le noyau qui active les sorties en fonction des stimulations en entrée.

Par analogie, le neurone formel est un modèle qui se caractérise par un état interne $s \in \mathcal{S}$, des signaux d'entrée $x_1, \ldots, x_p$ et une fonction d'activation

$$s = h(x_1, \ldots, x_p) = g\left(\alpha_0 + \sum_{j=1}^{p} \alpha_j x_j\right) = g(\alpha_0 + \boldsymbol{\alpha}'\boldsymbol{x}).$$

La fonction d'activation opère une transformation d'une combinaison affine des signaux d'entrée, α_0, terme constant, étant appelé le biais du neurone. Cette combinaison affine est déterminée par un *vecteur de poids* $[\alpha_0, \ldots, \alpha_p]$ associé à chaque neurone et dont les valeurs sont estimées dans la phase d'apprentissage. Ils constituent la *mémoire* ou *connaissance répartie* du réseau.

Les différents types de neurones se distinguent par la nature g de leur fonction d'activation. Les principaux types sont :

- *linéaire* g est la fonction identité,
- *seuil* $g(x) = \mathbf{1}_{[0,+\infty[}(x)$,
- *sigmoïde* $g(x) = 1/(1+e^x)$,
- *ReLU* $g(x) = \max(0, x)$ (*rectified linear unit*)
- *radiale* $g(x) = \sqrt{1/2\pi}\exp(-x^2/2)$,
- *stochastiques* $g(x) = 1$ avec la probabilité $1/(1+e^{-x/H})$, 0 sinon (H intervient comme une température dans un algorithme de recuit simulé),
- ...

Les modèles linéaires, sigmoïdaux, ReLU, sont bien adaptés aux algorithmes d'apprentissage impliquant (cf. ci-dessous) une rétro-propagation du gradient car leur fonction d'activation est différentiable ; ce sont les plus utilisés. Le modèle à seuil est sans doute plus conforme à la réalité biologique mais pose des problèmes d'apprentissage. Enfin le modèle stochastique est utilisé pour des problèmes d'optimisation globale de fonctions perturbées ou encore pour les analogies avec les systèmes de particules (machine de Bolzman).

2 Perceptron multicouche

Nous ne nous intéresserons dans ce cours qu'à une structure élémentaire de réseau, celle dite statique ne présentant pas de boucle de rétroaction et dans un but d'apprentissage supervisé. Les systèmes dynamiques, avec boucle de rétroaction ainsi que les cartes de Kohonen ou cartes auto-organisatrices pour

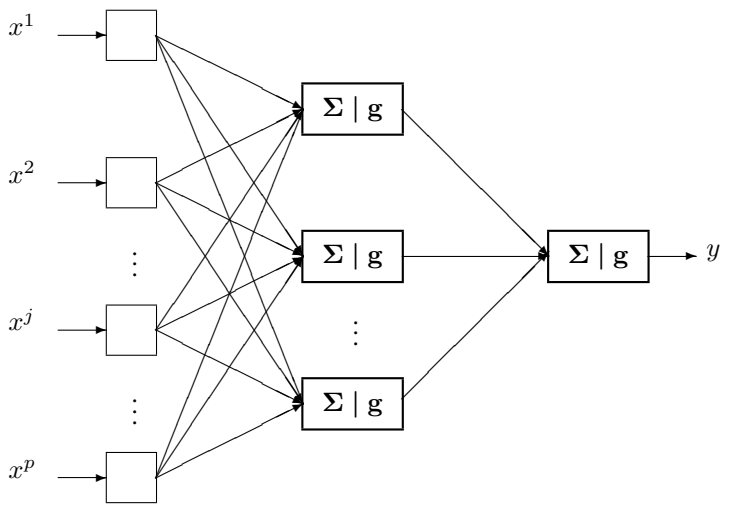


FIGURE 2 – *Exemple de perceptron multicouche élémentaire avec une couche cachée et une couche de sortie.*

la classification non supervisées ne sont pas abordés.

2.1 Architecture

Le perceptron multicouche (PMC) est un réseau composé de couches successives. Une *couche* est un ensemble de neurones n'ayant pas de connexion entre eux. Une couche d'entrée lit les signaux entrant, un neurone par entrée x_j, une couche en sortie fournit la réponse du système. Selon les auteurs, la couche d'entrée qui n'introduit aucune modification n'est pas comptabilisée. Une ou plusieurs couches cachées participent au transfert.

Dans un perceptron, un neurone d'une couche cachée est connecté en entrée à chacun des neurones de la couche précédente et en sortie à chaque neurone de la couche suivante.

2.2 Fonction de transfert

Par souci de cohérence, les mêmes notations ont été conservées à travers les différents chapitres. Ainsi, les *entrées* d'un réseau sont encore notées $X_1, \ldots, X_p$ comme les variables explicatives d'un modèle tandis que les *poids* des entrées sont des paramètres $\boldsymbol{\alpha}, \boldsymbol{\beta}$ à estimer lors de la procédure d'*apprentissage* et que la *sortie* est la variable Y à expliquer ou cible du modèle.

Un perceptron multicouche réalise donc une transformation des variables d'entrée :

$$Y = f(X_1, \ldots, X_p; \boldsymbol{\alpha})$$

où $\boldsymbol{\alpha}$ est le vecteur contenant chacun des paramètres $\alpha_{jk\ell}$ de la $j$ème entrée du $k$ème neurone de la ℓème couche ; la couche d'entrée ($\ell = 0$) n'est pas paramétrée, elle ne fait que distribuer les entrées sur tous les neurones de la couche suivante.

Un théorème dit d*approximation universelle* montre que cette structure élémentaire à une seule couche cachée est suffisante pour prendre en compte les problèmes classiques de modélisation ou apprentissage statistique. En effet, toute fonction régulière peut être approchée uniformément avec une précision arbitraire et dans un domaine fini de l'espace de ses variables, par un réseau de neurones comportant une couche de neurones cachés en nombre fini possédant tous la même fonction d'activation et un neurone de sortie linéaire. *Attention*, ce résultat, qui semble contradictoire avec les structures d'apprentissage profond, est théorique, il masque des difficultés d'apprentissage et de stabilité pour des problèmes complexes en très grande dimension.

De façon usuelle et en régression (Y quantitative), la dernière couche est constituée d'un seul neurone muni de la fonction d'activation identité tandis que les autres neurones (couche cachée) sont munis de la fonction sigmoïde. En classification binaire, le neurone de sortie est muni également de la fonction sigmoïde tandis que dans le cas d'une discrimination à m classes (Y qualitative), ce sont m neurones avec fonction sigmoïde, un par classe, qui sont considérés en sortie.

Ainsi, en régression avec un perceptron à une couche cachée de q neurones et un neurone de sortie, cette fonction s'écrit :

$$\begin{aligned} y = f(\boldsymbol{x};\boldsymbol{\alpha},\boldsymbol{\beta}) &= \beta_0 + \boldsymbol{\beta}'\boldsymbol{z} \\ \text{avec} \quad z_k &= g(\alpha_{k0} + \boldsymbol{\alpha_k}'\boldsymbol{x}); k = 1,\ldots,q. \end{aligned}$$

2.3 Apprentissage

Supposons que l'on dispose d'une base d'apprentissage de taille n d'observations $(x_i^1,\ldots,x_i^p;y_i)$ des variables explicatives $X^1,\ldots,X^p$ et de la variable à prévoir Y. Considérons le cas le plus simple de la régression avec un réseau constitué d'un neurone de sortie linéaire et d'une couche à q neurones dont les paramètres sont optimisés par moindres carrés. Ceci se généralise à toute fonction perte dérivable et donc à la discrimination à m classes.

L'apprentissage est l'estimation des paramètres $\boldsymbol{\alpha}_{j=0,p;k=1,q}$ et $\boldsymbol{\beta_{k=0,q}}$ par minimisation de la fonction perte quadratique (ou d'un fonction d'entropie en classification) :

$$Q(\alpha,\beta) = \sum_{i=1}^{n} Q_i = \sum_{i=1}^{n} \left[y_i - f(\boldsymbol{x};\boldsymbol{\alpha},\boldsymbol{\beta})\right]^2.$$

Différents algorithmes d'optimisation sont proposés, ils sont généralement basés sur une évaluation du gradient par rétro-propagation.

2.3.1 Rétro-propagation de l'erreur

Il s'agit donc dévaluer la dérivée de la fonction coût en une observation et par rapport aux différents paramètres. Soit $z_{ki} = g(\alpha_{k0} + \boldsymbol{\alpha_k}'\boldsymbol{x}_i)$ et $\boldsymbol{z}_i = \{z_{1i},\ldots,z_{qi}\}$. Les dérivées partielles de la fonction perte quadratique s'écrivent :

$$\begin{aligned} \frac{\partial Q_i}{\partial \beta_k} &= -2(y_i - \phi(x_i))(\boldsymbol{\beta}'\boldsymbol{z}_i)z_{ki} = \delta_i z_{ki} \\ \frac{\partial Q_i}{\partial \alpha_{kj}} &= -2(y_i - \phi(x_i))(\boldsymbol{\beta}'\boldsymbol{z}_i)\beta_k f'(\boldsymbol{\alpha}_k'\boldsymbol{x}_i)x_{ip} = s_{ki}x_{ip}. \end{aligned}$$

Les termes δ_i et s_{ki} sont respectivement les termes d'erreur du modèle courant à la sortie et sur chaque neurone caché. Ces termes d'erreur vérifient les équations dites de rétro-propagation :

$$s_{ki} = g'(\boldsymbol{\alpha}_k'\boldsymbol{x}_i)\beta_k\delta_i$$

dont les termes sont évaluée sen deux passes. Une *passe avant*, avec les valeurs courantes des poids : l'application des différentes entrées $\boldsymbol{x}_i$ au réseau permet de déterminer les valeurs ajustées $\widehat{f}(\boldsymbol{x}_i)$. La *passe retour* permet ensuite de déterminer les δ_i qui sont *rétro-propagés* afin de calculer les s_{ki} et ainsi obtenir les évaluations des gradients.

2.3.2 Algorithmes d'optimisation

Sachant évaluer les gradients, différents algorithmes, plus ou moins sophistiqués, sont implémentés. Le plus élémentaire est une utilisation itérative du gradient : en tout point de l'espace des paramètres, le vecteur gradient de Q pointe dans la direction de l'erreur croissante. Pour faire décroître Q il suffit donc de se déplacer en sens contraire. Il s'agit d'un algorithme itératif modifiant les poids de chaque neurone selon :

$$\begin{aligned} \beta_k^{(r+1)} &= \beta_k^{(r)} - \tau \sum_{i=1}^{n} \frac{\partial Q_i}{\partial \beta_k^{(r)}} \\ \alpha_{kp}^{(r+1)} &= \alpha_{kp}^{(r)} - \tau \sum_{i=1}^{n} \frac{\partial Q_i}{\partial \alpha_{kp}^{(r)}}. \end{aligned}$$

Le coefficient de proportionnalité τ est appelé le *taux d'apprentissage*. Il peut être fixe, à déterminer par l'utilisateur, ou encore varier en cours d'exécution selon certaines heuristiques. Il paraît en effet intuitivement raisonnable que, grand au début pour aller plus vite, ce taux décroisse pour aboutir à un réglage plus fin au fur et à mesure que le système s'approche d'une solution.

Si l'espace mémoire est suffisant, une version accélérée de l'algorithme fait intervenir à chaque itération un ensemble (*batch*) d'observations pour moyenner les gradients et mises à jour des poids.

Bien d'autres méthodes d'optimisation ont été adaptées à l'apprentissage d'un réseau : méthodes du gradient avec second ordre utilisant une approximation itérative de la matrice hessienne (algorithme BFGS, de Levenberg-Marquardt) ou encore une évaluation implicite de cette matrice par la méthode

Algorithm 1 Rétro propagation élémentaire du gradient

`Initialisation` des poids $b_{jk\ell}$ par tirage aléatoire selon une loi uniforme sur $[0,1]$.
Normaliser dans $[0,1]$ les données d'apprentissage.
while $Q >$ errmax ou niter<itermax **do**
  Ranger la base d'apprentissage dans un nouvel ordre aléatoire.
  for chaque élément $i = 1, \ldots, n$ de la base **do**
    Calculer $\varepsilon(i) = y_i - f(x_i^1, \ldots, x_i^p; (b)(i-1))$ en propageant les entrées vers l'avant.
    L'erreur est rétro-propagée dans les différentes couches afin d'affecter à chaque entrée une responsabilité dans l'erreur globale.
    Mise à jour de chaque poids $b_{jk\ell}(i) = b_{jk\ell}(i-i) + \Delta b_{jk\ell}(i)$
  end for
end while

dite du gradient conjugué. La littérature sur le sujet propose quantités de recettes destinées à améliorer la vitesse de convergence de l'algorithme ou bien lui éviter de rester collé à une solution locale défavorable. D'autres heuristiques proposent d'ajouter un terme d'inertie afin d'éviter des oscillations de l'algorithme.

D'autres algorithmes encore sont des versions adaptatives. Lorsque de nouvelles observations sont proposées une à une au réseau. Dans ce dernier type d'algorithme, des propriétés de dynamique markovienne (processus ergodique convergeant vers la mesure stationnaire) impliquent une convergence presque sûre : la probabilité d'atteindre une précision fixée *a priori* tend vers 1 lorsque la taille de l'échantillon d'apprentissage tend vers l'infini.

On pourra se reporter à l'abondante littérature sur le sujet pour obtenir des précisions sur les algorithme d'apprentissage et leurs nombreuses variantes. Il est important de rappeler la liste des choix qui sont laissés à l'utilisateur. En effet, même si les logiciels proposent des valeurs par défaut, il est fréquent que cet algorithme connaisse quelques soucis de convergence.

2.4 Contrôle de la complexité

Régularisation

Dans les réseaux élémentaires, une option simple pour éviter le sur-apprentissage consiste à introduire une terme de pénalisation ou régularisation, comme en régression *ridge*, dans le critère à optimiser. Celui-ci devient alors : $Q(\boldsymbol{\theta}) + \gamma\|\theta\|^2$. Plus la valeur du paramètre γ (*decay*) est importante et moins les poids des entrées des neurones peuvent prendre des valeurs chaotiques contribuant ainsi à limiter les risques de sur-apprentissage.

Choix des paramètres

L'utilisateur doit donc déterminer

1. les variables d'entrée et la variable de sortie ; leur faire subir comme pour toutes méthodes statistiques, d'éventuelles transformations, normalisations.
2. L'architecture du réseau : le nombre de couches cachées qui correspond à une aptitude à traiter des problèmes de non-linéarité, le nombre de neurones par couche cachée. Ces deux choix conditionnent directement le nombre de paramètres (de poids) à estimer et donc la complexité du modèle. Ils participent à la recherche d'un bon compromis biais/variance c'est-à-dire à l'équilibre entre qualité d'apprentissage et qualité de prévision.
3. Trois autres paramètres interviennent également sur ce compromis : le nombre maximum d'itérations, l'erreur maximum tolérée et un terme éventuel de régularisation *ridge (decay)*.
4. Le taux d'apprentissage ainsi qu'une éventuelle stratégie d'évolution de celui-ci.
5. la taille des ensembles ou *batchs* d'observation considérés à chaque itération.

En pratique, tous ces paramètres ne peuvent être réglés simultanément par l'utilisateur. Celui-ci est confronté à des choix concernant principalement le contrôle du sur-apprentissage : limiter le nombre de neurones ou la durée d'apprentissage ou encore augmenter le coefficient de pénalisation de la norme des paramètres. Ceci nécessite de déterminer un mode d'estimation de l'erreur : échantillon validation ou test, validation croisée ou bootstrap.

Une stratégie simple et sans doute efficace consiste à introduire un nombre

plutôt grand de neurones puis à optimiser le seul paramètre de régularisation (decay) par validation croisée.

2.5 Remarques

Les champs d'application des PMC sont très nombreux : discrimination, prévision d'une série temporelle, reconnaissance de forme... Ils sont en général bien explicités dans les documentations des logiciels spécialisés.

Les critiques principales énoncées à l'encontre du PMC concernent les difficultés liés à l'apprentissage (temps de calcul, taille de l'échantillon, localité de l'optimum obtenu) ainsi que son statut de boîte noir. En effet, contrairement à un modèle de discrimination ou un arbre, il est *a priori* impossible de connaître l'influence effective d'une entrée (une variable) sur le système dès qu'une couche cachée intervient. Néanmoins, des techniques de recherche de sensibilité du système à chacune des entrées permettent de préciser les idées et, éventuellement de simplifier le système en supprimant certaines des entrées.

En revanche, ils possèdent d'indéniables qualités lorsque l'absence de linéarité et/ou le nombre de variables explicatives (images) rendent les modèles statistiques traditionnelles inutilisables. Leur flexibilité par l'introduction de couches spécifiques en apprentissage profond, alliée à une procédure d'apprentissage intégrant la pondération (le choix) des variables comme de leurs interactions peuvent les rendre très efficaces.

3 Exemples

Les réseaux de neurones étant des boîtes noires, les résultats fournis ne sont guère explicites et ne conduisent donc pas à des interprétations peu informatives du modèle. Seule une étude des erreurs de prévisions et, dans le cas d'une régression, une étude des résidus, permet de se faire une idée de la qualité du modèle.

3.1 Cancer du sein

La prévision de l'échantillon test par un réseau de neurones conduit à la matrice de confusion ci-dessous et donc une erreur estimée de 3%.

```
        benign malignant
```

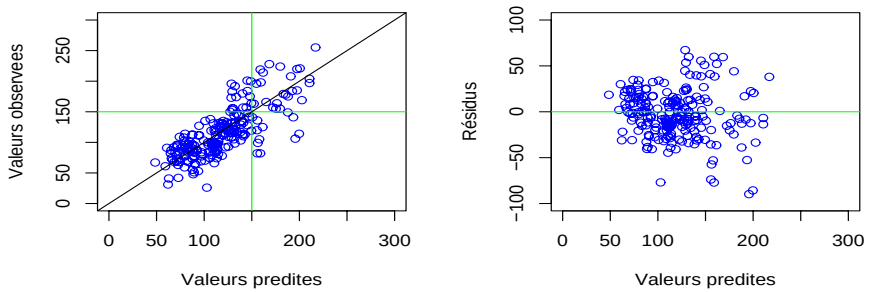


FIGURE 3 – Ozone : Valeurs observées et résidus de l'échantillon test en fonction des valeurs prédites par un réseau de 10 neurones

```
FALSE      83          1
TRUE        3         50
```

3.2 Concentration d'ozone

La comparaison des résidus (figure 3 montre que le problème de non-linéarité qui apparaissait sur les modèles simples (MOCAGE, régression linéaire) est bien résolu et que ces résidus sont plutôt moins étendus, mais le phénomène d'hétéroscédasticité est toujours présent quelque soit le nombre de neurones utilisés. Il a été choisi relativement important (10) et conduit donc à un bon ajustement ($R^2 = 0,77$) mais devra être réduit pour optimiser la prévision.

L'optimisation des paramètres d'un réseau de neurones est instable comme pour les proches voisins car chaque exécution de l'estimation de l'erreur par validation croisée fournit des résultats différents. Elle est en plus très compliquée par le nombre de paramètres à optimiser : nombre de neurones sur la couche (*size*), pénalisation (*decay*), nombre d'itérations. Une fonction de la librairie `e1071` permet de faire varier à la fois la taille et la pénalisation et fournit des graphiques élégants (figure 4) mais les exécutions sont très longues et les résultats pas toujours pertinents. Le plus efficace semble être de fixer

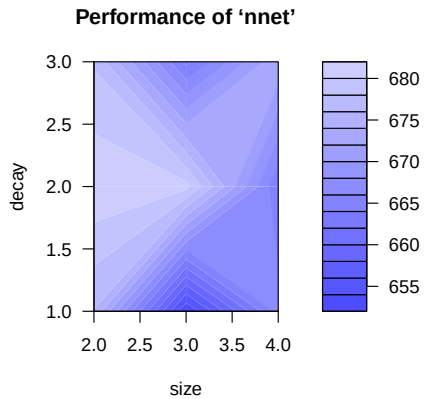


FIGURE 4 – Ozone : optimisation des paramètres (pénalisation et nombre de neurones) par validation croisée.

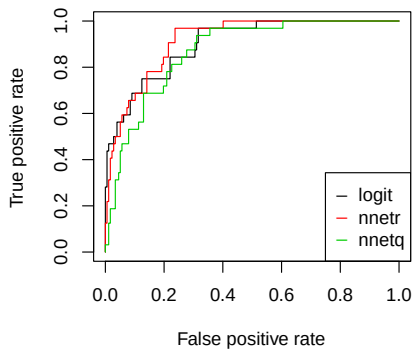


FIGURE 5 – Ozone : courbes ROC pour la régression logistique et les réseaux de neurones.

"assez grands" la taille (nombre de neurones) et le nombre d'itérations pour se focaliser sur le seul réglage de la pénalisation.

Comme pour les arbres de décision, les réseaux de neurones ne proposent pas de modèles très efficaces sur cet exemple. Les taux d'erreur de prévision du dépassement du seuil sont de 14,4% à partir du modèle quantitatif et de 15,6% avec une prévision directement qualitative. Les courbes ROC estimées sur l'échantillon test permettent de comparer les méthodes. Dans ce cas et pour l'échantillon test concerné, la méthode la plus efficace (figure 5) pour prévoir le dépassement du pic d'ozone est un réseau de neurone modélisant la concentration plutôt que la prévision directe du dépassement (logit ou réseau qualitatif).

3.3 Données bancaires

Une fonction de la librairie `e1071`, pratique mais très chronophage, propose une automatisation de l'optimisation des paramètres (*decay*, nombre de neurones). Elle produit une carte de type contour permettant d'évaluer "à l'œil" les valeurs optimales. La prévision de l'échantillon test par ce réseau de neurones conduit à la matrice de confusion ci-dessous et donc une erreur estimée de 21,5% :

```
pred.vistest FALSE TRUE
       FALSE   110   16
       TRUE     27   47
```

4 Introduction à l'apprentissage profond

Les techniques associées sont simplement introduites dans ce document, elles sont développées dans celui associé au cours de *Statistique en grande dimension*.

4.1 Préambule

Pendant les années 90s et le début des années 2000, le développement de l'apprentissage machine s'est focalisée sur les algorithmes de machines à vecteurs supports et ceux d'agrégation de modèles. Pendant une relative mise en

veilleuse du développement de la recherche sur les réseaux de neurones, leur utilisation est restée présente de même qu'une veille attendant le développement de la puissance de calcul et celle des grandes bases de données, notamment d'images.

Le renouveau de la recherche ans ce domaine est dû à Yoshua Bengio et Yan le Cun qui a tenu à jour un célèbre site dédié à la reconnaissance des caractères manuscrits de la base MNIST. La liste des publications listées sur ce site témoigne de la lente progression de la qualité de reconnaissance, de 12% avec un simple perceptron à 1 couche jusqu'à moins de 0,3% en 2012 par l'introduction et l'amélioration incrémentale d'une couche de neurones spécifique appelée *convulational neural network* (ConvNet). L'étude de ces données qui ont servi de benchmark pour la comparaison de très nombreuses méthodes sert maintenant de données jouet pour beaucoup de tutoriels des environnements dédiés (*tensorFlow, Keras, pyTorch, caffe...*)

Schématiquement, trois grandes familles de réseaux d'apprentissage profond sont développées avec des ambitions industrielles.

convolutional neural networks (ConvNet) pour l'analyse d'images.

long-short term memory (LSTM) lorsqu'une dimension temporelle ou plus généralement des propriétés d'autocorrélation sont à prendre en compte pour le traitement du signal ou encore l'analyse du langage naturel.

autoEncoder decoder ou réseau *diabolo* en apprentissage non supervisé pour, par exemple, le débruitage d'images ou signaux, la détection d'anomalies.

Seul le premier point est développé pour illustrer les principaux enjeux.

4.2 Reconnaissance d'images

Cette couche de neurones (ConvNet) illusrtrée par la figure 6 ou plutôt un empilement de ces couches introduit des propriétés spécifiques d'*invariance par translation*. Ces propriétés sont indispensables à l'objectif de reconnaissance de caractères et plus généralement d'images qui peuvent être vues sous des angles différents. C'est dans ce domaine que les résultats les plus spectaculaires ont été obtenus tandis que l'appellation *deep learning* était avancée afin d'accompagner le succès grandissant et le battage médiatique associé.

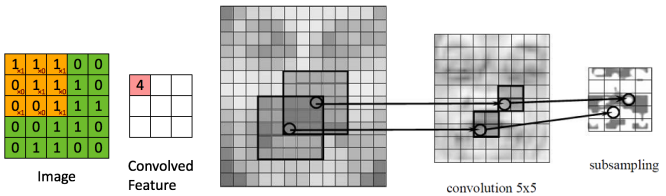


FIGURE 6 – *Principe élémentaire d'une couche de convolution et application à une image.*

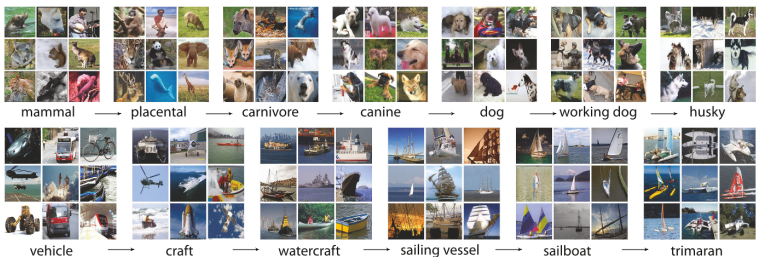
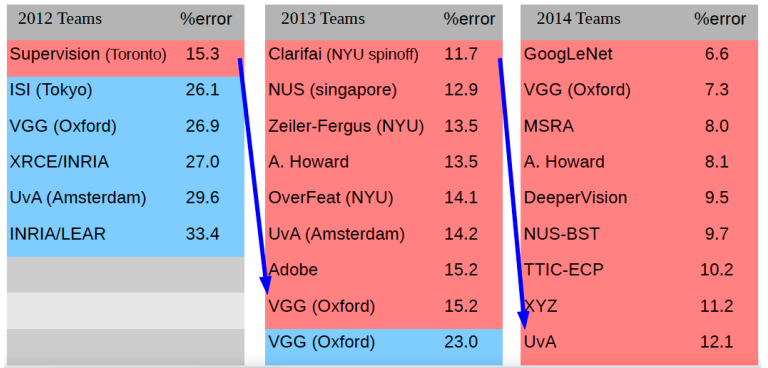


FIGURE 7 – *Échantillon de la base ImageNet.*

La communauté de reconnaissance d'images (figure 7) se confronte chaque année depuis 2010 sur une jeu de données issues d'une base d'images labellisées : 15 millions d'images, 22000 catégories hiérarchisées. De cette base sont extraites 1,2 milions d'images pour l'apprentissage avec 1000 catégories. Les participants au concours doivent prévoir la catégorie de 15000 images de l'échantillon test. Ce projet à l'initiative de l'Université Stanford est largement sponsorisé par Google.

Comme pour les données de reconnaissance de caractères, une progression largement empirique à conduit à l'introduction et au succès d'un réseau empilant des couches de neurones aux propriétés particulières. Cette progression est retracée dans le tableau 8. C'est en 2012 qu'une équipe utilise pour la



2012 Teams	%error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

2013 Teams	%error
Clarifai (NYU spinoff)	11.7
NUS (singapore)	12.9
Zeiler-Fergus (NYU)	13.5
A. Howard	13.5
OverFeat (NYU)	14.1
UvA (Amsterdam)	14.2
Adobe	15.2
VGG (Oxford)	15.2
VGG (Oxford)	23.0

2014 Teams	%error
GoogLeNet	6.6
VGG (Oxford)	7.3
MSRA	8.0
A. Howard	8.1
DeeperVision	9.5
NUS-BST	9.7
TTIC-ECP	10.2
XYZ	11.2
UvA	12.1

FIGURE 8 – Classements successifs (Le Cun 2016) des équipes participant au consours ImageNet. En rouge, celles utilisant des neurones profonds.

première fois un réseau de neurones profond contrairement à des traitements spécifiques et ad'hoc de l'analyse d'images utilisées jusque là. L'amélioration était telle que toutes les équipes ont ensuite adopté cette technologie pour une succession d'améliorations empiriques. En 2016 une équipe propose un réseau à 152 couches et atteint un taux d'erreur de 3%, mieux que les 5% d'un expert humain.

Ce concours est depuis lors abandonné au profit de problèmes plus complexes de reconnaissance de scènes associant plusieurs objets ou thèmes.

4.3 Couches pour l'apprentissage profond

Construire un réseau d'apprentissage profond consiste à empiler des couches de neurones aux propriétés spécifiques rapidement résumées ci-dessous. Le choix de du type, de l'ordre, de la complexité de chacune de ces couches ainsi que du nombre est complètement empirique et l'aboutissement de très nombreuses expérimentations nécessitant des moyens de calculs et bases de données considérables.

fully connected Couche classique de perceptron et dernière couche d'un réseau profond qui opère la discrimination finale entre par exemple des images à reconnaitre. Les couches précédentes construisant, extrayant, des caractéristiques (*features*) de celles-ci.

convolution opère une convolution sur le signal d'entrée en associant une réduction de dimension (cf. figure 6).

pooling réduction de dimension en remplaçant un sous-ensemble des entrées (sous-image) par une valeur, généralement le max.

normalisation identique au précédent avec une opération de centrage et / ou de normalisation des valeurs.

drop out les paramètres estimés sont les possibilités de supprimer des neurones d'une couche afin de réduire la dimension.

...

4.4 Utilisation rudimentaire

Sans bases de données très volumineuse et moyens de calcul substantiels il est illusoire de vouloir apprendre un réseau profond impliquant l'estimation de millions de paramètres. Une mise en œuvre simple sur des données spécifiques consiste à :

- Identifier un réseau ou modèle existant appris sur des données similaires. Pour les images, considérer par exemple les versions des réseaux *inception* de *tensorFlow* ou *AlexNe*t de Caffe.
- Supprimer la dernière couche du modèle dédiée à la classification,
- Apprendre les poids de cette seule dernière couche sur les données spécifiques.